RIDGE PROCESSOR REFERENCE MANUAL

January 20, 1983

Contents   Page

INTRODUCTION

The Ridge personal graphics work station contains a proprietary high performance 32-bit processor implemented with bipolar MSI and LSI logic technology. It has a simple, general purpose microcoded architecture and provides processing power equal to medium performance mainframes and high performance minicomputer systems. This document describes the instruction set, exception handling, and virtual memory system architecture of the Ridge processor. In addition, approximate instruction timings are given. All specifications in this manual are preliminary and subject to change without notice.

The processor is a 32-bit, byte addressable, general register computer. The processor operates in either of two modes, user mode or kernel mode. In kernel mode, all instructions are allowed, and real memory addresses are used. Certain privileged instructions are not allowed in user mode. In user mode, a program's address space is divided into two parts, called its code segment and its data segment. Storing is not permitted into the code segment. Each of these segments contains up to 4 gigabytes of linearly addressed virtual memory. These segments are divided into 4096 byte pages for efficient memory management. The processor supports virtual memory by allowing fetching of pages on demand from external storage devices.

The processor's major clock time is 125 nanoseconds. Minimum instruction time is one clock, giving a maximum instruction rate of eight million instructions per second. Memory access time is 375 nanoseconds, which is also the minimum memory reference instruction time. The processor contains a 256 byte instruction cache designed to increase the speed of loops. An instruction fetch ahead unit in the processor allows the processor to buffer two instructions in addition to the current instruction. The fetch ahead unit attempts to fetch from the instruction cache, and should there be a cache miss, then fetches from memory. The fetch ahead unit contains branch prediction logic that speeds execution of conditional branches and helps keep the instruction pipe full.

Data Types

Data is manipulated in the processor's sixteen 32-bit general registers. There are three 32-bit data types for which the processor has instructions: logical integers, signed integers, and reals. In addition, 64-bit extended precision real numbers are supported.

The processor has instructions to load and store four different sizes of operands. The basic addressable unit is the 8-bit byte. The other operand sizes are the halfword (sixteen bits), the word (32-bits) and the double word (64-bits).

Logical operations, such as AND, are performed bitwise on logical integers. Arithmetic operations are performed on signed integers using two's complement representation. Signed integer format is shown below:

```
                                                3
                      0 1                       1
                     +-+-------------------------+
             Integer |s|                         |
                     +-+-------------------------+
```

The range of integers which can be represented this way is -2,147,483,648 through 2,147,483,647.

Real numbers have the format illustrated below. The exponent is eight bits long, and is in excess 127 notation. The mantissa has an implicit leading one. For positive numbers, S=0. Zero is represented by an all zero value. Negative numbers have S=1, using sign magnitude form.

```
                                                3
                   0  1        8  9             1
                  +-+-----------+----------------+
          Real    |s|   Exp     |   Mantissa     |
                  +-+-----------+----------------+
```

Double reals are similar to reals, except that the mantissa is longer and the exponent is 11 bits long, using excess 1023 notation.

```
                                1  1                          6
                   0 1          1  2                          3
                  +-+------------+---------------~-------------+
   Double Real    |s|     Exp    |         Mantissa            |
                  +-+------------+---------------~-------------+
```

Double words occupy register pairs. A register pair, RP(R), consists of register (R) and register (R + 1) mod 16, with register (R) holding the most significant bits and register (R + 1) mod 16 holding the least significant bits. The notation used in this manual is RP(R) for the register pair, (R) and (R)' for the individual registers in the pair.

Exceptions

Abnormal execution of an instruction is termed an exception. This may be caused by an error in the instruction, an interrupt, or some other unusual condition such as a page fault. Exception handling is discussed in detail in a later section of this document. Errors that occur in the execution of the instruction are presented following the text describing each instruction or group of instructions in the instruction set. Some exceptions may be enabled or disabled under control of the traps word -- the traps word determines which instruction errors result in suspension of the user program. The exception description for each instruction specifies how the registers are affected and what action is taken when it occurs.

Instruction Timings

The instruction timings given were measured by placing the instruction in a loop, then subtracting the loop overhead. Instruction times may increase by one clock if the instruction is not word-aligned. Some instructions have different possible paths through the microcode that implements them, and in these cases typical instruction time is given. Computing the actual time of an arbitrary instruction sequence is difficult due to the overlap of instruction fetching and execution. All timings are in microseconds with integer overflow traps disabled.

MEMORY REFERENCE INSTRUCTIONS

Memory reference instructions have either of the two formats shown below. They are distinguished by the lengths of the displacements.

```
                          1 1   1 1             3
         0        6 7 8   1 2   5 6             1
        +----------+-+-----+-----+---------------+
Short   | opcode   |x|  R  | RX  | Displacement  |
        +----------+-+-----+-----+---------------+
```

```
                          1 1   1 1                          4
         0        6 7 8   1 2   5 6                          7
        +----------+-+-----+-----+-------------~---------------+
Long    | opcode   |x|  R  | RX  |        Displacement         |
        +----------+-+-----+-----+-------------~---------------+
```

32-bit format instructions have a sixteen bit displacement field, which is sign extended to a full 32-bits. The 48-bit format instructions have 32-bits for the displacement.

The effective address for a memory reference instruction is calculated as follows:

Data segment memory reference instructions

X	Effective Address
0	Displacement
1	(RX) + Displacement

Code segment memory reference instructions

X	Effective Address
0	(PC) + Displacement
1	(PC) + (RX) + Displacement

Indexing for both the code and data segments takes place with full 32-bit signed integers in two's complement notation. Note that all code segment address modes are program counter (PC) relative thus allowing relocatable code. No store instructions can store into the code segment. Loads from the data segment use one set of opcodes while loads from the code segment are a separate set of opcodes.

Load Instructions

LOADB	-	Load Byte
LOADH	-	Load Halfword Unsigned
LOAD	-	Load Word
LOADD	-	Load Double Word

Operation:

The register specified in the R field is loaded with the data stored in memory at the effective address. In the case of LOADD, the two words are loaded into RP(R). All the addressing modes described above are supported. The data element must be aligned on a boundary which is a multiple of the length of the data element. The LOADB instruction loads the byte into bits 24..31 of the register and sets bits 0..23 to zero. The LOADH instruction loads the halfword into bits 16..31 of the specified register and clears bits 0..15.

Exceptions:

A data alignment trap may result from any of the following:

o Attempting to LOADH or LOADHS with the effective address not on a half-word boundary.

o Attempting to LOAD with the effective address not on a word boundary.

o Attempting to LOADD with the effective address not on a double word boundary.

Instruction Timing:

Instruction	Time in Microseconds
LOADB	.625
LOADH	.625 typical
LOAD	.500
LOADD	.875

Indexing does not affect execution time. The times for the PC relative code segment loads are the same as the times given above.

