PERQ QCode Reference Manual

Miles A. Barel
John P. Strait

Three Rivers Computer Corporation

September 3, 1980

Three Rivers Computer Corporation
160 North Craig Street
Pittsburgh, PA 15213
(412) 621-6250

The information in this document is subject to change without notice and should not be construed as a commitment by Three Rivers Computer Corporation. Three Rivers Computer Corporation assumes no responsibility for any errors that may appear in this document.

Table of Contents

4.E Systems Programs Support Procedures

1. Q-Machine Architecture

1.A Definitions

Segment - A segment is the underlying structure of PERQ's virtual memory system. It is the largest area of contiguous memory, and also the unit of swappability. Segments come in two types: code segments, which are byte-addressed, read-only, and fixed in size with a maximum size of 64K bytes (32K words); and data segments, which are word-addressed, read-write, and variable in size with a maximum size of 64K words.

MSTACK - Memory Stack. A data segment which contains the user run-time stack.

ESTACK - Expression Stack. A 16 level expression evaluation stack (internal to the PERQ processor).

MTOS - Top of MSTACK. MTOS refers to the virtual address of the top of the memory stack. (MTOS) denotes the item on the top of the MSTACK.

ETOS - Top of ESTACK. (ETOS) denotes the item on the top of the ESTACK.

Activation Record - Stack segment fragment for a single routine containing local variables, parameters, function result, temporaries (anonymous variables), other housekeeping values (Activation Control Block - defined below), and a copy of the EStack at the time the activation record is created.

CB - Code Base (register). Physical adddress of the base of the current code segment.

SB - Stack Base (register). Physical address of the base of the current stack segment.

PC - Program Counter (register). Physical address of the current instruction.

GDB - Global Data Block. A GDB contains the global variables for a particular module. GDBs are always begin on a double-word boundary.

ISN - Internal Segment Number (compiler-generated).

SSN - System Segment Number (system-generated). Note, System Segment Ø is reserved and may never be used.

LL - Lexical Level. Note: the Lexical Level of the main body of a process is always Ø.

RN - Routine Number (register). RN contains the ordinal number of the current routine. Note: RN must lie in the range Ø to 255.

CS - Code Segment (register). CS contains the system segment number (SSN) for the current code segment. This segment must be resident in physical memory for a process to be runnable.

SS - Stack Segment (register). SS contains the system segment number (SSN) for the current stack segment. This segment must be resident in physical memory for a process to be runnable.

SØ - Auxilary Segment Ø (register). SØ, if non-zero, contains the system segment number (SSN) for a segment, other than the code and stack segments, which is needed for a process to be runnable. Note: SSN Ø is reserved for the system segment address table, which is always resident; hence if SØ contains Ø, this indicates that no auxilary segment is needed.

S1 - Auxilary Segment 1 (register). Same as SØ.

PS - Parameter Size. PS is the number of words in an activation record which are used for parameters.

RPS - Result + Parameter Size. This is the number of words in an activation record which are used for function result and parameters.

LTS - Local + Temporary Size. LTS is the number of words in an activation record which are used for locals and temporaries (anonymous variables). (Note: the LTS of a main program body is always forced to Ø.)

AP - Activation Pointer (register). AP contains the physical address of the current activation record.

DL - Dynamic Link. This is the AP of the caller, respresented as an offset from SB.

SL - Static Link. This is the AP of the surrounding routine, represented as an offset from SB.

TP - Top Pointer (register). TP contains the physical address of the top of the run-time MStack.

TL - Top Link. TP of the caller, represented as an offset from SB.

GP - Global Pointer (register). Physical address of the GDB for the current code segment.

GL - Global Link. GP of the caller, represented as an offset from SB.

LP - Local Pointer (register). Physical address of the current activation record. When the LP is stored in an activation control block (ACB), it is represented as an offset from SB. Unlike other values in the ACB, the LP value is the current value of the Local Pointer, not some previous value.

XGP - eXternal Global Pointer. Pointer to another code segment's GDB, represented as an offset from SB.

XST - eXternal Segment Table. For a given program module, the XST translates ISNs to SSNs and XGPs.

RS - Return Segment. RS is the CS of the caller.

RA - Return Address. PC of the caller, represented as an offset from CB.

RR - Return Routine. RN of the caller.

RD - Routine Dictionary. Each code segment contains a routine dictionary which is indexed by RN. For each routine, the routine dictionary gives the lexical level (LL), entry address, exit address, parameter size (PS), result + parameter size (RPS), and local + temporary size (LTS).

ACB - Activation Control Block. The ACB contains housekeeping values in the activation record. It contains the SL, LP, DL, GL, RS, RA, and RR. In the ACB, the DL, GL, RS, RA, and RR are the AP, GP, CS, PC, and RN of the caller, respectively. The SL is the AP of the routine that surrounds the current one. The LP in the ACB is the current local pointer.

1.B Memory Organization

The PERQ's virtual memory system features a segmented 32 bit virtual address space mapped into a 2Ø bit physical address space. The segment is the unit of swappability, and comes in two types:

1) Code segments which are byte-addressed, read-only, and fixed in size with a maximum size of 64K bytes (32K words).

2) Data segments which are word-addressed, read-write, and variable in size with a maximum size of 64K words.

A PERQ process is a collection of up to 64K code and data segments. One of the data segments is the stack segment. Every process must have a stack segment and at least one code segment.

All segments are allocated in 256 word chunks and when in physical memory are aligned on 256 word boundaries. Note: A single segment must exist in contiguous memory, it may not be fragmented.

1.B.1 Memory Organization at the Process Level

The memory organization is designed with the following attributes in mind: 1) to allow separately compiled code segments to be grouped into a single process, 2) to allow code segments to be shared among processes, 3) to allow each code segment to have its own global variables, and 4) to allow one code segment to reference routines and global variables in other code segments. To achieve this, the following high-level characterisics are implemented:

1) All code is re-entrant.

