CRAY 2

COMPUTING SYSTEM

December 20, 1982

Table of Contents

Introduction

The Cray-2 computing system is an evolution of the Cray-1 system and provides an order of magnitude performance improvement at a comparable price. There are several factors which make this possible.

1) Large Semiconductor Memory

The semiconductor industry has made giant strides forward in the development of large integrated circuit memory chips in the decade since the Cray-1 computer was designed. A million words of direct access memory was the largest memory practical at the time of the original Cray-1 design. This was later expanded to four million words as larger memory circuits became available. This size is the limit that addressing allows in a Cray-1 system.

The Cray-2 system contains a direct access memory of 256 million words. This increase of almost two orders of magnitude changes the way that large scientific problems can be solved.

2) Multi-processors

The Cray-2 system contains four independent background processors each somewhat faster than a Cray-1 computer. These four processors can be brought to bear on a single large computation using the common 256 million word memory. Computation is time interlocked with high speed semaphore flags which control the step advances during the computation.

3) Integrated Foreground Processing

The Cray-2 system brings control of the peripheral support equipments into the main frame hardware. A single foreground processor coordinates the data flow into and out of the system common memory. The synchronous operation of this foreground processor with the four background processors allows an order of magnitude increase in data throughput over the original Cray-1 system.

4) Liquid Immersion Cooling

The Cray-2 system operates in a small liquid filled main frame cabinet. The circuits are cooled by direct liquid contact with the integrated circuit packages. This liquid immersion technology allows greatly reduced physical size and higher speed computation than is possible with previous techniques.

Hardware Design

The Cray-2 hardware is constructed of synchronous networks of binary circuits. These circuits are packaged in 320 pluggable modules. The modules each contain 750 integrated circuit packages. Total integrated circuit population in the system is 240000 units of which 75000 are memory.

The pluggable modules are three dimensional structures with an array of circuit packages 8 by 8 by 12 units. There are eight printed circuit boards which form the module structure. Circuit interconnections are made in all three directions within the module. External dimensions of the module are one inch by four inches by eight inches. One end of the module contains a circuit connector which mates with a connector in the cabinet frame. This connector has 288 pairs of pins for twisted pair wire communication between the modules in the cabinet frame.

Modules are arranged in the cabinet frame in 14 columns each 24 modules high. The columns are arranged in a portion of a circle with a 20 degree angle between columns. An inert electronic liquid circulates in the cabinet frame and flows through the module circuit boards across the four inch surface. Liquid velocity is one inch per second through the modules. Total module column height is 24 inches.

The semicircle of module columns is located on top of a similar structure containing power supplies for the system. Total power consumption for the system is 180 kilowatts. Total cabinet height including the power supplies is 43 inches.

There are 20 types of integrated circuit packages used in the logical networks of the machine. The circuits consist of emitter coupled logic gates with a maximum gate width of six inputs. Total gate capacity of the circuit packages is 16. Most of the 20 types of circuit packages contain two levels of gates within the package. The package has 16 connecting pins.

A 250 megahertz oscillator controls the timing throughout the circuit modules in the machine. The oscillator signal is transmitted as a square wave over 120 ohm twisted pair wires to each of the module connectors. Wire lengths are controlled so that the travel time to the individual modules is accurate within 100 picoseconds. The oscillator square wave is delivered to each individual circuit package within the module. An 800 picosecond pulse is formed from the square wave to gate data into register latches within the packages. This 800 picosecond strobe pulse occurs simultaneously throughout the machine with a period of 4 nanoseconds. This time is referred to as the machine clock period.

Common Memory

The Cray-2 system common memory consists of 128 storage banks of two million words each. Each word consists of 64 data bits and eight error correction bits. This memory is shared by the foreground processor, background processors and peripheral equipment controllers. It contains program code for the background processors as well as data for problem solution. System tables are located here for the foreground processor but foreground program code is not. Data buffers for the disk files are located directly in the background job data fields in this memory.

Each two million word memory bank occupies one Cray-2 circuit module. There is an independent data path from each bank to each of four memory access ports. A background processor and a foreground communication channel are associated with each memory port. Total memory bandwidth is 64 gigabits per second. Total memory capacity is 17 gigabits. Each background processor can read or write a word of data per clock period in a vector mode.

The integrated circuits used in the common memory contain 256 thousand bits of data. The bank of memory consists of a 8 by 8 by 9 array of these circuits in a three dimensional package. The memory bank module then contains 576 of these memory circuits and 192 logic circuits to support the memory access paths. Memory access time at the circuit level is 100 nanoseconds. Memory cycle time is 160 nanoseconds.

The Cray-2 logic circuits are significantly faster than the memory circuits. This speed discrepancy is used to minimize the number of physical data paths that are necessary to connect the 128 memory bank modules to the four memory ports. Data is transported between memory bank and access port in a three clock period long packet of data. The packet is 24 bits wide. A similar packet four clock periods long is used between access port and memory bank. In this case the memory bank address precedes the write data packet.

An eight bit error correction code is generated at the memory access port as the write data is transmitted to the memory bank. This code is interpreted at a readout port for single error correction and double error detection.

Background Processing

The four background processors in the Cray-2 system are composed of processing elements similar to those in a Cray-1 processor. The Cray-2 instruction set includes similar registers and arithmetic functions. Incompatibilities exist in order to allow the Cray-2 to address the large common memory.

Computation in a background mode implies a memory to memory computation. A job is initiated by the foreground processor. Data is moved from the disk files to the semiconductor memory under foreground processor supervision. The program code for the background processors is positioned in the common memory and the computational field is defined. The foreground processor then initiates the background computation using one or more background processors as required. The background processors may call for further peripheral activity through the foreground processor as the computation proceeds.

Each background processor has a small high speed local memory to hold scalar operands during the computation. Data is moved from the common memory to the local memory and returned at the end of each computation. Arrays of data are addressed by the background processors directly in the common memory. The access to common data is interlocked by the background processor semaphore flags.

Resources of a background processor are summarized below.

Eight S registers - 64 bit length - For scalar operands
Eight A registers - 32 bit length - For address and integer operands
Eight V registers - 64 elements of 64 bits - For vector segments
Instruction buffer - 512 parcels of 16 bit length - For instruction loops
Local memory - 1024 words of 64 bit length - for scalar or vector backup
Floating point functional units - For computation in 64 bit floating point mode
Integer functional units - For computation in 64 bit integer mode
Address functional units - For computation in 32 bit integer mode

Foreground Processing

The foreground processor supervises overall system activity and responds to requests for interaction between the system members. System communication is accomplished through four high speed synchronous data channels. These channels interconnect the background processors, foreground processor, disk control units, and host system interfaces.

Instructions for execution in the foreground processor are loaded into a special memory at system dead start time from a diskette at the maintenance console. This memory then becomes a read only memory during system operation. Data for supervision of the system is maintained in the common memory and is moved to a foreground processor local data memory as required.

The majority of foreground processor activity involves data transfer between the disk file storage units and the common semiconductor memory. The system has provision for 40 disk file units. Control for these units is organized into groups of four units each. Group control allows the high speed circuits of the Cray-2 to time share the supervision of the slower speed disk units.