Store Instructions

STOREB	-	Store Byte
STOREH	-	Store Halfword
STORE	-	Store Word
STORED	-	Store Double Word

Operation:

The store instructions move data from the registers into memory. The STOREB instruction places bits 24..31 of the specified register into memory at the effective address. Other bits are ignored. The store half word instruction stores bits 16..31 and ignores bits 0..15. The effective address must be a multiple of the length of the data element.

Exceptions:

A data alignment trap may result from any of the following:

- o Attempting to STOREH with the effective address not on a half-word boundary.
- o Attempting to STORE with the effective address not on a word boundary.
- o Attempting to STORED with the effective address not on a double word boundary.

Instruction Timing:

Instruction	Time in Microseconds
STOREB	1.250
STOREH	1.000
STORE	.375
STORED	.625

Indexing does not affect execution time.

Load Address Instruction

LADDR - Load Address

Operation:

The load address instruction allows all of the code and data segment memory reference modes described above. It works just like the LOAD word command except that no memory reference actually takes place. Instead the effective address is placed in the specified register. The LADDR command can be used to load two or four byte immediate values and in indexed mode it can be used to add a constant to a register. The indexing operation occurs with full 32-bit signed integer arithmetic.

Instruction Timing:

Instruction	Time in Microseconds
LADDR	.125

Time is the same for indexed and non-indexed forms.

REGISTER FORMAT INSTRUCTIONS

The arithmetic and logical instructions use the register instruction format. Two registers are specified and the result generally replaces (R1).

```
                  1 1  1
 0           7 8  1 2  5
+-------------+----+----+
|   Opcode    | R1 | R2 |
+-------------+----+----+
```

Many of the register instructions also have an immediate mode in which the R2 specification is considered to be an integer in the range from 0 to 15.

Integer Arithmetic Instructions

NEG	-	Integer Negate	(R1) <- Two's complement of (R2)
ADD	-	Integer Add	(R1) <- (R1) + (R2)
SUB	-	Integer Subtract	(R1) <- (R1) - (R2)
MPY	-	Integer Multiply	(R1) <- (R1) * (R2)
DIV	-	Integer Divide	(R1) <- (R1) / (R2)
REM	-	Integer Remainder	(R1) <- (R1) - ((R1) / (R2))*(R2)

Operation:

The integer arithmetic instructions operate on 32-bit two's complement integers. The multiply instruction multiplies (R1) and (R2) and replaces the contents of (R1) with the low order 32 bits of the product. The divide instruction divides R1 by R2 and puts the quotient in R1. The remainder instruction divides R1 by R2 and puts the signed remainder in R1. The sign of the remainder is the sign of R1.

Exceptions:

Integer overflow can occur for the NEG, ADD, SUB, MPY, DIV, and REM instructions. An integer overflow trap is taken if this trap is enabled. NEG of 2**31 produces an integer overflow. (R1) remains unchanged. On integer overflow for ADD, SUB, and MPY, the 32 least significant bits of the result are placed in (R1) and the high order bits are discarded. Integer overflow can occur for the DIV and REM instructions when the largest negative integer is divided by -1. When this occurs, (R1) is unmodified. A divide by zero trap can occur for the DIV and REM instructions. An attempt to divide by zero leaves (R1) unmodified. A divide by zero trap is taken if this trap is enabled.

Instruction Timing:

Instruction	Time in Microseconds
NEG	.125
ADD	.125
SUB	.125
MPY	2.625 typical
DIV	4.750 typical
REM	5.000 typical

Logical Register Format Instructions

MOVE	-	Move Register	(R1) <- (R2)
NOT	-	Logical Not	(R1) <- One's complement of (R2)
AND	-	Logical And	(R1) <- (R1) and (R2)
OR	-	Logical Or	(R1) <- (R1) or (R2)
XOR	-	Logical Xor	(R1) <- (R1) exclusive or (R2)

Operation:

The logical instructions operate on 32 bit registers. The result replaces (R1).

Instruction Timing:

Instruction	Time in Microseconds
MOVE	.125
NOT	.125
AND	.125
OR	.125
XOR	.250

Extended Precision Integer Instructions

EADD	-	Extended Add	(R1) <- (R1) + (R2), 0 <- C
ESUB	-	Extended Subtract	(R1) <- (R1) - (R2), 0 <- C
EMPY	-	Extended Multiply	RP(R1) <- (R1) * (R2)
EDIV	-	Extended Divide	(R1) <- RP(R1) / (R2) (R1)' <- RP(R1) REM (R2)

Operation:

The extended integer arithmetic instructions allow multiple precision integers to be implemented. Carries on add and subtract operations are saved in register 0 and no overflow trap occurs. The multiply instruction takes two unsigned 32-bit numbers and produces an unsigned 64-bit product which it places in RP(R1). The divide instruction takes a 32-bit unsigned divisor and a 64-bit unsigned dividend and produces a 32-bit unsigned quotient and a remainder.

Exceptions:

The EDIV instruction can cause an integer overflow if the result is larger than 32 bits. An integer overflow trap occurs if this trap is enabled. If an overflow occurs, RP(R1) is unchanged. A divide by zero trap can occur on the EDIV instruction, and is taken if this trap is enabled. (R1) is unmodified if divide by zero is attempted.

Instruction Timing:

Instruction	Time in Microseconds
EADD	.375 typical
ESUB	.375 typical
EMPY	2.875 typical
EDIV	5.125 typical

Real Instructions

RNEG	-	Real Negate	(R1) <- - (R2)
RADD	-	Real Add	(R1) <- (R1) + (R2)
RSUB	-	Real Subtract	(R1) <- (R1) - (R2)
RMPY	-	Real Multiply	(R1) <- (R1) * (R2)
RDIV	-	Real Divide	(R1) <- (R1) / (R2)
FLOAT	-	Make Integer Real	(R1) <- FLOAT (R2)
FIXT	-	Truncate to Integer	(R1) <- TRUNC (R2)
FIXR	-	Round to Integer	(R1) <- RND (R2)
MAKERD	-	Real to Long Real	RP(R1) <- LONG (R2)

Operation:

The above instructions operate on 32-bit real numbers. They are all of the register format. The FLOAT instruction rounds if necessary.

Exceptions:

The RNEG, RADD, RSUB, RMPY, and RDIV instructions can overflow or underflow. When overflow occurs, (R1) is set to the largest value real number, with the appropriate sign bit. On real underflow, (R1) is set to zero. A real overflow or underflow trap is taken if the appropriate trap is enabled. The FIXT and FIXR instructions can cause an integer overflow, and an integer overflow trap is taken if this trap is enabled. On integer overflow (R1) is unmodified.