2) Each code segment only refers to other code segments by internal (compiler-generated) segment numbers, which are not necessarily the same as the system-assigned segment numbers.

3) Each code segment in a process has its own global data block on the run-time stack.

4) Each code segment has an external segment table to permit referencing global variables and routines from other code segments.

1.B.1.a Global Data

At the global level, there is a Global Data Block (GDB) and an eXternal Segment Table (XST) associated with each code segment in a process. For a particular program module, the GDB contains the global variables, and the XST translates internal (compiler-generated) segment numbers (ISNs) to actual system segment numbers (SSNs) and eXternal Global Pointers (XGPs). In order to simplify the system, we devote a songle pointer to reference both the current GDB and XST. This Global Pointer (GP) points to the lowest address in the GDB and is ALWAYS aligned on a double word boundary.

```
              +--------------------------+
     SB ----->|                          |
              |   undetermined space     |
              |                          |
              +--------------------------+
              |   XST 1                  |
              +--------------------------+
              |   GDB 1                  |
              +--------------------------+
              |                          |
                         ....
              |                          |
              +--------------------------+
              |                          |
              |   XST i                  |
              |                          |
              +--------------------------+
     GP ----->|                          |
              |   GDB i                  |
              |                          |
              +--------------------------+
              |                          |
                         ....

                toward the top of stack
```

The XST for each segment is indexed by the internal segment numbers (ISNs). The entry is at GP - 2*ISN (Note: There is no entry for ISN Ø; ISN Ø always refers to the current segment). Each entry contains the offset from stack base (SB) of an external data block (XGP) and the actual system segment number (SSN) of the external segment. The XGP values are set by the linker, and the SSN values are set by the loader.

```
+---------------------------------+
|  eXternal Global Pointer (XGP)|
+---------------------------------+
|  System Segment Number (SSN)   |
+---------------------------------+
```

1.B.1.b Local Data

At the local level, there is an activation record, which consists of local variables, function result, parameters, temporaries (anonymous variables), the Activation Control Block (ACB), the previous EStack, and extra values that the routine may push and pop from the run-time stack. Three pointers are used to access and keep track of this information: the top-of-stack pointer (TP), the current-activation pointer (AP), and the local-variables pointer (LP).

```
               +-------------------------+
               | Result m-1              |
               +-------------------------+
               | Parameters m-1          |
               +-------------------------+
               | Locals m-1              |
               +-------------------------+
               | Temporaries m-1         |
               +-------------------------+
               | ACB m-1                 |
               +-------------------------+
               | EStack m-2              |
               +-------------------------+
               | Extra m-1               |
               +-------------------------+
     LP ----->|                         |
               | Result m                |
               |                         |
               +-------------------------+
               |                         |
               | Parameters m            |
               |                         |
               +-------------------------+
               |                         |
               | Local m                 |
               |                         |
               +-------------------------+
               |                         |
               | Temporaries m           |
               |                         |
               +-------------------------+
     AP ----->|                         |
               | ACB m                   |
               |                         |
               +-------------------------+
               |                         |
               | EStack m-1              |
               |                         |
               +-------------------------+
               |                         |
               | Extra m                 |
     TP ----->|                         |
               +-------------------------+

                 toward the top of stack
```

The function result, parameters, locals and temporaries are located by an offset from LP.

Each ACB has the following form:

```
+-----------------------------------------+
| Static Link (SL)                        |
+-----------------------------------------+
| Local Pointer (LP) (current)            |
+-----------------------------------------+
| Dynamic Link (DL)                       |
+-----------------------------------------+
| Global Link (GL)                        |
+-----------------------------------------+
| Top Link (TL)                           |
+-----------------------------------------+
| Return Segment Number (RS)              |
+-----------------------------------------+
| Return Address within Segment (RA)      |
+-----------------------------------------+
| Return Routine Number (RR)              |
+-----------------------------------------+

         toward the top of stack
```

The values in the ACB are the AP of the surrounding routine (SL), the current (not previous) LP, the AP of the caller (DL), the GP of the caller (GL), the TP of the caller (TL), the SSN of the caller (RS), the PC of the caller (RA), and the RN of the caller (RR). Note: When previous pointer values are saved in the ACB they are called links: SL, DL, GL, TL; Because the current (not previous) LP is stored in the ACB, it is called a pointer, not a link.

The EStack image immediately follows the ACB and looks like this:

```
+-------------------------+
| Number of Words Saved   |
+-------------------------+
| (ETOS)                  |
+-------------------------+
| (ETOS-1)                |
+-------------------------+
|                         |
          ....
|                         |
+-------------------------+
| (ETOS-n)                |
+-------------------------+

   toward the top of stack
```