All disk files may read or write data at the same time. Disk files may be addressed as individual storage units, or they may be tied in synchronous groups to provide higher transfer rates than the 35 megabit per second rate of the individual disk units. Such synchronous grouping is of arbitrary size and may extend from two units to the entire system complement. Grouping is under the control of the foreground processor which decides the mapping and order of disk sectors into a common buffer.

Resources of the foreground processor are summarized below.

Eight A registers - 32 bit length - Integer operands
Local data memory - 2048 words of 32 bit length - Temporary storage
Instruction memory - 32,768 parcels of 16 bit length
Integer functional units - For computation in 32 bit integer mode
Four communication channels - 4 gigabits per second each
Capacity of 40 DD29 disk files - 200 gigabits

****

FOREGROUND SYSTEM

****

System Dead Start

The Cray-2 system is initialized from the maintenance control console. This control console is a microprocessor with diskette storage. The microprocessor is connected to the Cray-2 cabinet over an eight bit low speed communication path. The dead start process involves loading a special diskette with the proper microprocessor program to sequence the initialization.

The process begins with a control signal from the maintenance console to preset the foreground control processor in a instruction memory load mode. The data for the foreground instruction memory is then transmitted from the diskette to the foreground processor.

The background processors and common memory are forced into an idle mode during the loading of the foreground processor memory. Foreground processor execution then begins at address zero in the foreground memory. The preamble of this program code clears common memory of noise data and presets nominal conditions in the background processors.

The foreground processor then begins its normal scan of channel activity. A job may be initiated by a host system interface or by a maintenance console request for diagnostic activity.

Host System Interface

The Cray-2 system is intended to be connected into a host system. The host system would contain the data concentration and communication equipment that is appropriate for a large computational facility.

One or more wide bandwidth data paths should connect the host system equipment to the Cray-2 cabinet frame. Bandwidth is likely limited by the host system and configurations will vary depending on the type of front end equipment used.

Each data path from the host system to the Cray-2 system requires an interface module in the Cray-2 cabinet frame. Provision is made for a number of these interface modules. These modules perform the function of resynchronizing data from devices with a different clock period and buffering the data as appropriate between the two systems.

Bandwidth for each interface module is limited to one gigabit per second by the internal structure of the Cray-2 system.

An alternative to the wide bandwith data path described above is shared disk file storage. The DD29 disk files have dual access ports. One or more disk file units can be designated as shared facilities and data can move between host system and Cray-2 system through this media. Disk file data format is the same as in the Cray-1 system.

DD29 Disk Storage Unit

The Cray-2 system can support a number of DD29 disk storage units. These units are manufactured by Control Data Corporation and are currently used in Cray-1 systems.

Each DD29 disk storage unit has a capacity of 4.8 gigabits of data. This data is recorded on 40 surfaces which rotate at 3600 revolutions per minute. The rotational latency is then 16.6 milliseconds.

The data is recorded and read back from the disk surfaces by forty recording heads which can be positioned axially on the disk surface. These heads are connected in ten groups of four recording heads each. Data from the four heads in one group can be read concurrently into the disk controller.

The recording heads can be positioned on 822 tracks for each disk surface. All heads move together in the positioning process. Moving to an adjacent track requires 15 milliseconds. Moving the maximum distance requires 80 milliseconds.

Each recording track on the disk surface is divided into 18 sectors. Each sector contains 512 common memory words of data plus error detection and correction information. The position of the heads on the disk surface is verified by data recorded and read back in each sector. The data within the sector is self correcting through a Fire code polynomial generator.

Data transfer rate using the four recording heads in parallel is 35.5 megabits per second. Consecutive sectors of data may be read or written with alternating data buffers in the disk controller. The head group selection may be changed as data is moving for the last sector on a disk track. The first sector on the next track may then be processed without missing a disk revolution. The total data package which may be moved from the disk surface without positioning the recording heads is then 180 sectors. This package of data is called a disk cylinder.

Disk System Organization

The Cray-2 system can include up to 40 DD29 disk file storage units. These units can be addressed as individual storage devices as they are in a Cray-1 system. There are problems with this approach which the large common memory of the Cray-2 can avoid. These problems are the data transfer rate for individual files, the rotational latency of the disk units, and the reliability of mechanical devices.

The disk storage system on a Cray-2 has the option of operating in a synchronous mode with all disk units running in parallel in a lock step mode. For this approach to look attractive the buffer size for individual disk references must be of the order of 100,000 words. This was not practical in smaller memory machines.

Consider a system configuration with 16 DD29 disk files as an example for the synchronous mode of operation. The foreground processor is given a disk address which consists of a pseudo-track number. This is the cylinder and head group for a disk file with no flaws. A table lookup converts this pseudo-track into a physical track for each disk unit. All disk units are positioned in parallel.

The foreground processor reads angular position for each disk surface to determine the sector currently under the recording head. It then begins a data stream from common memory to disk surfaces choosing the portion of the common memory buffer appropriate for the current angular position of each disk unit. Data to 15 of the disk units is directly from the common memory job buffer. Data for the 16th disk unit is a logical difference data stream using the word by word data from the desired file. All 16 disk units write one track of data as the basic reservation unit.

On data readback the 16th disk is read concurrently with the other 15 disks. If the Fire code detectors indicate no data errors the 16th disk data is discarded. If an error has occurred it can be corrected without time loss in the data stream.

The overhead introduced by this arrangement is one disk file on fifteen. The benefit is three-fold.

1) Data rate is 525 megabits per second instead of 35 megabits per second.

2) The disk file rotational latency has gone to zero.

3) A disk file may fail completely due to head crash or motor failure with no loss of data or time.

A disk failure in this system can be corrected during system operation by removing the defective file and replacing it with another unit. The new unit can then be brought on line by running a background job which takes 2.5 minutes of disk system time to record the faulty unit data from the data on the other 15 files.

Foreground Communication Channels

There are four communication channels in the system which link the background processors, foreground processor, and peripheral equipment controllers. Most of the data traffic is between controllers and memory. Data blocks are generally 512 common memory words in length. Typical channel reservation time is ten microseconds.

Each channel has associated with it one common memory access port and one background processor control. The foreground processor is associated with all four channels in a supervisory role. Normally four peripheral controllers are associated with each channel.

Each channel consists of 16 data paths and two control paths. One control path carries a channel busy flag and the other a data ready flag. These 18 paths interconnect the members of the communication group in a continuous loop. Each member receives data on 18 bit paths each clock period and transmits that data onward to the next member in the following clock period. Data may then move about the loop from any transmitting member to any receiving member.

Channel Sequence for Function Initiation

An idle channel is indicated by a zero bit moving about the channel busy path. The data present flag and 16 data bits will also be zero values at this time. Activity is initiated for the channel by the foreground processor. The foreground processor breaks the channel loop at its node and transmits a function code on to the next member of the loop. The busy flag is set by the foreground processor and remains set as the function code moves around the loop.

The function code may be followed by a parameter for the addressed member of the communication group. The function code and parameter data are each 32 bits in length. They are encoded into four 16 bit parcels as they leave the foreground processor node.