Instruction Timing:

Instruction	Time in Microseconds
RNEG	.250
RADD	1.250 typical
RSUB	2.000 typical
RMPY	3.500 typical
RDIV	6.000 typical
FLOAT	1.250 typical
FIXT	1.000 typical
FIXR	1.000 typical
MAKERD	1.500 typical

Double Real Instructions

DRNEG	- Double Real Negate	RP(R1) <- - RP(R2)
DRADD	- Double Real Add	RP(R1) <- RP(R1) + RP(R2)
DRSUB	- Double Real Subtract	RP(R1) <- RP(R1) - RP(R2)
DRMPY	- Double Real Multiply	RP(R1) <- RP(R1) * RP(R2)
DRDIV	- Double Real Divide	RP(R1) <- RP(R1) / RP(R2)
DFLOAT	- Integer to Double Real	RP(R1) <- FLOAT (R2)
DFIXT	- Double Real Truncate to Integer	(R1) <- TRUNC RP(R2)
DFIXR	- Double Real Round to Integer	(R1) <- RND RP(R2)
MAKEDR	- Round Double Real to Real	(R1) <- REAL RP(R2)

Operation:

The above instructions operate on double real format data. They use the register instruction format and all work on register pairs. Otherwise, their operation is exactly like their short real counterparts.

Exceptions:

The DRNEG, DRADD, DRSUB, DRMPY, and DRDIV instructions can overflow or underflow. When overflow occurs, RP(R1) is set to the largest value real number, with the appropriate sign bit. On real underflow, RP(R1) is set to zero. A real overflow or underflow trap is taken if the appropriate trap is enabled. The DFIXT and DFIXR instructions can cause an integer overflow, and an integer overflow trap is taken if this trap is enabled. On integer overflow (R1) is unmodified.

Instruction Timing:

Instruction	Time in Microseconds
DRNEG	.500 typical
DRADD	5.000 typical
DRSUB	5.000 typical
DRMPY	11.000 typical
DRDIV	20.000 typical
DFLOAT	2.000 typical
DFIXT	2.000 typical
DFIXR	2.000 typical
MAKEDR	1.000 typical

Register Format Immediate Instructions

MOVEI	-	Move Immediate	(Rl) <- R2
NOTI	-	Not Immediate	(Rl) <- One's complement of R2
ADDI	-	Add Immediate	(Rl) <- (Rl) + R2
SUBI	-	Subtract Immediate	(Rl) <- (Rl) - R2
ANDI	-	And Immediate	(Rl) <- (Rl) and R2
MPYI	-	Multiply Immediate	(Rl) <- (Rl) * R2

Operation:

These instructions function the same as their register format counterparts above, except that the second operand is the value of the four bit R2 field rather than the contents of the register it specifies.

Exceptions:

Integer overflow can occur for the ADDI, SUBI, and MPYI instructions. The 32 least significant bits of the result are placed in (Rl) and the high order bits are discarded. An integer overflow trap is taken if this trap is enabled.

Instruction Timing:

Instructions	Time in Microseconds
MOVEI	.125
NOTI	.125
ADDI	.125
SUBI	.125
ANDI	.125
MPYI	.750 typical

Register Format Bit Oriented Instructions

TBIT	-	Test Bit
SBIT	-	Set Bit
CBIT	-	Clear Bit

Operation:

In the TBIT instruction R2 specifies a bit number from 0..63 which is tested in RP(Rl). The bit to be tested replaces bit 31 of Rl and bits 0..30 of Rl are set to zero. SBIT and CBIT specify a bit number from 0..63 in R2. The specified bit of RP(Rl) is set to zero or one respectively.

Instruction Timing:

Instruction	0 <= (R2) <= 31	32 <= (R2) <= 63
TBIT	.500	1.000
SBIT	.500	.875
CBIT	.500	.875

Trap Instructions

CHK	-	Check Instruction	Trap if (Rl) > (R2)
CHKI	-	Check Immediate Instruction	Trap if NOT (0 <= (Rl) <= R2)

Operation:

The CHK and CHKI instructions check whether (Rl) is in the above range. If so, the instruction performs no operation, otherwise the instruction traps to the kernel.

Instruction Timing:

Instruction	Time in Microseconds
CHK	.250
CHKI	.375

The above times are for the case when no trap occurs.

TRAP	-	Trap Instruction	Trap if trap R2 enabled.

Operation:

There are sixteen traps which can be individually enabled or disabled under control of the kernel. The TRAP instruction will trap to the kernel if the bit of the traps word specified by the four bit R2 field is on. Otherwise it does nothing. It can be used for optional breakpoints.

Instruction Timing:

Instruction	Time in Microseconds
TRAP	.500

The above time is for the case when no trap occurs.

KCALL <Kernel entry point number> - Kernel Call Instruction

Operation:

The Kernel Call instruction is used by a user mode program to enter the kernel. The Rl and R2 field of the instruction are catenated (bits 8 through 15), making a kernel entry point number, which is used to choose one of 256 entry points within the kernel. Special register 15 is set to user PC + 2 on entry to the kernel. This varies from all other traps, which set SR15 to PC.

Exceptions:

Executing a KCALL in kernel mode is also invalid and results in a kernel violation trap.

Instruction Timing:

Instruction	Time in Microseconds
KCALL	1.625

Test Instructions

TEST	(R1) rop (R2)	-	Test Instruction
TESTI	(R1) rop R2	-	Test Immediate Instruction

Operation:

The test instruction compares two values and sets (R1) to either 0 (false) or 1 (true), depending upon the result of the test. The second operand is either the contents of the register specified by the R2 field of the instruction or the four bit number R2. The comparison is done using signed two's complement arithmetic. The comparison relational operator (rop) is:

rop
=
<
>
<>
<=
>=

Instruction Timing:

Instruction	Time in Microseconds
TEST, TESTI rop <, >, <=, >=	.250 typical
TEST, TEST1 rop =, <>	.375 typical

Compare Register Format Instructions

DCOMP RP(Rl), RP(R2) - Double register compare

Operation:

Register pair RP(Rl) and RP(R2) are compared using two's complement arithmetic. (Rl) is set to -1, 0 or 1 depending upon whether RP(Rl) is less than, equal to, or greater than RP(R2), respectively.

Instruction Timing:

Instruction	Time in Microseconds
DCOMP	.625 average

LCOMP (Rl),(R2) - Logical compare

Operation:

Registers (Rl) and (R2) are compared using unsigned arithmetic. (Rl) is set to -1, 0, or +1 depending on whether (Rl) is less than, equal to, or greater than (R2) respectively.

Instruction Timing:

Instruction	Time in Microseconds
LCOMP	.500

Compare Register Format Instructions -- Continued

RCOMP (R1), (R2) - Real compare

Operation:

Registers (R1) and (R2) are compared as real numbers using sign magnitude form. (R1) is set to -1, 0, or +1 depending on whether (R1) is less than, equal to, or greater than (R2) respectively.

Instruction Timing:

Instruction	Time in Microseconds
RCOMP	.750 typical

DRCOMP RP(R1), RP(R2) - Double real compare

Operation:

The register pairs RP(R1) and RP(R2) are compared as double real numbers using sign magnitude form. (R1) is set to -1, 0 or +1 depending upon whether RP(R1) is less than, equal to, or greater than RP(R2) respectively.