1.B.1.c Run-Time Stack Organization

The following is an outline of the stack for a process of n segments, executing the mth routine call, which is in the ith segment:

```
                 +-------------------------+
       SB ---->  |                         |
                 | undetermined space      |
                 |                         |
                 +-------------------------+
                 | XST 1                   |
                 +-------------------------+
                 | GDB 1                   |
                 +-------------------------+
                 |                         |
                 |          ....           |
                 |                         |
                 +-------------------------+
                 |                         |
                 | XST i                   |
                 |                         |
                 +-------------------------+
       GP ---->  |                         |
                 | GDB i                   |
                 |                         |
                 +-------------------------+
                 |                         |
                 |          ....           |
                 |                         |
                 +-------------------------+
                 | XST n                   |
                 +-------------------------+
                 | GDB n                   |
                 +-------------------------+
                 | ACB Ø (main program)    |
                 +-------------------------+
                 | Extra Ø                 |
                 +-------------------------+
                 | Result 1                |
                 +-------------------------+
                 | Parameters 1            |
                 +-------------------------+
                 | Locals 1                |
                 +-------------------------+
                 | Temporaries 1           |
                 +-------------------------+
                 | ACB 1                   |
                 +-------------------------+
                 | EStack Ø                |
                 +-------------------------+
                 | Extra 1                 |
                 +-------------------------+
                |                        |
                           ....
                |                        |
                +------------------------+
      LP ----> |                        |
                |   Result  m            |
                |                        |
                +------------------------+
                |                        |
                |   Parameters  m        |
                |                        |
                +------------------------+
                |                        |
                |   Locals  m            |
                |                        |
                +------------------------+
                |                        |
                |   Temporaries  m       |
                |                        |
                +------------------------+
      AP ----> |                        |
                |   ACB  m               |
                |                        |
                +------------------------+
                |                        |
                |   EStack  m-1          |
                |                        |
                +------------------------+
                |                        |
                |   Extra  m             |
      TP ----> |                        |
                +------------------------+

                  toward the top of stack
```

1.B.2 Memory Organization at the System Level

The system makes use of two tables to control memory usage, the System Segment Address Table and the System Segment Information Table. The former contains all information which is needed by the Q-Code micro-code (location, size, resident, etc). The latter contains other information which is only referenced by the operating system (reference, I/O and lock counts, maximum size, etc).

1.B.2.a System Segment Address Table

The System Segment Address Table is a dynamic table, which is always resident in physical memory starting at physical address Ø. This table contains two words per segment, and contains all information that the Q-Code micro-code needs to know about each segment. The information contained in this table is:

1) Segment Base Address (upper 12 bits)

2) Segment Size (number of 256 word blocks - 1)

3) Flags
   - Not Resident
   - Recently Used
   - Moving
   - Sharable
   - Segment Kind
   - Segment Full
   - Segment Table Entry In Use

The Segment Base Address is the upper 12 bits of the physical address of the base of the segment. If the segment is not resident in physical memory, this field is undefined. The lower 8 bits of the Segment Base Address are always guarenteed to be zero (since all segments are aligned on 256 word boundaries).

The Segment Size plus one is the size of the segment in 256 word blocks (i.e., Segment Size Ø = 256 words).

The Flags have the following meanings and uses:

Not Resident - When true, this flag indicates that the segment is either swapped out or that the segment table entry is not in use. When false, this flag indicates that the entry is in use and the segment it describes is resident in physical memory. (See the "Segment Table Entry In Use" flag.)

Recently Used - This flag is set when a segment is accessed. It is used by the swapper to determine which segments are likely candidates to be swapped out when space is needed.

Moving - This flag, when true, indicates that the segment is being moved from one location in physical memory to another. If moving is true, Resident will be false. Moving is used only by the swapper to determine how to handle segment faults. (Not used by the Q-Code micro-code).

Sharable - When true, this flag indicates that a segment may be shared by several processes. (Not used by the Q-Code micro-code)

Segment Kind - This flag indicates whether the segment is a data or code segment. (Not used by the Q-Code micro-code)

Segment Full - This flag, when true, indicates that the entire data segment has allocated (via the Pascal New procedure). This flag is needed to distinguish full and empty data segment (and has no relevant meaning for code segments). (Not used by the Q-Code micro-code)

Segment Table Entry In Use - This flag is set true when the segment table entry contains a valid segment.

The arrangement of these fields within the two words are shown below:

```
Bit      15                       8 7                      0
        +--------------------------+------------------------+
Word Ø  | Base Addr (bits 8-15)    |        Flags           |
        +--------------------------+------------------------+

Bit      15                              4 3               0
        +---------------------------------+-----------------+
Word 1  |       Segment Size              | BA (16-19)      |
        +---------------------------------+-----------------+
```

The positions of the flags within the low byte of Word Ø are:

Bit	Flag
Ø	Resident
1	Moving
2	Recently Used
3	Sharable
4	Segment Kind
5	Segment Full
6	Table Entry In Use
7	not used

1.B.2.b System Segment Information Table

There is no information in the System Segment Information Table which is needed by the Q-Code micro-code; hence it is not described here. See the documentation on the Memory Manager.

1.B.2.c Code Segment Organization

A code segment contains the code for all routines in a segment and a routine dictionary which contains vital information about each of these routines.

The first word of every code segment is the offset from the base of the segment to the first word of the routine dictionary. The second word contains the number of routines which are defined in the segment. These two words are followed by the actual code which comprise the routines. Finally, the code is followed by the routine dictionary. The code is padded with Ø to 3 words of Øs (by the compiler) so that the routine dictionary is aligned on a quad-word boundary. This is possible since the compiler knows that the base of the segment will also be aligned on a quad-word boundary. It should also be noted that each entry in the dictionary is exactly 2 quad-words long (8 words). The routine dictionary is indexed by (Base Address of Dictionary)+8*RN. Each entry has the following form:

```
+--------------------------------+
| Parameter Size (PS)            |
+--------------------------------+
| Result + Parameter Size (RPS)  |
+--------------------------------+
| Local + Temporary Size (LTS)   |
+--------------------------------+
| Entry Address Within Segment   |
+--------------------------------+
| Exit Address Within Segment    |
+--------------------------------+
| Lexical Level (LL)             |
+--------------------------------+
| not used 1                     |
+--------------------------------+
| not used 2                     |
+--------------------------------+

        toward high memory
```

The Entry and Exit Addresses are the offsets from code base (CB) to the beginning of the routine and the beginning of the "terminate code" of the routine.