Each member of the channel group tests for a busy flag transition from zero to one state. This transition indicates a message is passing the node from the foreground processor. Each member translates the function code to determine if action is required. If this member is addressed, and a response is required, the response is transmitted in the two clock periods following the parameter data.

The foreground processor, having initiated the function transmission, waits for the zero to one transition on its busy flag input path. It then skips the two clock periods corresponding to its own parameter data and reads the two following parcels of response data. If this completes the communication for the function, the foreground processor clears the busy flag, waits for the return transition, and the channel becomes idle.

Disk Controller

A disk controller supervises the activities of four DD29 disk files. This controller interprets function codes sent by the foreground processor over the communication channel. The controller determines its own channel call number by wiring connections in the cabinet frame.

The function code transmitted from the foreground processor on the channel link consists of 32 bits plus an optional 32 bits of parameter data. The basic 32 bit format is interpreted by the controller circuits as follows.

4 bits) Controller number
4 bits) Unit number
8 bits) Function code
16 bits) Next program address

A zero controller number is interpreted as a general call for foreground request.

Function codes are translated in the following manner.

00) Release unit
01) Reserve unit
02) Clear fault flags
03) Return to zero cylinder
04) Select margin conditions
05) Read sector number
06) Read error flags
07) Read disk status

10) Select cylinder
11) Select head group
12) Read status response
13) Enter error call address

20) Begin normal read at specified sector
21) Read correction code
22) Begin early read at specified sector
23) Begin late read at specified sector
24) Read format data in short block

30) Begin normal write at specified sector
31) Write format data

------ Function 20 - Begin normal read at specified sector

This foreground processor function call begins the process of streaming data from the disk surface to the common memory. The length of the data stream may be as short as one sector of disk data or as long as an entire disk cylinder. The steps in the data transfer are as follows.

1) Foreground to disk - Begin read at sector

The disk controller reads the sector number from the parameter word and enters it in the appropriate disk unit coincidence register. The next program address is held for call to foreground on completion.

2) Disk to foreground - Sector data ready

The disk controller senses a full sector data buffer from the sector read operation. It then responds to the next channel call with the program address from step 1.

3) Foreground to memory - Write block at address

The memory port senses this function call and prepares its input buffer for a block of data intended for the common memory address in the parameter word.

4) Foreground to disk - Transmit sector data

The foreground processor holds the channel busy flag on this call in preparation for data transmission. The disk controller sends a burst of 64 common memory data words around the loop to the memory port. A data ready flag accompanies each element. The disk controller then pauses to wait for a memory response.

5) Memory to disk - Buffer free

The memory port loads a 64 word buffer with the data from the channel. When the buffer data has been moved into common memory the memory port sends a single data ready pulse to the disk controller.

6) Disk to memory - Data burst

The disk controller receives the data ready signal and sends the next 64 word burst of data over the channel. Steps 5 and 6 repeat until the sector data transmission is complete. The disk controller then senses the last data ready pulse from the memory port and drops the channel busy flag.

Steps 2 through 6 repeat until the foreground processor requirement has been satisfied.

------ Function 30 - Begin normal write at specified sector

This foreground processor function call begins the process of streaming data from the common memory to the disk surface. The length of the data stream may be as short as one sector or as long as an entire disk cylinder. The steps in the data transfer are as follows.

1) Foreground to disk - Begin write at sector

The disk controller reads the sector number from the parameter word and enters it in the appropriate disk unit coincidence register. The program address is held for call on completion.

2) Foreground to memory - Read block from address

The memory port reads the common memory address from the parameter word and loads its 64 word buffer from that location. The channel busy flag remains set during this operation. The memory port then transmits a 64 word burst of data over the channel to the disk controller. A data ready flag accompanies each element.

3) Disk to memory - Buffer free

The disk controller loads the first 64 words of its sector data buffer with the data burst from the memory port. It then sents a data ready pulse to the memory port.

4) Memory to disk - Data burst

The memory port reads the next 64 words from the common memory and transmits the next data burst to the disk controller. Steps 3 and 4 repeat until the disk sector buffer is full. The disk controller then drops the channel busy flag which frees the channel for other use.

5) Disk to foreground - Sector data written

The disk controller moves the sector data to the write deskewing buffer for the appropriate disk unit. It begins recording on the disk surface as soon as the proper sector is under the recording head. It then responds to the next channel call with the program address from step 1.

Steps 1 through 5 repeat until the foreground processor requirements are satisfied.

Foreground Processor Instruction Summary

```
000---  Pass
001---  Pass
002i--  Enter Ai with console data
003i--  Enter console data from Ai

004i-k  Enter Ai from local memory address Ak
005i-k  Store Ai into local memory address Ak
006ijk  Enter local memory address Ai from common address Aj with length Ak
007ijk  Store local memory address Ai into common address Aj with length Ak

010ijk  Jump to Aj and hold return Ai if Ak is zero
011ijk  Jump to Aj and hold return Ai if Ak is nonzero
012ijk  Jump to Aj and hold return Ai if Ak is positive
013ijk  Jump to Aj and hold return Ai if Ak is negative

014ijk  Jump to Aj and hold return Ai if channel k is busy
015ijk  Jump to Aj and hold return Ai if channel k is idle
016ij-  Jump to Aj and hold return Ai if console data ready
017ij-  Jump to Aj and hold return Ai if console data not ready

020ijk  Enter Ai with positive jk
021ijk  Enter Ai with negative jk
022i--  Enter Ai with real time count
023---  Unassigned

024ijk  Shift Ai left jk
025ijk  Shift Ai right jk
026ijk  Enter Ai with integer sum Aj + Ak
027ijk  Enter Ai with integer difference Aj - Ak

030ijk  Enter Ai with logical product Aj and Ak
031ijk  Enter Ai with logical product Aj and complement Ak
032ijk  Enter Ai with logical difference Aj and Ak
033ijk  Enter Ai with logical sum Aj and Ak

034--k  Abort channel k
035ijk  Function Aj on channel k with response Ai.  Clear busy.
036ijk  Function Aj on channel k with parameter Ai.  Clear busy.
037ijk  Function Aj on channel k with parameter Ai.  Hold busy.

040i--  Enter Ai with positive 16 bit constant
041i--  Enter Ai with negative 16 bit constant
042i--  Enter Ai with 32 bit constant
043i--  Unassigned

044i--  Enter Ai from constant local memory address
045i--  Store Ai into constant local memory address
```

****

BACKGROUND PROCESSOR

****

General Description

The background processors are intended for floating point computation in a 64 bit mode. Significant speed improvements occur when the computation can be organized into vector streams. The structure of the background processors is designed to enhance the vector capability of the system. Resources of a background processor can be summarized as follows.

Vector registers (V)

There are eight vector registers in a background processor. Each register contains 64 words of 64 bit length. These registers are used as a computational way station for data between the memory and the functional units.

Scalar registers (S)

There are eight scalar registers in a background processor. Each register contains a single 64 bit word. These registers are used to support the vector registers in vector streaming where one element of the computation is a constant value. The scalar registers are used as computational way stations between the memory and the functional units where vector implementation of the work is not possible.