Instruction Timing:

Instruction	Time in Microseconds
DRCOMP	1.000 typical

SHIFT INSTRUCTIONS

These instructions take the shift count from (R2) and can shift from 0 to 31 bits in the case of single register shifts and from 0 to 63 bits in the case of double register shifts. Only the low order 5 bits or 6 bits of (R2) are used as the shift count for single or double shifts respectively. In addition, immediate forms are available which use the four bits of the R2 field as the shift count thus allowing shifts from 0 to 15 bits. The shift times are independent of the number of bits shifted.

Circular Shift Instructions

```
CSL    (Rl), (R2)  -  Circular shift left (single register)
CSLI   (Rl), R2    -  Circular shift left immediate
```

Operation:

The CSL instruction shifts the specified register left by the shift count. Bits shifted out bit 0 are shifted into bit 31.

Instruction Timings:

Instruction	Time in Microseconds
CSL	.375
CSLI	.125

Single Register Logical Shifts

LSL	(Rl), (R2)	-	Logical shift left
LSLI	(Rl), R2	-	Logical shift left immediate
LSR	(Rl), (R2)	-	Logical shift right
LSRI	(Rl), R2	-	Logical shift right immediate

Operation:

The logical shift instructions shift in zero bits and the bits shifted out are lost.

Instructions Timings:

Instruction	Time in Microseconds
LSL	.375
LSLI	.125
LSR	.500
LSRI	.500

Single Register Arithmetic Shifts

ASL	(R1), (R2)	-	Arithmetic left shift
ASLI	(R1), R2	-	Arithmetic left shift immediate
ASR	(R1), (R2)	-	Arithmetic right shift
ASRI	(R1), R2	-	Arithmetic right shift immediate

Operation:

The arithmetic shift right fills the register shifted with the value of the sign bit prior to shifting. The ASL keeps the sign bit constant as bits are shifted out the left.

Exceptions:

In the ASL and ASLI instructions, when a bit different than the initial sign bit is shifted out of bit one into the sign bit, an integer overflow trap is taken if the integer overflow trap is enabled.

Instruction Timing:

Instruction	Time in Microseconds
ASL	.563
ASLI	.438
ASR	.563
ASRI	.563

Double Register Logical Shifts

DLSL	(R1), (R2)	-	Double logical shift left
DLSLI	(R1), R2	-	Double logical shift left immediate
DLSR	(R1), (R2)	-	Double logical shift right
DLSRI	(R1), R2	-	Double logical shift right immediate

Operation:

The double logical shift operate on pairs of adjacent registers and are otherwise identical to the single logical shifts.

Instruction Timings:

Instruction	Time in Microseconds
DLSL	1.250
DLSLI	.750
DLSR	1.250
DLSRI	.750

Sign Extend Instructions

SEB	(Rl), (R2)	- Sign extend byte.
SEH	(Rl), (R2)	- Sign extend halfword.

Operation:

The sign extend instructions change partial word integers into full word integers. The SEB instruction makes bits 0..23 in (Rl) the same as bit 24 in (R2). Bits 24..31 in (R2) are copied to (Rl). The SEH instruction makes bits 0..15 in (Rl) the same as bit 16 in (R2). Bits 16..31 in (R2) are copied to (Rl).

Instruction Timings

Instruction	Time in Microseconds
SEB	.500
SEH	.500 typical

No Operation (NOP) Instruction

NOP This instruction performs no operation.

Instruction Timings:

Instruction	Time in Microseconds
NOP	.125

BRANCH INSTRUCTIONS

Branch instructions generally use either of the two memory reference instruction formats as shown below:

```
                             1 1    1 1              3
         0             7 8   1 2    5 6              1
        +-------------+-----+-----+--------------+
Short   |   Opcode    | R1  | R2  | Displacement |
        +-------------+-----+-----+--------------+
```

```
                             1 1    1 1                                  4
         0             7 8   1 2    5 6                                  7
        +-------------+-----+-----+----------------~-------------+
Long    |   Opcode    | R1  | R2  |            Displacement       |
        +-------------+-----+-----+----------------~-------------+
```

The CALL register form instruction and the RET instruction both use the register format:

```
                             1 1    1
         0             7 8   1 2    5
        +-------------+-----+-----+
        |   Opcode    | R1  | R2  |
        +-------------+-----+-----+
```

The branch instructions use program counter (PC) relative addressing which allows self relocating code. The target address of the branch instruction is computed by adding the displacement (sign extended in the short form case) to the PC of the beginning of the branch instruction.

The low order bit of the displacement field is used by the processor to predict whether or not the branch will be taken. If it is one, the processor will prefetch the instruction at the target address, and if it is zero, the processor will prefetch the next sequential instruction. If the prediction bit turns out to be incorrect the program will execute correctly, but the next instruction after the branch will be delayed by one memory cycle.

Unconditional Branch Instruction

BR - Displacement

Operation:

An unconditional branch is taken to the target address. The branch prediction bit is ignored and the target instruction is prefetched.

Instruction Timing:

Instruction	Short Displacement	Long Displacement
BR	.250	.375

If the processor fetch ahead unit has fetched the unconditional branch instruction, the processor advances the program counter to the branch target, and the unconditional branch instruction takes zero time.

Conditional Branch Instructions

BR	(Rl) rop (R2), Displacement	-	Compare and branch
BR	(Rl) rop R2, Displacement	-	Compare immediate and branch

Operation:

The compare and branch instructions compare (Rl) to the contents of R2 or the four bit immediate value R2. Comparisons are done using two's complement arithmetic. The relational operator (rop) for the compare and branch immediate instruction may be: =, <>, >, <, >=, or <=. The compare and branch rop may be: =, <>, >, or <=. The < and >= relational operators are accomplished by reversing (Rl) and (R2).

Instruction Timing:

Instruction	Predicted	Not Predicted
BR	.250	.750

The long displacement form of the branch instruction takes an additional .125 microseconds.

Subroutine Call and Return Instructions

CALL	Rl, Displacement	-	Call instruction
CALLR	Rl, R2	-	Register Format Call
RET	Rl, R2	-	Return or Call absolute

Operation:

The CALL instruction puts the address of the next instruction into Rl and then branches to the effective address. In the case of the CALLR the relative displacement is taken from (R2) and is added to the PC at the beginning of the CALLR instruction. The RET instruction places the address of the next sequential instruction into (Rl) and branches to the absolute address in R2. The main use for RET is in returning from subroutines but it can also be used as a call to a subroutine when the absolute address is known rather than the relative address. Care must be taken in using the RET instruction so that the code remains self relocating.

Instruction Timings:

Instruction	Time in Microseconds
CALL	.250
CALLR	.625
RET	.500

The long displacement form of the CALL instruction takes an additional .125 microseconds.

Loop Control Instructions

LOOP (Rl), R2, Displacement - Increment and branch if < 0

Operation:

The LOOP instruction adds the four bit value R2 to the contents of (Rl) and branches if the result is less than zero.

Instruction Timing:

Instruction	Predicted	Not Predicted
LOOP	.250	.750

The long displacement form of the LOOP instruction takes an additional .125 microseconds.