The following is a sample of a code segment containing 3 routines:

```
+----------------------------------+
| Pointer to Routine Dictionary |  >--+
+----------------------------------+    |
| Number of Routines (3)           |    |
+----------------------------------+    |
|                                  |    |
| Code for Routine 1               |    v
|                                  |    |
+----------------------------------+    |
|                                  |    |
| Code for Routine 2               |    v
|                                  |    |
+----------------------------------+    |
|                                  |    |
| Code for Routine 3               |    v
|                                  |    |
+----------------------------------+    |
| RD Entry for Routine 1           | <--+
+----------------------------------+
| RD Entry for Routine 2           |
+----------------------------------+
| RD Entry for Routine 3           |
+----------------------------------+

         toward high memory
```

1.C Error Handling and Fault Conditions

Errors and faults are handled by performing CALLVs to special routines (See section 4.D). The variable routine descriptors for these special routines can be found in a special table, the location and format of which will be known by the micro-code.

2. Instruction Format

Instructions on the Q-machine are one byte long followed by zero to four parameters. Parameters are either a signed byte (B : range -128 to 127), an unsigned byte (UB : range Ø to 255) or a word (W). Words need not be word aligned (unless specified). The low byte is first in the instruction byte stream.

Any exceptions to these formats are noted with the instructions where they occur.

3. Pointers

There are five different types of pointers, defined as follows: (Note: 2Ø bit offsets may only exist on the EStack).

Word Pointer: A 2Ø bit offset from StackBase (StackBase is the 2Ø bit physical address of the base of the stack).

Byte Pointer: A 2Ø bit offset from StackBase to the base of the byte array (TOS-1) and a byte offset into the array (TOS).

String Pointer: Same as a byte pointer.

Packed Field Pointer: A 2Ø bit offset from StackBase to the base of the word the field is in (TOS-1) and a one word field descriptor (TOS).

Field Descriptor:

Bits Ø-3: The field width (in bits) minus 1

Bits 4-7: The rightmost bit of the field.

Pascal Pointer: Obtained by declaring a variable as a pointer to another data type. (i.e., var I:^Integer;) (TOS-1) is the system segment number that contains the datum. (TOS) is the offset from the segment base to the datum.

Implementation Note: Stacks grow from low addresses to high addresses (i.e., if the address of TOS is 1Ø then the address of TOS-1 is 9 -- not 11).

4. QCode Descriptions

4.A Variable Fetching, Indexing, Storing and Transferring

4.A.1 Loads and Stores of One Word

4.A.1.a Constant One Word Loads

LDCØ..15		Ø-15	Load Word Constant. Pushes the value(Ø..15), with high byte zero, onto the EStack.
LDCN		22	Load Constant Nil. Pushes the value of NIL onto the EStack.
LDCMO		16	Load Constant -1.
LDCB	B	17	Load Constant Byte. Pushes the next byte on the EStack, with sign extend.
LDCW	W	18	Load Constant Word. Pushes the next word on the EStack.

4.A.1.b Local One Word Loads and Stores

LDLØ..15		1Ø9-124	Short Load Local Word. LDLx fetches the word with offset x in the current activation record and pushes it onto the EStack.
LDLB	UB	1Ø7	Load Local Word/Byte Offset. Fetches the word with offset UB in the current activation record and pushes it on the EStack.
LDLW	W	1Ø8	Load Local Word/Word Offset. Fetches the word with offset W in the current activation record and pushes it on the EStack.
LLAB	UB	125	Load Local Address/Byte Offset. Pushes a word pointer to the word with offset UB in the current activation record on EStack.
LLAW	W	126	Load Local Address/Word Offset. Pushes a word pointer to the word with offset W in the current activation record on EStack.
STLØ..7		129-136	Short Store Local Word. Store (ETOS) into word with offset x in the current activation record.
STLB	UB	127	Store Local Word/Byte Offset. Store (ETOS) into word with offset UB in the current activation record.
STLW	W	128	Store Local Word/Word Offset. Store (ETOS) into word with offset W in the current activation record.

Implementation Note: The address of the first local (offset Ø) is contained in the Local Pointer register (LP). The address of the Nth local is computed as (LP) + N.

4.A.1.c Own One Word Loads and Stores

LDOØ..15		139-154	Short Load Own Word. LDOx fetches the word with offset x in the current Global Data Block (GDB) and pushes it on the EStack.
LDOB	UB	137	Load Own Word/Byte Offset. Fetches the word with offset UB in the current Global Data Block (GDB) and pushes it on the EStack.
LDOW	W	138	Load Own Word/Word Offset. Fetches the word with offset W in the current Global Data Block (GDB) and pushes it on the EStack.
LOAB	UB	155	Load Own Address/Byte Offset. Pushes a word pointer to the word with offset UBin the current Global Data Block (GDB) on EStack.
LOAW	W	156	Load Own Address/Word Offset. Pushes a word pointer to the word with offset W in BASE activation record on EStack.
STOØ..7		159-166	Short Store Own Word. STOx stores (ETOS) into the word with offset x in the current Global Data Block (GDB).
STOB	UB	157	Store Own Word/Byte Offset. Stores (ETOS) into the word with offset UB in the current Global Data Block (GDB).
STOW	W	158	Store Own Word/Word Offset. Stores (ETOS) into the word with offset W in the current Global Data Block (GDB).

Implementation Note: The address of the first own (offset Ø) is contained in the Global Pointer register (GP). The address of the Nth own is computed as (GP)+N.