Address registers (A)

There are eight address registers in a background processor. Each register contains a single 32 bit word. These registers are use to calculate memory locations for both the local memory and the common memory. They are also used to compute integer results in a 32 bit mode.

Local memory

There are 1024 words of local memory associated with each background processor. Each word is 64 bits in length. This memory is treated as a register file to hold scalar operands during a computation period and then return the data to the common memory.

Instruction stack

Each background processor has an instruction stack to allow program loops to execute without additional common memory references. This instruction stack contains a contiguous field of 512 parcels of program code. Programs may loop within this field using any of the branch instructions.

Floating point functions

Each background processor has dedicated functional units to perform calulations in a floating point format. There are two independent units in each processor with functions assigned in two groups as follows.

Floating point multiplication
Reciprocal approximation
Reciprocal iteration

Floating point addition
Floating point subtraction
Float to fixed point conversion

Integer functions

Each background processor has two dedicated functional units to perform fixed point calculations in a 64 bit mode. These units are similar but are specialized, one for scalar use and one for vector use. These functions may then be performed concurrently. Each unit has the following modes.

Long integer addition
Long integer subtraction
Long word data shift
Population count
Leading zero count

Logical functions

Each background processor has two dedicated functional units to perform logical operations on 64 bit data in a bit by bit mode. These units are similar but are specialized, one for scalar use and one for vector use. These functions may then be performed concurrenly. Each unit has the following modes.

Bit by bit logical product
Bit by bit logical different
Bit by bit logical sum

Address functions

Each background processor has 32 bit functional units for address or short integer computation. These units operate in a scalar mode only with the following functions.

Integer addition
Integer subtraction
Integer product

Floating Point Data Format

Floating point computation is performed on data in a special packed format within a 64 bit word. This format is generally referred to as 'sign and magnitude' to distinguish it from other forms in common use. The 64 bits of the data word are structured into three fields as listed below from highest order to lowest order bit position.

1 bit) Sign of coefficient
15 bits) Biased binary exponent
48 bits) Fractional coefficient

The numerical value of the floating point data is determined by multiplying the coefficient times a base 2 with a signed binary exponent.

---- Coefficient

The 48 bit coefficient is a binary number with the binary point to the left of the highest order bit position. The highest order bit in the coefficent is normally a one value. The floating point number is said to be normalized if this is the case. Correct results in some functional units depend on normalized operands. The range of decimal values for the coefficient in a normalized form is 0.5 to 1.0.

---- Exponent

The 15 bit exponent is a binary number with a bias to allow integer testing of floating point data. An octal value of 40000 represents a zero exponent. A larger value represents a positive exponent and a smaller value represents a negative exponent. A floating point representation of the integers zero, plus one, and minus one in a normalized form are then as follows in an octal form for each of the three fields.

Zero:	0 00000 0000000000000000
Plus one:	0 40001 4000000000000000
Minus one:	1 40001 4000000000000000

Exponent values of 60000 and greater are considered to have overflowed the exponent range. Hardware tests are performed for these values to indicate floating point range error. Exponent values less than 20000 are considered to have underflowed the floating point range. Such values are treated in many cases as if they had a zero value. There is no hardware indication when a computation underflows the floating point range.

Fixed Point Data Format

Fixed point computation is performed on data in a twos complement integer format. Long word arithmetic is performed with 64 bit integers. A fixed point representation of the integers zero, plus one, and minus one in this format are illustrated below using octal notation.

Zero:	0000000000000000000000
Plus one:	0000000000000000000001
Minus one :	1777777777777777777777

Address computation and short integer computation are also performed in a twos complement integer format. In this case the arithmetic is performed with 32 bit integers. A fixed point representation of the integers zero, plus one, and minus one in this format are illustrated below using octal notation.

Zero:	00000000000
Plus one:	00000000001
Minus one:	37777777777

Instruction Format

The background processors translate instruction code in 16 bit parcels of data. These parcels are packed four per word in the common memory. The parcels are addressed as if the common memory had four times as many locations and the data were 16 bits long. A branch instruction to a parcel location out of the buffer area of the instruction stack results in a common memory reference. In this case the common memory address is formed in the instruction fetch hardware by shifting the instruction parcel address down by two bit positions.

Instruction translation involves the interpretation of four designators within the 16 bit parcel. These designators are identified below from higher to lower order bit position.

f designator - 7 bits
i designator - 3 bits
j designator - 3 bits
k designator - 3 bits

The f designator determines the instruction type. The f designator values are indicated in octal notation in the instruction descriptions. The i, j, and k designators generally refer to V, S, or A registers in a three address format. The i designator generally specifies the destination register for the functional computation. The j and k designators generally specify the source operands.

Constants are entered into the operating registers from the instruction stream. These constant values appear in parcels following the instruction which refer to them. There may be one, two, or four parcels of constant data depending on the specific instruction. The first parcel in a multiparcel group is always the highest order portion of the resulting constant.

Single parcel constants are used to address the local memory. Two parcel constants are used to address the common memory. Four parcel constants are used to enter long word values in the S registers.

Vector Registers (V)

There are eight vector registers in each background processor. Each register contains 64 words of 64 bit length. These registers are used as a computational way station for data between the memory and the functional units.

The instruction issue control mechanism reserves the vector registers which are involved in a functional unit streaming operation. This may be one, two, or three vector registers depending on the specific instruction. The functional unit is reserved at this same time. The instruction sequence may then proceed to the next instruction and initiate concurrent activity as long as the resources reserved are not required in the subsequent instruction.

The length of the vector stream is determined by the content of a six bit length register (L). This register is preset before the vector instruction issues and may be used for a number of vector operations of the same length. Maximum vector length is 64 elements. A zero value in the L register is interpreted as a 64 bit length indicator.

The i, j, and k designators in a vector instruction may have the same value. In the case of identical source operands the data is streamed from the same vector register to both data paths. In the case of a destination register which is the same as the source register the vector register writing function takes priority over reading but with a delay equal to the transit time of the functional unit. This may be useful as a recursive vehicle for merging vector data in some situations.

The vector registers are implemented in the hardware with 16 by 4 register chips. These integrated circuit chips have a two clock period cycle time. One clock period is required for address soak. The second clock period is then used for readout or for write strobe.

The 64 elements of a vector register are arranged in four banks of 16 elements each. Each bank has its own address register but they share a common read/write register. Consecutive element addresses for vector data move through the four banks before advancing the pointer for the first bank. This allows data to stream into, or out of, a vector register in consecutive clock periods. Data coming from common memory and moving into a vector register may come out of order with respect to the common memory access requests. This is because of quadrant and bank delays in the common memory access control mechanism.

Scalar Registers (S)

There are eight scalar registers in a background processor. Each register contains a single 64 bit word. These registers are used to support the vector registers in vector streaming where one element of the computation is a constant value. The scalar registers are used as computational way stations between the memory and the functional units where vector implementation of the work is not possible.

The instruction issue control mechanism reserves a scalar register which is the destination register for a functional operation. This interlocks subsequent instructions which wish to use the result in this register as a source operand. Scalar registers used as the source for functional unit data are not reserved. The issue control mechanism tests the reservation flag at the time of issue for the instruction. If the register is free the data is immediately read and is free for the next instruction. Data is kept at the functional unit for those cases where a scalar operand is used in a vector streaming operation.