KERNEL MODE INSTRUCTIONS

The normal execution of a user mode program may be suspended by either an interrupt from an external device or a trap resulting from an error in the program. The term exception is used to include both interrupts and traps. The occurrence of an exception will cause the processor operating in user mode to switch to kernel mode and begin execution at a specific (dependent on the exception) memory location.

Kernel mode is distinct from user mode in three important ways. First, the processor operates with real memory addresses instead of virtual addresses. Second, interrupts are disabled. And lastly, certain instructions become available which cause an illegal instruction trap in user mode.

In addition to the 16 general registers the system also contains 16 32-bit special registers which may be used as high speed scratch pad areas by the kernel. These special registers are also used to pass the kernel information about exceptions.

The addresses of the exception handling routines are in the CPU Control Block (CCB), which is in an area of memory pointed to by special register 11 (SR11). When an exception occurs the following sequence of events occurs. The current instruction is completed if the exception is an interrupt, or terminated if the exception is a trap. The contents of the program counter, which points to the next user mode instruction in the case of an interrupt or to the aborted instruction in case of a trap, are placed in SR15. If the trap occurred in kernel mode, SR15 is left unchanged and the program counter is placed in SR0. This allows traps such as double bit parity error to occur in both user and kernel mode. SR0 is set to 1 if the exception occurs in user mode.

Special registers SR1, SR2 and SR3 are set by the processor to describe the type of exception. The processor then switches to kernel mode, and begins execution at the location pointed to by the exception table.

The processor may be returned to user mode by executing the Resume User Mode instruction. This instruction causes the processor to load the program counter with the value found in SR15 and switch to user mode.

Note that neither exceptions nor the Resume User Mode instruction have any effect on the contents of the general registers, the code and data segment numbers, or the traps word. These, together with the user program's program counter value form the Process Control Block (PCB), the format of which is given below. Two instructions, Load User State, and Save User State, are used to prepare the processor for the execution of another user mode process, or to save the state information of the one that was just halted by the exception. These two instructions take the pointer to the process control block from SR14.

```
                                        3
       0                                1
      ----------------------------------
 0    |           Register 0           |
      +--------------------------------+
 4    |           Register 1           |
      +--------------------------------+
 8    |                                |
      ~                                ~
      ~                                ~
      |                                |
      +--------------------------------+
60    |           Register 15          |
      +--------------------------------+
64    |         Program Counter        |
      +--------------------------------+
68    | Code Segment   | Data Segment  |
      +--------------------------------+
72    |            Reserved            |
      +--------------------------------+
76    |           Traps Word           |
      +--------------------------------+
80    |          Process Clock         |
      ----------------------------------
```

Process Control Block (PCB)

All of the kernel mode instructions use the register format.

Privileged Mode

The least significant bit of the traps word (bit 31) is used as a privileged mode indicator. Certain kernel mode instructions (such as the I/O instructions), can be executed in user mode if the privileged mode bit in the traps word is set. All the kernel mode instructions require either kernel mode or privileged user process to be executed, and cause a kernel violation trap when not in the appropriate mode.

State Switching Instructions

SUS R1, R2 - Save user state

Operation:

The Save User State instruction stores the user program counter and the process clock into the PCB pointed to by special register SR14. In addition, the general registers are stored beginning with R1 and ending with R2. The instruction can store from one to 16 of the general purpose registers. If the R1 specification is greater than R2 then only R1 is stored (i.e. register numbers do not wrap around). If SR14 = 1, no registers are stored and the instruction performs no operation.

Exceptions:

An attempt to execute this instruction when not in kernel mode results in a kernel violation trap.

Instruction Timing:

Instruction	Time in microseconds
SUS	0.750 + .375n

where "n" is the number of registers saved.

LUS R1, R2 - Load User State

Operation:

The Load User State instruction is the inverse of the SUS instruction. It loads the user program counter, the code and data segment numbers, and the traps word from the PCB pointed to by SR14 into SR15, SR8, SR9, and SR10, respectively. From one to 16 general registers can also be loaded. If R1 is greater than R2, then only R1 is loaded (i.e., register numbers do not wrap-around). The instruction cache and translation mapping table (see virtual memory section below) are flushed. If SR14 = 1, no registers are loaded and the instruction performs no operation.

Exceptions:

An attempt to execute this instruction when not in kernel mode results in a kernel violation trap.

Instruction Timing:

Instruction	Time in microseconds
LUS	10.125 for 5 or fewer registers, 10.125 + .375(n-5) for greater than 5 registers, where "n" is the number of registers loaded.

LDREGS R1,R2 - Load Registers

Operation:

The LDREGS instruction loads from one to 16 registers from the PCB pointed to by SR14. The LDREGS instruction differs from LUS in that the program counter, code and data segments and traps word are not read from the PCB, and the instruction cache and translation mapping table are unchanged. This instruction provides a faster method for loading registers than the LUS instruction. This instruction is useful in restoring user state after a Kernel operation which does not cause a process context switch. If R1 is greater than R2, then only R1 is loaded (i.e., no register wrap-around). IF SR14 = 1, no registers are loaded and the instruction performs no operation.

Exceptions:

An attempt to execute this instruction when not in kernel mode results in a kernel violation trap.

Instruction Timing:

Instruction	Time in microseconds
LDREGS	.750 + .375n

where "n" is the number of registers loaded.

RUM - Resume User Mode

Operation:

The processor switches from kernel to user mode, loading the program counter from SR15. The user program begins executing at the location in SR15. If SR14 = 1, the processor pauses until an interrupt occurs, at which time the kernel is entered. SR0 is then set to the kernel's program counter. The kernel interrupt handler may then LUS and RUM to a new user program, or if SR14 remains set to one, the LUS has no effect and the RUM again causes the processor to pause.

Exceptions:

An attempt to execute this instruction when not in kernel mode results in a kernel violation trap.

Instruction Timing:

Instruction	Time in microseconds
RUM	1.125

MOVE (SR1), (R2) - Move general register to special register
MOVE (R1) , (SR1) - Move special register to general register

Operation:

The above forms of move transfer data between the special registers and the general registers.

Exceptions:

An attempt to execute these instructions when not in kernel mode results in a kernel violation trap.

Instruction Timing:

Instruction	Time in microseconds
MOVE S-R	.375
MOVE R-S	.375

Maintenance Instructions

Maintenance instructions are register format instructions that use the R2 field as part of the opcode. In these instructions, (Rl), or RP(Rl) are used for both input and output. The format of the maintenance instructions is shown below:

```
          0           7 8    11 12    15
          +-----------+------+-------+
   MAINT  |  Opcode   |  Rl  | Subop |
          +-----------+------+-------+
```

Subop is an extension of the opcode:

Subop	Maintenance Instruction
0	ELOGR
1	ELOGW
6	FLUSH
7	TRAPEXIT
8	ITEST

ELOGR - Memory Error Logging RAM Read

Operation:

The ELOGR instruction reads the memory error logging RAM, using (Rl) as an address. The logging RAM data is returned as a bit in (Rl). Output (Rl) also contains the processor status, regardless of the input (Rl) value. The pertinent status bits are described below.