4.A.1.d Global One Word Loads and Stores

LDGB	UB1,UB2	192	Load Global Word/Byte Offset. Loads the word with offset UB2 in the Global Data Block (GDB) for program segment UB1 onto EStack.
LDGW	UB,W	193	Load Global Word/Word Offset. Same as LDGB except a full word offset is used.
LGAB	UB1,UB2	194	Load Global Address/Byte Offset. Pushes a word pointer to the word with offset UB2 in the Global Data Block (GDB) for program segment UB1 onto EStack.
LGAW	UB,W	195	Load Global Address/Word Offset. Same as LGAB except a full word offset is used.
LGAWW	W1,W2	181	Load Global Address/Word Segment, Word Offset. Same as LGAB except a full word is used both for the segment number and the offset.
STGB	UB1,UB2	196	Store Global Word/Byte Offset. Stores (ETOS) in word with offset UB2 in the Global Data Block (GDB) for program segment UB1.
STGW	UB,W	197	Store Global Word/Word Offset. Same as STGB except a full word offset is used.

Note: To achieve LDGW and STGW with full word segment numbers, use LGAWW with LDIND or STIND.

Implementation Note: Self-relative pointers to the Global Data Blocks (GDB) for each externally referenced segment are contained in the External Segment Table (XST), pointed to by the Global Pointer (GP). The address of the first global (offset Ø) in the designated GDB is computed as GP - 2 * ISN, where ISN (Internal Segment Number) is the program segment number specified in the load or store instruction. The Nth global is addressed by the base address (computes as above) plus N.

4.A.1.e Intermediate One Word Loads and Stores

LDIB	UB1,UB2	215	Load Intermediate Word/Byte Offset. UB1 indicates the number of static links to traverse to find the activation record to use. UB2 is the offset within the activation record of the desired word. The datum is pushed on EStack.
LDIW	UB,W	216	Load Intermediate Word/Word Offset. Same as LDIB except a word offset is used.
LIAB	UB1,UB2	217	Load Intermediate Address/Byte Offset. A word pointer is pushed on EStack (determined as in LDIB).
LIAW	UB,W	218	Load Intermediate Address/Word Offset. A word pointer is pushed on EStack (determined as in LDIW).
STIB	UB1,UB2	219	Store Intermediate Word/Byte Offset. Stores (ETOS) in memory (address determined as in LDIB).
STIW	UB,W	220	Store Intermediate Word/Word Offset. Stores (ETOS) in memory (address determined as in LDIW).

Implementation Note: The Activation Pointer register (AP) contains the address of the current Activation Control Block (ACB). Within the ACB is the Static Link (SL) to the previous ACB. To compute the address of the first intermediate word of the desired level, traverse the Static Links to the correct ACB. Within the ACB is the Local Pointer (LP) for that activation record.

4.A.1.f Indirect One Word Loads and Stores

STIND 21 Store Indirect. (ETOS) is stored into the word pointed to by word pointer (ETOS-1).

LDIND 173 Load Indirect. Word pointed to by word pointer (ETOS) is pushed on EStack.

4.A.2 Loads and Stores of Multiple Words

4.A.2.a Double Word Loads and Stores (Reals and Pointers)

LDDC	<block>	237	Load Double Word Constant. <block> is a double word constant. Load the constant onto EStack.
LDDW		239	Load Double Word. (ETOS) is a word pointer to a double word. The double word is pushed onto EStack.
STDW		183	Store Double Word. (ETOS),(ETOS-1) is a double word and (ETOS-2) is a word pointer to a double word block of memory. The double word is popped from ESTACK into the double word pointed to by (ETOS-2).

4.A.2.b Multiple Word Loads and Stores (Sets)

LDMC	UB,<block>	236	Load Multiple Word Constant. UB is the number of words to load, and <block> is a block of UB words, in reverse word order. Load the block onto the MStack.
LDMW		238	Load Multiple words. (ETOS-1) is a word pointer to the beginning of a block of (ETOS) words. Push the block onto the MStack.
STMW		182	Store Multiple Words. The MStack contains a block of (ETOS) words, (ETOS-1) is a word pointer to a similar block. Transfer the block from MStack to the destination block.

4.A.3 Byte Arrays

Note: A byte pointer is loaded onto the stack with a LLA, LOA or LGA of the base address of the array followed by the computation of the offset.

LDB		23	Load Byte. Push the byte (after zeroing the high Byte) pointed to by byte pointer (ETOS),(ETOS-1) on EStack.
STB		24	Store Byte. Store the low byte of (ETOS) into the location specified by byte pointer (ETOS-1),(ETOS-2).
MVBB	UB	167	Move Bytes/Byte Counter. (ETOS),(ETOS-1) is a source byte pointer to a block of UB bytes, and (ETOS-2),(ETOS-3) is the destination byte pointer to a similar block. Transfer the source block to the destination block.
MVBW		168	Move Bytes/Word Counter. Same as MVBB except (ETOS-1),(ETOS-2) is the source byte pointer, (ETOS-3),(ETOS-4) is the destination byte pointer, and (ETOS) is the number of bytes to transfer.

4.A.4 Strings

LSA UB,<chars> 19

Load String Address. UB is the length of the string constant <chars>. A string pointer is pushed on EStack (the virtual address of UB is pushed followed by a zero). UB is word aligned.

SAS 184

String Assign. (ETOS-1),(ETOS-2) is the source string pointer, and (ETOS-3),(ETOS-4) is the destination string pointer. (ETOS) is the declared length of the destination. The length of the source and destination are compared, and if the source string is longer than the destination a run-time error occurs. Otherwise all bytes of source containing valid information are transferred to the destination string.

LDCH 25

Load Character. (ETOS),(ETOS-1) is a string pointer. (ETOS) is checked to insure that is lies within the dynamic length of the string. If so, the character pointed to by (ETOS),(ETOS-1) is pushed; otherwise, a run-time error occurs.

STCH 28

Store Character. (ETOS) is a character and (ETOS-1),(ETOS-2) is a string pointer. (ETOS-1) is checked to insure that is lies within the dynamic length of the string. If so, the character (ETOS) is stored in the string, at the position pointed to by (ETOS-1),(ETOS-2); otherwise a run-time error occurs.