The eight scalar registers as a group have one data readout path and one data destination path. Each path may be used independently in each clock period. An instruction which requires two scalar source operands reads these operands in the two clock periods following instruction issue. The destination path must be reserved at issue time for the specific clock period of data arrival.

An exception to the destination path reservation described above is made for the common memory read to scalar register. A separate path from the common memory to the scalar registers as a group is provided because the arrival time cannot be known at issue time.

Address Registers (A)

There are eight address registers in a background processor. Each register contains a single 32 bit word. These registers are used to calculate memory locations for both the local memory and the common memory. They are also used to compute integer results in a 32 bit mode.

The instruction issue control mechanism reserves an address register which is the destination register for a functional operation. This interlocks subsequent instructions which wish to use the result in this register as a source operand. Address registers used as the source for functional unit data are not reserved. The issue control mechanism tests the reservation flag at the time of issue for the instruction. If the register is free the data is immediately read and is free for the next instruction. Data is kept at the functional unit for those cases where an address operand is used in a vector streaming operation.

The eight address registers as a group have one data readout path and one data destination path. Each path may be used independently in each clock period. An instruction which requires two address source operands reads these operands in the two clock periods following instruction issue. The destination path must be reserved at issue time for the specific clock period of data arrival.

Local Memory

There are 1024 words of local memory associated with each background processor. Each word is 64 bits in length. This memory is treated as a register file to hold scalar operands during a computation period and then return the data to the common memory.

The local memory is implemented in hardware with 256 by one register chips. These integrated circuit chips have a three clock period cycle time. The first clock period is used for address soak only. The second clock period is used for address soak plus chip select. The third clock period is used for data readout or for write strobe.

The local memory is organized into four banks of 256 words each. Each bank has its own address register. All four banks share a data write register and a data readout register. A scalar reference initiates all four banks at the same clock period. The proper bank is then sampled to the readout register three clock periods later for a read reference or the write strobe is enabled to the proper bank for a write reference.

Vector references to the local memory require one extra clock period for address setup. This is to allow an arbitrary starting point for the vector stream. The bank address registers then advance sequentially for the length of the vector stream. Data moves at the rate of one word per clock period in a vector mode.

The local memory is reserved at instruction issue time in the same manner as a functional unit. The reservation has a three clock period duration for a scalar reference. The reservation has a duration of four clock periods plus the vector length for a vector reference.

Instruction Stack

Each background processor has an instruction stack to allow program loops to execute without additional common memory references. This instruction stack contains a contiguous field of 512 parcels of program code. Programs may loop within this field using any of the branch instructions.

The instruction stack is implemented in hardware with 16 by 4 register chips. These integrated circuit chips have a two clock period cycle time. One clock period is required for address soak. The second clock period is then used for readout or for write strobe.

The instruction stack is organized into four banks of 32 words each. Each word is 64 bits in length. Each bank has its own address register. The four banks share a common data write register and a data readout register. Consecutive addresses in the instruction stack move through the four banks before advancing the pointer for the first bank. This allows the common memory data to enter the instruction stack 64 bits wide with a word each clock period.

A branch out of the instruction stack field initiates a fetch vector in the common memory. The common memory address is formed from the requested branch destination parcel address by a shift down of two bit positions. The fetch vector length is always 32 common memory words. The instruction stack is cleared and limit registers which represent the instruction stack parcel boundaries are reset to correspond with the new field. The arriving data is then entered in the first 128 parcel positions of the instruction stack.

Data is read from the instruction stack to the instruction issue control module one parcel at a time. A threshold test is made with each parcel readout to determine if the program sequence is nearing the end of the available data in the instruction stack. A new fetch vector is initiated when the readout enters the last 64 parcels of the available data.

The instruction stack limit registers are adjusted with each new fetch vector. The oldest 128 parcels of data are discarded as a full instruction stack advances.

Instruction Issue

Background instructions are translated in several steps and are allowed to issue sequentially by an instruction issue control mechanism. Instruction parcels are delivered one at a time from the instruction stack. These parcels are placed in a queue where the translation functions occur. The instruction issue control process involves checking the reservation flags for the registers and functional unit involved in the instruction sequence. The instruction parcel waits in the issue position in the queue until all required resources are free.

An instruction is considered to issue during the last clock period in which it resides in the instruction issue queue. That instruction parcel is then discarded and the next instruction parcel shifts forward in the queue. The instruction stack continues to fill the queue as the parcels shift forward.

The instruction issue mechanism makes reservations for the assigned resources during the clock period that the instruction issues. Tests for the next instruction can then be made against the updated reservation flags. This reservation and test process takes two clock periods for a complete interactive cycle. The maximum instruction issue rate is then one instruction every other clock period.

Constants are intermixed with instruction parcels in the instruction stream. The constant parcels are passed through the instruction issue queue without test. An instruction with a following constant parcel may then issue and the following parcel issue without additional delay. Multiparcel constants delay the next instruction issue by one clock period for each constant parcel beyond the first.

Detail control signals are sent from the issue control module to the register modules and functional unit modules in the two clock periods following issue. These signals are formed and transmitted concurrent with the translation and issue test of the following instruction parcel. Most instructions involve a two clock period time sequence of register selection and readout for source operands.

The issue control module sends the detail control signals to the destination register module at a later time. This delay corresponds to the transit time of the source operands through the functional unit. A shift register in the issue control module provides the memory vehicle for these destination tags.

The S registers have a single destination path for data entry into the eight member registers. The issue control mechanism must reserve this path in addition to the more obvious register reservation at issue time. The path reservation is for the one clock period of intended use.

The A registers have a single destination path for data entry into the eight member registers. This path must be reserved for an A register destination in the same manner as for the S registers.

Floating Point Multiply Unit

The floating point multiply unit performs floating point calculations for both scalar and vector instructions. The unit is reserved during the period of the vector stream for the case of a vector instruction sequence. Scalar use is locked out during this period. No reservation is made at the time of issue for a scalar floating multiply instruction. Successive scalar instructions may issue at the maximum rate.

The floating point multiply unit accepts vector element pairs at the rate of one pair per clock period. The source operands move through the unit, stopping at an internal register briefly for each clock period boundary. The transit time through the unit is a constant for each mode of operation.

---- Floating Point Product

The floating point multiply unit forms the product of two operands in floating point format and delivers a result in floating point format. If both operands are normalized the result will also be normalized. Instructions which use this sequence are the following.

124ijk Enter Si with floating product Sj * Sk
154ijk Enter Vi with floating product Sj * Vk
155ijk Enter Vi with floating product Vj * Vk

The product is formed by merging the 48 bit coefficients into a matrix of logical product circuits. All combinations of coefficient bits are formed in the same clock period. The resulting 48 x 48 bits are then summed using three input adder circuits. The exponents are added to form the result exponent. A one count correction to the exponent is required if the resulting coefficient must be shifted by one bit position for normalization.