Status Description	Bit No.
Memory error logging RAM data	16
External interrupt	27
Secondary/primary boot device	30
Load enable switch set	31

Exceptions:

An attempt to execute this instruction when not in kernel mode or a privileged user process (bit 31 in SR10) results in a kernel violation trap.

Instruction Timing:

Instruction	Time in microseconds
ELOGR	1.125 in kernel mode 1.500 in privileged user mode

ELOGW Memory Error Logging RAM Write.

Operation:

The ELOGW instruction writes one bit into the memory error logging RAM. The logging RAM address and data bit are both contained in (R1).

Exceptions:

An attempt to execute this instruction when not in kernel mode or a privileged user process (bit 31 in SR10) results in a kernel violation trap.

Instruction Timing:

Instruction	Time in microseconds
ELOGW	0.750 in kernel mode 1.125 in privileged user mode

FLUSH - Flush Translation Buffer

Operation:

The processor's virtual to real translation mapping table is invalidated and the instruction cache is emptied. This instruction may be executed in Privileged mode as well as kernel mode.

Exceptions:

An attempt to execute this instruction when not in kernel or privileged mode results in a kernel violation trap.

Instruction Timing:

Instruction	Time in microseconds
FLUSH	9.000

TRAPEXIT Exit From Trap Instruction in Kernel Mode.

Operation:

The TRAPEXIT instruction is used to start executing in the kernel at the location contained in SR0. The instruction cache and translation mapping table are flushed. This instruction is intended to be used with the TRAP instruction to set breakpoints in the kernel. TRAPEXIT may be used in lieu of the RUM instruction in order to resume executing in kernel mode rather than user mode.

Exceptions:

An attempt to execute this instruction when not in kernel mode results in a kernel violation trap.

Instruction Timing:

Instruction	Time in microseconds
TRAPEXIT	9.000

ITEST RP(Rl) <- interrupt, IOIR data

Operation:

The ITEST instruction tests for the presence of an interrupt, and returns the I/O Interrupt Read (IOIR) word if an interrupt is present in (Rl)'. (Rl) is set to zero to indicate the presence of an interrupt. If there is no interrupt, (Rl) is set to one and (Rl) is unchanged.

Exceptions:

An attempt to execute this instruction when not in kernel mode results in a kernel violation trap.

Instruction Timings:

Instruction	Time in Microseconds
ITEST	1.000 if no interrupt, 1.785 if interrupt.

Virtual Memory Support Instructions

TRANS	(R1) <- real RP(R2).	Translate virtual address to real.
DIRT	(R1) <- real RP(R2).	Translate virtual address and mark page dirty.

Operation:

The TRANS instruction takes the segment number in (R2) and the virtual address in (R2)' and replaces (R1) with the corresponding real address. If the address is not translatable with the current VRT, then (R1) is set to -1. If the address is translatable, then the reference bit is set in the VRT for this address. The DIRT instruction is the same as the TRANS instruction except that the modified bit in the VRT for the page containing the virtual address is also set.

Exceptions:

An attempt to execute this instruction when not in kernel mode results in a kernel violation trap.

Instruction Timing:

Instruction	Time in microseconds
TRANS,DIRT	3.0 typical

Input/Output Instructions

```
READ     Read data from device
WRITE    Write data to device
```

Operation:

The READ instruction sends (R2) as an I/O read address word to the device number specified in the most significant byte of (R2). The least significant bytes of (R2) are device dependent data. (R1) is set to the I/O status and (R1)' contains the I/O read data word from the device. The WRITE instruction sends (R2) as an I/O write address word and (R1) as an I/O write data word to the device specified in the most significant byte of (R2). (R1) is set to the I/O status. Register formats for read, write and I/O status are as follows:

READ Instruction

```
                     0         7 8                      31
(R1)     I/O Read    +----------+-----------------------+ Sent to
         Address     | device # | device dependent data | Device
         Word        +----------+-----------------------+

                     0                                  31
(R1)'    I/O Read    +----------------------------------+
         Data Word   |      data read from device       |
                     +----------------------------------+
```

WRITE Instruction

```
                     0         7 8                      31
(R2)     I/O Write   +----------+-----------------------+ Sent to
         Address     | device # | device dependent data | Device
         Word        +----------+-----------------------+

                     0                                  31
(R1)     I/O Write   +----------------------------------+
         Data Word   |     data written to device       |
                     +----------------------------------+
```

Input/Output Instructions -- Continued

I/O STATUS returned in (R1)

```
0                          30  31
+-------------------------+---+---+
|                         | T | N |
+-------------------------+---+---+
```

T = "0" is ok, "1" is device timed out and did not respond.

N = "0" is ok, "1" is I/O device not ready to accept command.

Instruction Timing:

Instruction	Time in microseconds
READ	1.250
WRITE	0.875

Exceptions:

An attempt to execute this instruction when not in kernel mode or a privileged user process (bit 31 of SR10) results in a kernel violation trap.

EXCEPTION HANDLING

This section describes the processor's functions upon exceptions to instructions. This activity includes traps, external interrupts and timer interrupts, each of which are described below.

Traps

The table below lists the CPU Control Block and the address referenced by each trap (offset from SR11). The values of SR1, SR2 and SR3 upon entry to the kernel are also given. SR0 is used as a flag to indicate kernel or user mode when the trap occurs. If the trap occurred in user mode, SR0 is set to 1, otherwise SR0 contains the value of the kernel's program counter. SR15 contains the value of the user program counter.

CPU Control Block (CCB)

Trap	Hex Addr	SR1	SR2	SR3
KCALL 0	0			
KCALL 1	4			
KCALL 2	8			
. .	.			
. .	.			
KCALL 255	3FC			
Data Alignment	400		Segment #	Virt Addr
Illegal Instr	404	Opcode	R1 Field	R2 Field or Virt Addr
Double Bit Parity Error - Code Fetch	408			Virt Addr or Real Addr
Double Bit Parity Error - Execute	40C			Virt Addr or Real Addr
Page Fault	410		Segment #	Virt Addr
Kernel Violation	414	Opcode	R1 Field	R2 Field
Check Trap	414		R1 Field	R2 Field
Traps Word Traps	41C			
Trap Instruction			R2 Field	
Integer Overflow			16	
Integer Div by zero			17	
Real Overflow			18	
Real Underflow			19	
Real Div by Zero			20	
External Interrupt	420	Dev # word		
Switch 0 Interrupt	424			
Power Fail Warning	428			
End Power Glitch	42C			
Timer 1 Interrupt	430			
Timer 2 Interrupt	434			
Reserved	438			
Data Area				
Clock Tick While Paused Count	43C			
Timer 1 Count	440			
Timer 2 Count	444			
Time of Day in Nanoseconds	448- 44C			

Notes on Traps:

1. KCALL. SR15 is set to PC + 2 on entry (rather than PC).

2. Illegal Instruction. The contents of SR2 is dependent upon the format of the illegal instruction (determined by which opcode group it is in). If it is a register format instruction then SR2 contains the value (i.e. four bits) of the R2 field. Otherwise it contains the virtual address.