4.A.5 Record and Array Indexing and Assignment

MOVB	UB	169	Move Words/Byte Counter. (ETOS) is a word pointer to a block of UB words, and (ETOS-1) is a word pointer to a similar block. The block pointed to by (ETOS) is transferred to the block pointed to by (ETOS-1).
MOVW		170	Move Words/Word Counter. Same as MOVB except (ETOS-1) is the source pointer, (ETOS-2) is the destination pointer, and (ETOS) is the number of words to be transfered.
SIND0-7		173-180	Short Index and Load Word. SINDx indexes the word pointer (ETOS) by x words, and pushes the word pointed to by the result on ESTACK. (Note: SIND0 is synonymous to LDIND).
INDB	UB	171	Static Index and Load Word/Byte Index. Indexes the word pointer (ETOS) by UB words, and pushes the word pointed to by the result on ESTACK.
INDW	W	172	Static Index and Load Word/Word Index. Same as INDB except a full word index is used.
INCB	UB	232	Increment Field Pointer/Byte Index. The word pointer (ETOS) is indexed by UB words and the resultant pointer is pushed on ESTACK.
INCW	W	233	Increment Field Pointer/Word Index. Same as INCB except a full word index is used.

Note: INCB and INCW are equivalent to add UB or W to (ETOS).

IXAB	UB	222	Index Array/Byte Array Size. (ETOS) is an integer index, (ETOS-1) is a word pointer to the base of the array, and UB is the size (in words) of an array element. A word pointer to the first word of the indexed element is pushed on ESTACK.
IXAW		223	Index Array/Word Array Size. Same

			as IXAB except (ETOS-1) is the integer index, (ETOS-2) is the word pointer to the base of the array, and (ETOS) is the size (in words) of an array element. (Gen1A) full word is used for the array element size.
IXA2..4		224-226	Index Array/Short Array Size. Same as IXAB except array element sizes are fixed at 2-4.
IXP	UB	214	Index Packed Array. (ETOS) is an integer index, and (ETOS-1) is a word pointer the base of the array. Bits 4-7 of UB contain the number of elements per word minus 1, and bits Ø-3 contain the field width (in bits) minus 1. Compute and push a packed field pointer.
LDP		26	Load a Packed Field. Push the field described by the packed field pointer (ETOS),(ETOS-1) on ESTACK.
STP		27	Store into Packed Field. Store (ETOS) in the field described by the packed field pointer (ETOS-1),(ETOS-2).
ROTSHI	UB	221	Rotate/Shift. (ETOSi-1) is the argument to be rotated or shifted, and (ETOS) is the distance to rotate or shift. If UB is Ø then a right rotate occurs, and if UB is 1 then a shift occurs. The direction of the shift is determined from (ETOS); If (ETOS) >= Ø then a left shift occurs, otherwise a right shift.

4.B Top of Stack Arithmetic and Comparisons

4.B.1 Logical

LAND	30	Logical Add. AND (ETOS) into (ETOS-1).
LOR	31	Logical Or. OR (ETOS) into (ETOS-1).
LNOT	32	Logical Not. Take one's complement of (ETOS).
EQUBOOL	33	Boolean =,
NEQBOOL	34	<>,
LEQBOOL	35	<=,
LESBOOL	36	<,
GEQBOOL	37	>=,
GTRBOOL	38	and > comparisons. Compare (ETOS-1) to (ETOS) and push true or false on ESTACK.

4.B.2 Integer

ABI	71	Absolute Value of Integer. Take absolute value of (ETOS). Result is undefined if (ETOS) is initially -32768.
ADI	72	Add Integers. Add (ETOS) and (ETOS-1).
NGI	73	Negate Integer. Take the two's complement of (ETOS).
SBI	74	Subtract Integers. Subtract (ETOS) from (ETOS-1).
MPI	75	Multiply Integers. Multiply (ETOS) and (ETOS-1). This instruction may cause overflow if the result is larger than 16 bits.
DVI	76	Divide Integers. Divide (ETOS-1) by (ETOS) and push quotient (as defined by Jensen and Wirth).
MODI	77	Modulo Integers. Divide (ETOS-1) by (ETOS) and push the remainder (as defined by Jensen and Wirth).
CHK	78	Check Against Subrange Bounds. Insure that (ETOS-1) <= (ETOS-2) <= (ETOS), leaving (ETOS-2) on top of the stack. If conditions are not met a run-time error occurs.
EQUI	39	Integer =,
NEQI	40	<>,
LEQI	41	<=,
LESI	42	<,
GEQI	43	>=,
GTRI	44	and > comparisons. Compare (ETOS-1) to (ETOS) and push true or false on ESTACK.

4.B.3 Reals

All over/underflows cause a run-time error.

FLT	79	Float. The integer (ETOS) is converted to a floating point number and pushed onto EStack.
TNC	80	Truncate Real. The real (ETOS),(ETOS-1) is truncated (as defined by Jensen and Wirth), converted to an integer, and pushed onto EStack.
RND	81	Round Real. The real (ETOS),(ETOS-1) is rounded (as defined by Jensen and Wirth), truncated and converted to an integer, and pushed onto EStack.
ABR	82	Absolute Value of Reals. Take the absolute value of the real (ETOS),(ETOS-1).
ADR	83	Add Reals. Add (ETOS),(ETOS-1) and (ETOS-2),(ETOS-3).
NGR	84	Negate Real. Negate the real (ETOS),(ETOS-1).
SBR	85	Subtract Reals. Subtract (ETOS),(ETOS-1) from (ETOS-2),(ETOS-3).
MPR	86	Multiply Reals. Multiply (ETOS),(ETOS-1) and (ETOS-2),(ETOS-3).
DVR	87	Divide Reals. Divide (ETOS-2),(ETOS-3) by (ETOS),(ETOS-1).
EQUREAL	45	Real =,
NEQREAL	46	<>,
LEQREAL	47	<=,
LESREAL	48	<,
GEQREAL	49	>=,
GTRREAL	50	and >

comparisons. Push true or false on ESTACK.