The 48 x 48 matrix of logical product bits is truncated eight bit positions below the lowest order result coefficient bit. Round bits are added to this lower field to give an equal population of high and low round errors for random operands. A round bias will exist over narrow ranges of operands because of the one bit correction shift after the round operation.

The Sj operand is held in the floating multiply unit during the execution of the 154 instruction. This frees the Sj register for other use during the vector streaming period.

Several special cases are treated in floating point multiplication for operands out of range. These cases are as follows.

1) One or both operands have overflow exponent
2) Sum of operand exponents is an overflow
3) Sum of exponents is an underflow
4) Both exponents are underflow

Cases 1 and 2 cause a floating point error signal to the background processor status register. The computational result delivered to the destination register is forced to an overflow value. Case 3 results in an all zero word sent to the destination register. Case 4 computes the coefficients with no normalize correction. Result exponent is zero for this case. This last case aids mutiple precision calculations.

---- Reciprocal Approximation

The floating point mutiply unit forms a 25 bit approximation to the reciprocal of a floating point operand value. This approximation is computed in two steps. The first step is a table lookup using a read only memory. The second step is a linear interpolation using the multiply mechanism. Instructions which use this sequence are the following

126i-k Enter Si with reciprocal approximation of Sk
157i-k Enter Vi with reciprocal approximation of Vk

The table used in the first step of the reciprocal approximation contains 4096 words of data each 26 bits long. The address for the table lookup is taken from the highest order 13 bits of the operand coefficient. The operand is assumed normalized so that only 12 bits are required in the table address.

The table readout is separated into two sections each 13 bits in length. These values are then used in a linear interpolation computation. The form of this computation is as follows.

Let A be a reciprocal approximation for the upper 13 bits of operand coefficient
Let B be the operand coefficient truncated at 26 bits
Let R be the better reciprocal approximation
Then the iteration step for interpolation is:

$$R = 2A - A*A*B$$

The two approximations read from the table are 2A and -A*A. The multiply unit is then used to form the product in the iteration formula with the additional sum included in the hardware pyramid.

There are two special cases treated in the reciprocal approximation sequence.

1) Operand exponent has overflow value
2) Operand exponent has underflow value

Both cases cause an error signal to be sent to the background processor status register. Both cases cause the computational result to be forced to an overflow value.

---- Reciprocal Iteration

The floating point multiply unit forms a floating point number which is used in a second iteration for the reciprocal of a full precision operand. The first iteration is formed in the reciprocal approximation described above. The second iteration uses the same process to form a reciprocal approximation with 47 bits of coefficient accuracy. The instructions which use this sequence are the following.

125ijk Enter Si with the reciprocal iteration 2 - Sj * Sk
156ijk Enter Vi with the reciprocal iteration 2 - Vj * Vk

The above instructions are followed by a normal floating point multiply instruction to complete the iteration.

An error signal is sent to the background processor status register if either of the operands in the above instructions has an overflow exponent. An overflow result exponent is forced in this case.

Floating Point Add Unit

The floating point add unit performs floating point calculations for both scalar and vector instructions. The unit is reserved during the period of the vector stream for the case of a vector instruction sequence. Scalar use is locked out during this period. No reservation is made at the time of issue for a scalar floating add operand. Successive scalar instructions may issue at the maximum rate.

The floating point add unit accepts vector element pairs at the rate of one pair per clock period. The source operands move through the unit, stopping at an internal register briefly for each clock period boundary. The transit time through the unit is a constant for all modes.

---- Floating Addition

The floating point add unit forms the sum of two operands in floating point format and delivers a result in floating point format. The result will always be normalized regardless of source operand status. Instructions which use this sequence are the following.

120ijk Enter Si with the floating sum Sj + Sk
170ijk Enter Vi with the floating sum Sj + Vk
171ijk Enter Vi with the floating sum Vj + Vk

The sum is formed by first subtracting operand exponents. The sign of the exponent difference determines the larger operand. The larger operand coefficent is entered in a 49 bit adder with a one round bit in the 49th bit position. The smaller operand coefficient is entered in the adder with a zero round bit in the 49th bit position. A binary illustration of the two quantities is shown below.

```
1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1
1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx0
```

The smaller operand coefficient field is then complemented if the algebraic signs of the operands disagree. The smaller operand coefficient field is shifted down by the amount of the exponent difference. The two quantities now appear as in the binary illustrations below.

```
1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1              signs
000001xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx0            agree
-------------------------------------------------

1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1              signs
111110xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1          disagree
-------------------------------------------------
```

The two quantities are now added in the 49 bit window illustrated above to form the result coefficient. A carry from the highest order bit is retained as part of the result coefficient.

The result coefficient may be negative if the operand exponents were equal or if the operands were not normalized. The floating add unit hardware anticipates this possibility and forms a second result coefficient which includes a complement of the data and the addition of a one bit in the 48th bit position.

The next step in the floating add process involves a test of the leading zeros in the result coefficient to determine the amount of shift necessary to normalize it. An overflow of the adder means a negative correction of one bit position. Leading zeros in the result coefficient can mean as much as a 47 bit positive correction.

A special test is made for all zero bits in the result coefficient. In this case the exponent field in the result is also cleared. A word of all zeros is delivered to the destination register.

The result coefficient is shifted by the appropriate amount and a correction is made in the larger operand exponent. The algebraic sign of the resulting floating point form is computed from the types of steps in the process.

A special case test is made for one or both operands with an overflow exponent. In this case an error signal is sent to the background processor status register and an overflow exponent is forced in the result delivered to the destination register.

---- Floating Subtraction

The floating point add unit forms the difference of two operands in floating point format and delivers a result in floating point format. This mode of operation is essentially identical to the addition mode. The only difference is the toggle of the sign bit for the subtrahend operand. The instructions which use this form of the sequence are the following.

121ijk Enter Si with floating difference Sj - Sk
172ijk Enter Vi with floating difference Sj - Vk
173ijk Enter Vi with floating difference Vj - Vk

---- Floating to Fixed Conversion

The floating point add unit forms an integer representation of a floating point operand. This process is accomplished by adding the operand to a constant in floating point format and then masking the coefficient field into the result integer. The result must be complemented and a plus one added if the resulting integer is negative. This mechanism already exists in the mid sequence option of the floating point addition sequence. The instructions which use this form of the floating add sequence are the following.

132i-k Enter Si with integer form of floating Sk
174i-k Enter Vi with integer form of floating Vk

The constant which is used as the second operand for this sequence is the quantity illustrated below in octal notation.

0 40060 0000000000000000

The addition of the specified operand to this constant will shift the operand coefficient down by the desired amount to separate the integer portion. The round bit is omitted in this process. The resulting integer is at most 48 bits in length. A negative result is sign extended by special hardware to form a 64 bit integer result.

An operand with a floating point value greater than a 48 bit integer is sensed as an error condition. A signal is sent to the background processor status register and a zero result is delivered to the destination register.

Vector Mask Register (M)

The vector mask register is a 64 bit special purpose register which is explicitly referenced in the background processor instructions. The vector mask register is used to merge vector data according to a set of precomputed element flags. In effect it provides a vehicle for implementing vector branch operations.