3. Double Bit Parity Error - Code Fetch. A code fetch error can occur when the CPU fetch-ahead unit reads a word of code from memory. The address of the parity error is determined as follows:

   o User Mode. SR8 contains the code segment number, and SR3 contains the virtual address.

   o Kernel Mode. SR3 contains the real memory address.

4. Double Bit Parity Error - Execute. An execute error can occur as the CPU executes a memory reference instruction. The address and type of the parity error is determined as follows:

   o User Mode. Special registers 8 and 15 contain the segment number and virtual address of the memory reference instruction that failed. The opcode of the instruction determines whether it is a reference to a code or data segment. Special register 8 contains the code segment number, and SR9 contains the data segment number. The virtual address is in SR3.

   o Kernel Mode. SR0 contains the address of the memory reference instruction that failed. The real address of the parity error is in SR3.

5. Page Fault. This fault can never occur in kernel mode.

6. Trap Word Traps. These traps are under control of the traps word, which has the format given below.

```
             1   1   1   1   1   2         3
 0           5   6   7   8   9   0         1
+------~------+----------------------~--+---+
| Trap Instr  |   O| D0| RO| RU |RD0|     | PM|
+------~------+---+---+---+---+---+--~--+---+
```

Trap Instr - These sixteen bits control the trap instruction. The R2 field from the trap instruction is placed in SR3.

```
O    - Integer Overflow.
D0   - Integer Divide by zero.
RO   - Real Overflow.
RU   - Real Underflow.
RD0  - Real Divide by Zero.
PM   - Privileged Mode.  An attempt to execute an instruction
         which requires privileged mode results in a kernel
         violation.
```

7. Switch 0 Interrupt. This interrupt occurs when switch 0 on the clock board is depressed and released.

8. Power Fail Warning. This interrupt is caused when the power supply detects that AC power is being removed.

9. End Power Glitch. When AC power is removed only momentarily, and not lost, first a power fail warning interrupt occurs, which is then followed by an end power glitch interrupt.

External Interrupts

An external interrupt is an interrupt caused by a peripheral device. Special register 0 contains the I/O Interrupt Read word from the device. The data in this word is device dependent except that the device number is contained in the most significant byte. Interrupts that occur in kernel mode are held off until the process returns to user mode.

Timer Interrupts

The processor has four timekeeping facilities. One is process time, which is incremented once each millisecond. When SR14 <> 1, a user processor is running, and the process clock word in the PCB is incremented. If SR14=1, the processor is paused, and the "clock tick while paused count" in the CCB data area is incremented.

The second facility is time of day. Once each millisecond one million nanoseconds is added to the "time of day in nanoseconds" double word in the CCB data area.

Two interval timers are also provided. Once each millisecond the timer 1 count and then the timer 2 count in the CCB data area are decremented. If either timer is less than zero, the kernel is entered at the appropriate timer trap.

VIRTUAL MEMORY SUPPORT

The processor supports demand paged virtual memory using 4096 byte pages. The memory system has a Translation Mapping Table (TMT) which contains virtual-to-real address mapping entries for 32 pages. Sixteen of these entries are dedicated to code segment pages and sixteen are dedicated to data segment pages. When the processor is in user mode it communicates with the memory using virtual addresses and indicates whether each reference is for code or data. The TMT is searched to determine the real address of the data. The TMT search is overlapped with memory access so that no time penalty is incurred when the mapping information is in the TMT.

When no entry is found in the TMT, a microcode interrupt occurs in the processor which causes it to search the Virtual to Real Translation table (VRT). If the VRT table contains the entry, it is loaded into the TMT and processing continues. If not, the current macro instruction is aborted and a software page fault interrupt is generated. When the processor is executing in kernel mode, real addresses are used and the TMT search is bypassed. Direct memory access by I/O devices also uses real addresses.

The VRT performs the same function as the page table in conventional virtual memory machines. Because of the very large virtual address range supported by the Ridge processor, using this standard technique would likely result in very large page tables. For this reason the VRT is organized as a hash table with one entry for each real page. The processor has firmware support for efficiently searching the VRT and this is done automatically without the user programmer's knowledge. The VRT is kept in main memory and its size is dependent upon the amount of memory in the system.

When the processor needs to search the VRT, it proceeds as follows (Please refer to figures containing VRT table entries and VRT layout on the following pages):

1. The segment number of the code or data segment to be referenced is added to bits 0..19 of the virtual address.

2. This sum is logically ANDed with the contents of VRMASK which is kept in special register SR13.

3. The result is shifted left 3 bits and added to the VRT table base address which is stored in SR12.

4. The VRT entry is fetched and the tag and segment number parts are compared with virtual address and segment number desired.

5. If they match, the real page number, virtual address, and modify bits are loaded into the TMT and the referenced bit is set.

6. If not, the link pointer is followed (added to SR12) to the next VRT entry. If a link pointer of zero is found the end of the chain has been reached and a page fault interrupt is generated.

The average time to search the VRT is approximately 2 microseconds.

Special registers SR11, SR12, and SR13 are involved in virtual memory address translation and must be properly set before the first VRT search is performed.

The VRT table entries are defined below:

```
                1 1                  3
 0              5 6                  1
----------------+--------------------
|     Seg #     |        Tag         |
-------------------------------------


                1 1    2 2           3
 0              5 6    0 1           1
----------------+------+------------
|    Link ^     | RVVVM | Real Page |
-------------------------------------
```

Seg # - Uniquely defines a particular code or data segment.

Tag - High order bits of the virtual address (bits 0..15).