4.B.4 Sets

ADJ	UB	185	Adjust Set. The set on the top of the MSTACK is forced to occupy UB words, either by expansion or compression, and its length word is popped from ESTACK.
SGS		66	Build Singleton Set. The integer (ETOS) is checked to insure that Ø <= (ETOS) <= 32,767, the set [(ETOS)] is pushed on MSTACK, and the size of the set is pushed on ESTACK. If (ETOS) is out of range, the null set is pushed (a zero is pushed on ESTACK, the MSTACK is not altered).
SRS		68	Build SubRange Set. The integers (ETOS) and (ETOS-1) are checked as in SGS, the set [(ETOS-1)..(ETOS)] is pushed onto MSTACK, and the size of the set is pushed on ESTACK. (The null set is pushed if (ETOS-1) > (ETOS) or either is out of range).
INN		88	Set Membership. See if integer (ETOS) is in set contained on the top of MSTACK, and with length (ETOS-1), pushing TRUE or FALSE on ESTACK.
UNI		89	Set Union. The union of the two sets contained on the top of MSTACK, and sizes (ETOS) and (ETOS-1) is pushed on MSTACK, and the length of the result on ESTACK.
INT		9Ø	Set Intersection. The intersection of the two sets contained on the top of MSTACK, and sizes (ETOS) and (ETOS-1) is pushed on MSTACK, and the length of the result on ESTACK.
DIF		91	Set Difference. The difference of the two sets contained on the top of MSTACK, and sizes (ETOS) and (ETOS-1) is pushed on MSTACK, and the length of the result on ESTACK.
EQUPOWR		63	Set =,
NEQPOWR		64	<>,

LEQPOWR 65 <= (subset of),

GEQPOWR 67 and >= (superset of) comparisons of the two sets on top of ESTACK, with sizes (ETOS) and (ETOS-1).

4.B.5 Strings

EQUSTR	51	String =,
NEQSTR	52	<>,
LEQSTR	53	<=,
LESSTR	54	<,
GEQSTR	55	>=,
GTRSTR	56	and > comparisons. The string pointed to by string pointer (ETOS-2),(ETOS-3) is lexicographically compared to the string pointed to by string pointer (ETOS),(ETOS-1).

4.B.6 Byte Arrays

EQUBYT	UB	57	Byte Array =,
NEQBYT	UB	58	<>,
LEQBYT	UB	59	<=,
LESBYT	UB	6Ø	<,
GEQBYT	UB	61	>=,
GTRBYT	UB	62	and > comparisons. <=, <, >=, and > are only emitted for packed arrays of char. The argument, UB, if non-zero, is the size of the array. If UB is equal to Ø then (ETOS) is the size of the array.

4.B.7 Array and Record Comparisons

EQUWORD UB	69	Word or multiword structure = and <> comparisons. The argument, UB, if non-zero, is the size of the array. If UB equals Ø, then (ETOS) is the size of the array.
NEQWORD UB	7Ø	

4.C Jumps

JMPB	B	204	Unconditional Jump/Byte Offset. B is added to the IPC. Negative values of B cause backward jumps.
JMPW	W	205	Unconditional Jump/Word Offset. W is added to the IPC. Negative values of W cause backward jumps.
JFB	B	206	False Jump/Byte Offset. Jump (as in JMPB) if (ETOS) is false.
JFW	W	207	False Jump/Word Offset. Jump (as in JMPW) if (ETOS) is false.
JTB	B	208	True Jump/Byte Offset. Jump (as in JMPB) if (ETOS) is true.
JTW	W	209	True Jump/Word Offset. Jump (as in JMPW) if (ETOS) is true.
JEQB	B	210	Equal Jump/Byte Offset. Jump (as in JMPB) if integer (ETOS) equals (ETOS-1).
JEQW	W	211	Equal Jump/Word Offset. Jump (as in JMPW) if integer (ETOS) equals (ETOS-1).
JNEB	B	212	Not Equal Jump/Byte Offset. Jump (as in JMPB) if integer (ETOS) is not equal to (ETOS-1).
JNEW	W	213	Not Equal Jump/Word Offset. Jump (as in JMPW) if integer (ETOS) is not equal to (ETOS-1).
XJP	W1,W2,W3,<Case Table>	100	Case Jump. W1 is word-aligned, and is the minimum index of the table. W2 is the maximum index. W3 is the offset to the code to be executed if the case specified has no entry in the case table. The case table is W2 - W1 + 1 words long and contains offsets to the code to be executed for each case. If (ETOS), the actual index, is not in the range W1..W2 then W3 is added to IPC. Otherwise (ETOS) - W1 is

used as an index into the case table and the index entry is added to IPC.

4.D Routine Calls and Returns

Note: There can be at most 256 routines in a segment.