One bit of the vector mask register is associated with each element in the 64 element vector registers. The highest order bit of the vector mask corresponds to the first element of the vector data. The bits of the mask then proceed in order to represent the following vector elements.

The vector mask data can be formed by a vector streaming operation in which each element of the stream is evaluated for a specific criterion. The instructions for this purpose are the following.

030--k Enter mask bits where Vk has zero elements
031--k Enter mask bits where Vk has nonzero elements
032--k Enter mask bits where Vk has positive elements
033--k Enter mask bits where Vk has negative elements

The vector mask register is cleared at the beginning of these instruction sequences and then bits are entered one at a time as the vector stream passes the test station.

The vector mask data can be used to merge two vector streams into a single result stream. The instructions for this purpose are the following.

146ijk Enter Vi with Sj masked into Vk
147ijk Enter Vi with Vj masked into Vk

Elements of the j operand are selected where there are one bits in the mask. Elements of the k operand are selected where there are zero bits in the mask.

Data may be moved to, or from, the vector mask register using an S register as a way station. Instructions for this purpose are the following.

034--k Enter M from Sk
114i-- Enter Si from M

Vector Length Register (L)

The vector length register is a six bit special purpose register which is explicitly referenced in the background processor instructions. The vector length register is used to hold the vector segment length during a portion of the background computation. All vector streaming operations capture the segment length at the time of instruction issue from the L register.

The L register content is interpreted modulo 64. The allowed values are one through 64. A zero value is therefore interpreted as a 64. The L register may be set either from a constant in the instruction stream or from a computed parameter. The instructions which communicate explicitly with the L register are the following.

025i-- Enter Ai from L
036-jk Enter L with jk
037--k Enter L from Ak

Real Time Clock

Each background processor has a 64 bit register which counts continuously at the clock period rate. This count value may then be used to determine the passage of real time to an accuracy of one clock period. The instruction which reads the real time clock is the following.

115i-- Enter Si with real time count

Semaphore Flags

The four background processors share access to 16 semaphore flags. These flags are single bit registers which provide time interlocks for common memory access to critical parameter fields.

The semaphore flags are continuously moving between the background processors. The flags are organized in four groups of four flags each. There are four flags in each processor in each clock period. In the following clock period the flags will have moved in a circular fashion to new positions. The reason for this rotation of flag positions is to provide each processor an exclusive access to any flag in a four clock period unit of time. The processor can then read or alter the flag without interference from another processor.

It is essential for the read and alter function to occur in the same clock period. The background processor instructions which perform these functions are the following.

004--k Jump to constant parcel if semaphore Ak clear
005--k Jump to constant parcel if semaphore Ak set
006--k Jump to constant parcel if semaphore Ak clear. Set semaphore
007--k Jump to constant parcel if semaphore Ak set. Clear semaphore

The lowest order four bits of data in A register k select the flag. The branch test and optional flag alteration occur in the same clock period.

Status Register

The background processor requires a number of mode flags and error condition flags to indicate to the foreground processor the status of the computation. These flags are collected into a background processor status register which resides in the background processor channel interface. There are 16 bits in this status register with bit significance as follows.

Enable Flags

bit 15) Unassigned
bit 14) Enable semaphore access
bit 13) Enable memory field error
bit 12) Enable floating point error

Mode Flags

bit 11) Interrupt on semaphore error
bit 10) Interrupt on memory field error
bit 09) Interrupt on floating point error
bit 08) Interrupt on error exit
bit 07) Interrupt on normal exit
bit 06) Interrupt on idle processor

Status Flags

bit 05) Semaphore error
bit 04) Memory field error
bit 03) Floating point error
bit 02) Error exit
bit 01) Normal exit
bit 00) Idle processor

A status flag is set by a transient condition in the background processor and remains set until action is taken by the foreground processor. The enable flags and mode flags are generally set and cleared by the foreground processor. An exception is the enable for floating point error and memory field error for which there is a special background processor instruction as well.

A call to the foreground processor occurs when a mode flag and the corresponding status flag are both set. The foreground processor then reads the status register to determine the reason for the call. See 'Background Processor Channel Interface' for details.

Individual bits in the status register have the following functions.

---- Bit 14 - Enable semaphore access

This bit permits the background processor program to alter the state of the interprocessor semaphore flags. If this bit is cleared an instruction execution which attempts to alter a semaphore flag is aborted and bit five in the status register is set.

---- Bit 13 - Enable memory field error

This bit enables the setting of an error status flag if the program makes a reference to a common memory address outside the prescribed field. Such a reference may be for either an operand or an instruction fetch.

---- Bit 12 - Enable floating point error

This bit enables the setting of an error status flag if a floating point calculation is made with an operand which is larger than the floating point range.

---- Bit 11 - Interrupt on semaphore error

This bit enables a foreground processor call if the semaphore error flag is also set.

---- Bit 10 - Interrupt on memory field error

This bit enables a foreground processor call if the memory field error flag is also set.

---- Bit 09 - Interrupt on floating point error

This bit enables a foreground processor call if the floating point error flag is also set.

---- Bit 08 - Interrupt on error exit

This bit enables a foreground processor call when the background processor executes an error exit instruction.

---- Bit 07 - Interrupt on normal exit

This bit enables a foreground processor call when the background processor executes a normal exit instruction.

---- Bit 06 - Interrupt on idle processor

This bit enablgs a foreground processor call when the background processor is in an idle mode.

---- Bit 05 - Semaphore error

This bit is set when enabled by a zero bit 14 in the status register and a background processor instruction attempt to alter a semaphore flag.

---- Bit 04 - Memory field error

This bit is set when enabled by bit 13 in the status register and a background processor attempt to reference common memory beyond the prescribed range.

---- Bit 03 - Floating point error

This bit is set when enabled by bit 12 in the status register and a background computation with an operand larger than the foating point range.

---- Bit 02 - Error exit

This bit is set when the background processor executes an error exit instruction.

---- Bit 01 - Normal exit

This bit is set when the background processor executes a normal exit instruction.

---- Bit 00 - Idle processor

This bit is set when the background processor is in an idle mode.

Channel Interface

The background processors have an interface to the foreground processor for the control information necessary to sequence the background jobs. This interface is a node in one of the four foreground communication channels.

The foreground processor directs background computation through a set of background processor directives. These directives are transmitted over the communication channel as a function call with a background processor node address. The function call consists of a 32 bit primary code and an optional 32 bit parameter. The call addressee may respond with a 32 bit parameter. Organization of the primary code is as follows.

4 bits) Node address
4 bits) Unassigned
8 bits) Function code
16 bits) Interrupt address

Function codes are translated as follows

00) Null function
01) Read status register (response)
02) Read program address (response)
03) Interrupt and idle
04) Enter base address (parameter)
05) Enter limit address (parameter)
06) Enter status data (parameter)
07) Enter program address and start (parameter)

All function calls to the background processor node store the interrupt address in a holding register. This includes the null function call. The address in the holding register is used to interrupt the foreground processor when required by conditions in the background processor status register.

The interrupt process is accomplished by responding to a general channel call with the address in the holding register. This address begins a foreground processor sequence to process the interrupt request.

Individual function calls are described as follows.