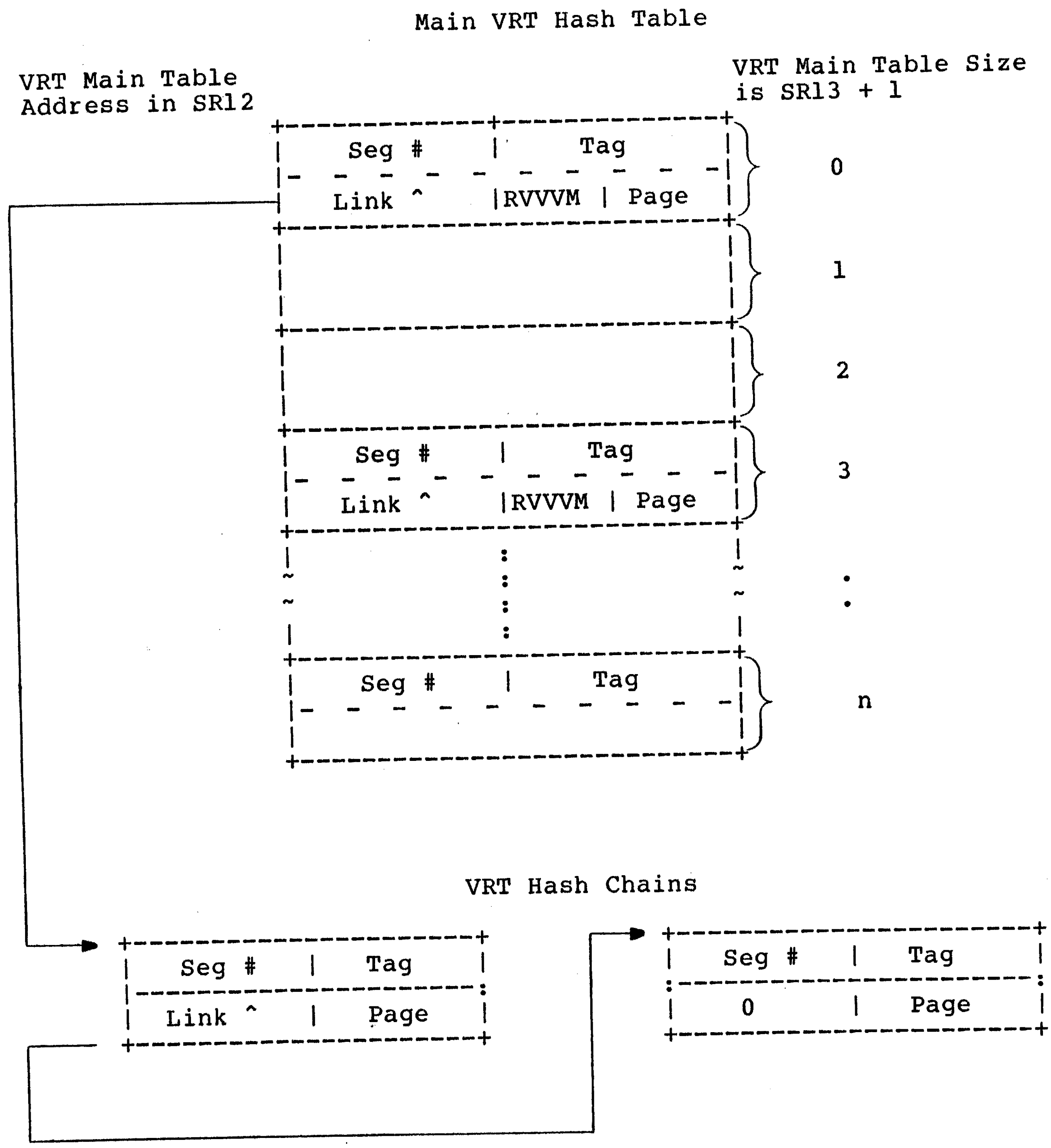
Link ^ - Pointer to next VRT entry with the same hash code.

R - Referenced bit.

VVV - 000 indicates entry is invalid
111 indicates entry is valid.

M - Modified bit.

Real Page - Real page number containing the virtual page.


Main VRT Hash Table
VRT Main Table Address in SR12
VRT Main Table Size is SR13 + 1
Seg #
Tag
Link ^
RVVVM
Page
0
1
2
Seg #
Tag
Link ^
RVVVM
Page
3
Seg #
Tag
n
VRT Hash Chains
Seg #
Tag
Link ^
Page
Seg #
Tag
0
Page

Notes on VRT

1. SR12 is full 32-bit pointer, so the Main VRT Hash Table can reside anywhere in real memory.

2. All VRT link pointers are 16-bit unsigned quantities. When a link is followed, its 16-bit value is added to the Main VRT base (SR12) to get the real address of the linked object. Thus, chained VRT entries can be no more than 65535 bytes away from the VRT base.

3. There is no method for constructing an invalid VRT entry. Kernel software must select one segment number to indicate an invalid entry, then take care to never use this number as a segment number, in SR8, SR9, or as input to the TRANS or DIRT instructions.

4. VRMASK must be a right justified mask of 1's. VRT main hash table sizes must be a power of 2, with VRMASK set to hash table size minus one.

5. The smallest value for VRMASK (SR13) is 15 (000F in Hex). Smaller values would allow collisions in the hash table which have identical tag fields.

SPECIAL REGISTER ASSIGNMENT

Register	Function
SR0	Kernel flag. Set on trap. SR0 = 1, trapped in user mode, otherwise SR0 = kernel PC.
SR1	KCALL no./opcode. Set on trap.
SR2	Segment no./R1 field. Set on trap.
SR3	Virtual address/R2 field. Set on trap.
SR4 - SR7	Unused.
SR8	Code segment number.
SR9	Data segment number.
SR10	Traps word.
SR11	Address of CPU Control Block in memory.
SR12	Address of VRT table in memory.
SR13	VRMASK used for hash generation (set to one less than hash table size)
SR14	Current process pointer. (if SR14=1, then there is no current process)
SR15	User program counter (PC)

INSTRUCTION INDEX AND OPCODE ASSIGNMENTS

RIDGE OPCODE MAP

Least Significant Nibble --- Opcode (4:7)

Most Significant Nibble --- Opcode(0:3)

	Segment Referenced	Length		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Register Format			0		MOVE	NEG	ADD	SUB	MPY	DIV	REM	NOT	OR	XOR	AND	CBIT	SBIT	TBIT	CHK
			1	NOP	MOVEI		ADDI	SUBI	MPYI			NOTI			ANDI				CHKI
			2	FIXT	FIXR	RNEG	RADD	RSUB	RMPY	RDIV	MAKERD	LCOMP	FLOAT	RCOMP		EADD	ESUB	EMPY	EDIV
			3	DFIXT	DFIXR	DRNEG	DRADD	DRSUB	DRMPY	DRDIV	MAKEDR	DCOMP	DFLOAT	DRCOMP	TRAP				
			4	SUS	LUS	RUM	LDREGS	TRANS	DIRT	MOVESR	MOVERS					MAINT		READ	WRITE
			5	TEST >	<	=	CALLR	TESTI >	<	=	RET	TEST <=	>=	<>	KCALL	TESTI <=	>=	<>	
			6	LSL	LSR	ASL	ASR	DLSL	DLSR			CSL		SEB					
			7	LSLI	LSRI	ASLI	ASRI	DLSLI	DLSRI			CSLI		SEH					
		Short	8	BR		BR	CALL	BR	IMM		LOOP	BR		BR	BR	BR	IMM		
		Long	9	>		=		>	<	=		<=		<>	UNC	<=	>=	<>	
		Short	A	STOREB	X	STOREH	X			STORE	X	STORED	X						
		Long	B		X		X				X		X						
	Data	Short	C		X		X				X		X						X
	Data	Long	D	LOADB	X	LOADH	X			LOAD	X	LOADD	X					LADDR	X
	Code	Short	E		X		X				X		X						X
	Code	Long	F		X		X				X		X						X

X = Indexed