CALL	UB	186	Call Routine. Call routine UB, which is in the current segment.
CALLXB	UB1,UB2	234	Call External Routine/Byte Segment. UB1 is the internal segment number (ISN) which contains the routine numbered UB2 to be called. First the ISN is translated to the correct SSN, and residency of that segment is checked. If the segment is resident, the call proceeds; if not, SØ is loaded with the SSN, S1 is cleared, the PC is backed up so that the call will be re-executed, and a segment fault occurs. The second attempt is guarenteed to suceed, since the process will be unable to resume execution until the segment in SØ is resident.
CALLXW	W,UB	235	Call External Routine/Word Segment. Same as CALLXB except the internal segment number (ISN) is given in a full word.
LVRD	W,UB1,UB2	98	Load Variable Routine Descriptor. This Q-Code pushes a Variable Routine Descriptor on the EStack for the routine UB1 in segment ISN W, at lexical level UB2. The following values (which comprise a variable routine descriptor) are pushed: ETOS = System Segment Number (SSN); ETOS-1 = Global Pointer, represented as an offset from SB; ETOS-2 = Routine Number; and ETOS-3 = Static Link (determined as if a call were actually performed to the routine here).
CALLV		187	Call Variable Routine. (ETOS) -- (ETOS-3) are a variable routine descriptor (as described above in LVRD). Residency of the segment are checked. If the segment is resident, the call is made as will CALL, except the GP and SL are taken from the variable routine descriptor; if not, a segment fault

occurs as with CALLX.

RETURN		200	Return from Routine. Return from the current routine. If the routine was a function, the function value is left on the top of the MStack. Since the first word of a code segment is not code, but an offset to the routine dictionary, if the RA which is being returned to is 0, the return is performed to the exit code of that routine. (This will prove useful for the EXIT and EXGO Q-Codes described below).
EXIT	W,UB	92	Exit from Routine. Exit from all routines up to and including the most recent invokation of the routine UB in ISN W. This is accomplished by setting the RAs in all the ACBs to 0, from the most recent through and including the first ACB which was created from an invokation the routine to be exitted, and jumping to the exit code of the current routine.
EXGO	W1,UB,W2	29	Exit and Goto. Exit from all routines up to, but not including, routine UB in ISN W1, and then jump to the instruction with offset W2 from CB. The implementation is similar to EXIT, except the last RA modified is loaded with W2.

4.E Systems Programs Support Procedures

NOOP	93	No-Operation.
REPL	94	Replicate. Replicate (ETOS).
REPL2	95	Replicate Two. Replicate two top-of-estack words (i.e., first push original (ETOS-1), then push original (ETOS)).
MMS	96	Move to Memory Stack. Push (ETOS) onto MTOS (16 bit transfer).
MES	97	Move to Expression Stack. Push (MTOS) onto ETOS (16 bit transfer - top 4 bits are zeroed).
MMS2	201	Move Double to Memory Stack. Transfer the top two words from the EStack to the MStack. The order is reversed; old (ETOS) will be (MTOS-1), (ETOS-1) will be (MTOS).
MES2	202	Move Double to Expression Stack. Transfer the top two words from the MStack to the EStack. THe order is reversed; old (MTOS) will be (ETOS-1), (MTOS-1) will be (ETOS).
PSW	101	Process Switch.
RASTER-OP	102	(ETOS) and (ETOS-1) are word pointers to RODef records. (ETOS-2) and (ETOS-3) are the height and width respectively of the transfer to be performed. (ETOS-4) is the function to be performed. RODef Record: First 2 words are the virtual address to the first word of the image area. The next word is the length of a scan line in words. Next 2 words are the X and Y bit offsets of window to be transferred. Function Codes:

0 - Source

1 - NOT Source

2 - Destination AND Source

3 - Destination AND (NOT Source)

4 - Destination OR Source

5 - Destination OR (NOT Source)

6 - Desintation XOR Source

7 - Destination XNOR Source

The above functions can replace, merge, erase, or compliment black on white or white on black data (Note: a "1" is a black pixel, a "0" is a white pixel).

Note: scan lines are word aligned.

STARTIO	103	(ETOS) is the channel on which to start IO. Performs a block if IO is to be synchronous.
BLOCK	104	Return control to the schedular. Set process as not runable (until set runable by some other process). More to be added.
INTOFF	105	Disable interrupts.
INTON	106	Enable interrupts.
EXCH	230	Exchange. (ETOS) and (ETOS-1) are swapped.
EXCH2	231	Exchange Double. The pair (ETOS) and (ETOS-1) are swapped with the pair (ETOS-2) and (ETOS-3).
TLATE1	227	Translate Top of Stack. (ETOS),(ETOS-1) is a virtual address. If the segment SSN (ETOS-1) is resident, convert the virtual address to an offset from stack base (SB) and execute the next Q-Code (what ever it may be), with out interrupts, to competion. If

the segment SSN (ETOS-1) is non-resident, restore the EStack to its previous state, backup the PC to re-execute the TLATE1 and perform a segment fault.

TLATE2 228 Translate Top of Stack - 1. Same as TLATE1 except the virtual address is at (ETOS-1),(ETOS-2).

TLATE3 229 Translate Top of Stack - 2. Same as TLATE1 except the virtual address is at (ETOS-2),(ETOS-3).

STLATE UB 24Ø Special Translate. This translate is similar to the previous translate Q-Codes, except is can specify a greater depth that TLATE3, and that it may specify the translation of 2 virtual addresses. Each half of UB is interpretted as the depth of the System Segment Number word of the virtual address to be translated (prior to any stack alteration). A depth of Ø indicates no translation. All segments specified in the STLATE must be resident before any translations occur, otherwise a segment fault occurs. Note, if both nibbles of UB are non-zero then the low order nibble (bits Ø-3) must be less than the high order nibble (bits 4-7).

LSSN 99 Load Stack Segment Number. Pushes the system segment number of the MStack onto EStack.

LDTP 2Ø3 Load Top Pointer (plus 1). Pushes the value of Top Pointer (TP) plus 1 onto EStack.

ATPB SB 188 Add to Top Pointer/Byte Value. Adds SB to TP.

ATPW 189 Add to Top Pointer/Word Value. Adds (ETOS) to TP.

INDEX