---- Function one - Read status register

This function call directs the background processor channel node to respond with a readout of the processor status register. The status register data consists of 16 bits which are transmitted in the low order portion of the 32 bit response field. See 'Background Processor Status Register' for bit significance in this data.

---- Function two - Read program address

This function call directs the background processor channel node to respond with a readout of the current value in the program address register. The background processor is not stopped in this process.

---- Function three - Interrupt and idle

This function call directs the background processor channel node to interrupt the background processor computation and change to an idle mode. No further activity will occur in this processor until directed by another foreground call.

---- Function four - Enter base address

This function call directs the background processor channel node to replace the current value of the base address with the value in the function parameter. The background processor should be in an idle mode for this call.

---- Function five - Enter limit address

This function call directs the background processor channel node to replace the current value of the limit address with the value in the function parameter. The background processor should be in an idle mode for this call.

---- Function six - Enter status data

This function call directs the background processor channel node to enter the status register with the lowest order 16 bits of the parameter data. See 'Background Processor Status Register' for details of this register.

---- Function seven - Enter program address and start

This function call directs the background processor channel node to enter the program address register with the function parameter data and then begin background processor execution of the instruction code at that common memory address.

Instruction Summary

```
000---  Error exit
001---  Normal exit

002i-k  Jump to parcel Ak and hold return parcel Ai
003---  Jump to constant parcel

004--k  Jump to constant parcel if semaphore Ak clear
005--k  Jump to constant parcel if semaphore Ak set

006--k  Jump to constant parcel if semaphore Ak clear.  Set semaphore
007--k  Jump to constant parcel if semaphore Ak set.  Clear semaphore

010--k  Jump to constant parcel if Ak is zero
011--k  Jump to constant parcel if Ak is nonzero

012--k  Jump to constant parcel if Ak is positive
013--k  Jump to constant parcel if Ak is negative

014--k  Jump to constant parcel if Sk is zero
015--k  Jump to constant parcel if Sk is nonzero

016--k  Jump to constant parcel if Sk is positive
017--k  Jump to constant parcel if Sk is negative

020ijk  Enter Ai with integer sum Aj + Ak
021ijk  Enter Ai with integer difference Aj - Ak

022ijk  Enter Ai with integer product Aj * Ak
023---  Unassigned

024i-k  Enter Ai from Sk
025i--  Enter Ai from L

026ijk  Enter Ai with positive jk
027ijk  Enter Ai with negative jk

030--k  Enter M bits where Vk has zero elements
031--k  Enter M bits where Vk has nonzero elements

032--k  Enter M bits where Vk has positive elements
033--k  Enter M bits where Vk has negative elements

034--k  Enter M from Sk
035--k  Alter status register enable flags

036--k  Enter L from Ak
037---  Unassigned
```

```
040i--  Enter Ai with positive 16 bit constant
041i--  Enter Ai with negative 16 bit constant

042i--  Enter Ai with 32 bit constant
043---  Unassigned

044i--  Enter Ai from constant local memory address
045i--  Store Ai into constant local memory address

046i-k  Enter Ai from local memory address Ak
047i-k  Store Ai into local memory address Ak

050i--  Enter Si with positive 16 bit constant
051i--  Enter Si with negative 16 bit constant

052i--  Enter Si with positive 32 bit constant
053i--  Enter Si with 64 bit constant

054i--  Enter Si from constant local memory address
055i--  Store Si into constant local memory address

056i-k  Enter Si from local memory address Ak
057i-k  Store Si into local memory address Ak

060ijk  Enter Si from common memory address Aj + Ak
061ijk  Store Si into common memory address Aj + Ak

062---  Unassigned
063---  Unassigned

064---  Unassigned
065---  Unassigned

066---  Unassigned
067---  Unassigned

070ijk  Enter Vi from common memory address Aj with increment Ak
071ijk  Store Vi into common memory address Aj with increment Ak

072ijk  Enter Vi from common memory address Aj + Vk
073ijk  Store Vi into common memory address Aj + Vk

074i-k  Enter Vi from local memory address Ak
075i-k  Store Vi into local memory address Ak

076---  Unassigned
077---  Unassigned
```

100ijk Enter Si with logical product Sj and Sk
101ijk Enter Si with logical product Sj and complement Sk

102ijk Enter Si with logical difference Sj and Sk
103ijk Enter Si with logical sum Sj and Sk

104ijk Enter Si with integer sum Sj + Sk
105ijk Enter Si with integer difference Sj - Sk

106i-k Enter Si with population count of Sk
107i-k Enter Si with leading zero count of Sk

110ijk Enter Si with Si shifted left 64 - jk
111ijk Enter Si with Si shifted right jk

112ijk Enter Si with Si,Sj shifted left Ak
113ijk Enter Si with Sj,Si shifted right Ak

114i-- Enter Si from M
115i-- Enter Si with real time count

116ijk Enter Si with positive jk
117ijk Enter Si with negative jk

120ijk Enter Si with floating sum Sj + Sk
121ijk Enter Si with floating difference Sj - Sk

122i-k Enter Si with integer form of floating Sk
123--- Unassigned

124ijk Enter Si with floating product Sj * Sk
125ijk Enter Si with reciprocal iteration 2 - Sj * Sk

126i-k Enter Si with reciprocal approximation of Sk
127--- Unassigned

130i-k Enter Si with zero extended Ak
131i-k Enter Si with sign extended Ak

132i-k Enter Si with floating form of Ak
133--- Unassigned

134--- Unassigned
135--- Unassigned

136--- Unassigned
137--- Unassigned

140ijk Enter Vi with logical product Sj and Vk
141ijk Enter Vi with logical product Vj and Vk

142ijk Enter Vi with logical difference Sj and Vk
143ijk Enter Vi with logical difference Vj and Vk

144ijk Enter Vi with logical sum Sj and Vk
145ijk Enter Vi with logical sum Vj and Vk

146ijk Enter Vi with Sj masked into Vk
147ijk Enter Vi with Vj masked into Vk

150ijk Enter Vi with Vj elements shifted left Ak
151ijk Enter Vi with Vj elements shifted right Ak

152ijk Enter Vi with Vj long shifted left Ak
153ijk Enter Vi with Vj long shifted right Ak

154ijk Enter Vi with floating product Sj * Vk
155ijk Enter Vi with floating product Vj * Vk

156ijk Enter Vi with reciprocal iteration 2 - Vj * Vk
157i-k Enter Vi with reciprocal approximation Vk

160ijk Enter Vi with integer sum Sj + Vk
161ijk Enter Vi with integer sum Vj + Vk

162ijk Enter Vi with integer difference Sj - Vk
163ijk Enter Vi with integer difference Vj - Vk

164i-k Enter Vi with population count of Vk
165i-k Enter Vi with leading zero count of Vk

166--- Unassigned
167--- Unassigned

170ijk Enter Vi with floating sum Sj + Vk
171ijk Enter Vi with floating sum Vj + Vk

172ijk Enter Vi with floating difference Sj - Vk
173ijk Enter Vi with floating difference Vj - Vk

174i-k Enter Vi with integer form of floating Vk
175--- Unassigned

176--- Unassigned
177--- Pass