Supplement to the
645 Processor Manual

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
INFORMATION PROCESSING CENTER

May 22, 1973

SUMMARY OF THE H6180 PROCESSOR

INTRODUCTION

This memo presents a summary of the H6180 processor from the point of view of the subsystem writer who must write or debug code at the assembly language level. The H6180 processor represents an extension and general cleanup of the H645, and this memo assumes that the reader is reasonably familiar with the H645.*

Experience with the H645 and the Multics system has uncovered a number of areas in which performance and maintainability could be substantially improved with certain deviations from the H645 specification. The following areas were felt to be of sufficient importance to warrant changes to the H645 specification.

1. Hardware aids for improved compatibility with standard H600 and H6000 line software.

2. Refinements to the paging and segmentation mechanisms to improve overall system performance.

3. Complete implementation of the Multics protection ring mechanism in the H6180 processor to improve system performance and reduce software complexity.

4. Incorporation of the H6080 Extended Instruction Set (EIS)* within the H6180 processor. The EIS provides a comprehensive set of instructions for bit and character string manipulation as well as decimal arithmetic.

These changes are described in detail on the following pages.

*For more information, refer to the Honeywell Model 645 Processor Reference Manual and the Series 6000 Extended Instruction Set Processor manual.

1. Hardware Compatibility with H600/H6000

The issue of compatibility with product line software is really two issues stemming from two distinctly different motivations. One issue is that of "stand-alone compatiblity" which allows the running of standard product line software (e.g., GCOS, T and D monitor, etc.) on a stand-alone machine. The other issue is that of "slave program compatibility" which allows for the efficient execution of H600/6000 slave programs under the control of the Multics system. A compatibility switch on the H6180 processor handles the problem of stand-alone compatiblity while a program-settable mode is provided as a software aid in handling the problem of slave program compatibility.

2. Modifications to the Paging and Segmentation Hardware

The H6180 contains a number of modifications to the H645 paging and segmentation hardware to improve the performance of the Multics software and to simplify some areas now felt to have been overdesigned in the H645. The changes to the H645 specification are summarized below and are described in more detail later in this memo.

2.1 The address field in the segment descriptor word is extended to a full 24-bit absolute address to allow page tables (and unpaged segments) to begin on any main memory address. This modification allows the software a great deal more flexibility in the allocation of page table space and greatly reduces the amount of main storage reserved for page tables.

2.2 The processor supports only a single page size rather than the two page sizes supported by the H645. However, provision is made to allow the page size to be modified by field engineering in an orderly and well understood fashion.

2.3 Each of the eight address base registers (now called Pointer Registers) of the H6180 processor is extended to contain both a segment number and a word number portion. The H645 concept of internal and external base registers and control fields is dropped. Each of the eight address base registers on the H6180 processor behaves as a 645 base register pair. In addition, each of the new address base registers contains a bit offset field for use by the new EIS instructions.

2.4 Some minor changes have been made to the definition of the 645 master and slave modes which are renamed respectively to be the privileged and unprivileged modes to avoid confusion with H6000 terminology. A minor change has also been made in the treatment of execute-only segments to allow entry at locations other than zero.

2.5 The access control information contained in the H645 page table word (not used in Multics) has been removed. In the H6180, all access control is implemented in the segment descriptor word.

3. Hardware Implementation of Multics Ring Protection

The Multics concept of protection rings, ring crossing and argument validation has been implemented as an integral part of the segmentation hardware on the H6180 processor. The hardware implementation of rings is really a further modification to the H645 segmentation hardware. However, the modification is introduced separately at this point since it involves perhaps the most significant deviation from the H645 specification and, as such, deserves somewhat more motivation. In the H645 version of Multics, the ring protection mechanism is, of necessity, completely simulated by the Multics software. The H645 Multics maintains, in parallel, separate descriptor segments for each ring of each process. The ring crossing is simulated by a rather costly and complex fault processing mechanism which includes the copying and validation of argument pointers and the switching of descriptor segments to simulate the effect of switching rings. The cost of the current simulation amounts to approximately 5 to 10 percent of the useful chargeable CPU time and contributes substantially to the overall complexity of the system. In the H6180 processor, ring crossing and argument validation are handled directly by the hardware without costly software intervention. As a result, a call to an inner ring requires no more CPU time than a call to a procedure in the same ring.

4. The Extended Instruction Set

The H6080 and the H6180 include instructions for string manipulation and decimal arithmetic. The eight address registers of the H6080 are a direct subset of the eight address base registers of the H6180 processor (also extended to contain bit offset fields).

The H6180 permits Multics programs to take full advantage of the new instructions in a manner which is sympathetic to the Multics environment. In addition, H6080 slave programs which use the new instructions will be able to continue to run efficiently under Multics.

SLAVE PROGRAM COMPATIBILITY WITH THE H6000 SERIES

Many H635 and H6000 slave programs can be run on the H645 version of Multics (e.g., GCOS and the Dartmouth System). The H635 (or H6000) slave program is loaded into a single paged segment and Multics library programs are provided to intercept faults generated by the 635 program and provide the user-callable functions of the GCOS or DTSS supervisor. This technique works quite well for most slave programs. However, a number of 635 slave programs (particularly subsystems such as BASIC and the DTSS debugger) make indirect use of the H635 Base Address Register (BAR) by calling the supervisor to set the BAR to span a subset of the current user core image (e.g., to protect the subsystem from its user). In the H645 there is no general and effecient way to simulate the address relocation and protection of the BAR within a single paged segment. Therefore an H600/6000 BAR compatibility mode is provided on the H6180 which operates in the following manner when the H6180 is set to run in Multics mode.

1. A program-settable indicator is introduced to control the use of the BAR. The normal and usual setting of this indicator is ON, and in this state the contents of the BAR is ignored. When the indicator is OFF, the processor is said to be in BAR mode.

2. When the processor is in BAR mode, the word number of each computed address is checked against the bound field of the BAR and augmented by the BAR base field. This operation cannot be interrupted and is done before the address is used to select the proper page table word. A boundary violation causes a "store" fault.

3. The contents of the BAR are settable by the normal H6000 LBAR instruction which is unprivileged when the H6180 is in Multics mode.

4. The BAR mode is turned on by the Transfer and Set Slave (TSS) instruction. (The contents of the BAR apply only to the final effective address of the TSS instruction.) The TSS instruction is also unprivileged in Multics mode.

5. The BAR mode is exited by a fault or an interrupt, and can only be reentered by an RCU or TSS instruction.

Many H600/6000 slave programs are written to expect certain externally generated faults and interrupts (e.g., timer runout) to be reflected to the slave program through a software-simulated fault vector. To avoid race conditions these programs commonly use bit 28 of the instruction word to inhibit external faults and interrupts. Furthermore, such programs are unprepared to handle interrupts in mid-instruction. So that H600/6000 slave programs can be run efficiently under Multics, the function of the BAR mode is extended as follows.

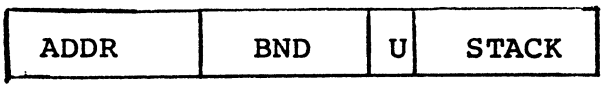
6. When in BAR mode, an unprivileged program (e.g., a 635 slave program) is allowed to inhibit interrupts via the use of bit 28 of the instruction word. In any mode, asynchronous interrupts encountered in mid-instruction are delayed until the instruction is complete. (In fact interrupt sampling is done only after an instruction pair completes execution, i.e., after most "odd" instructions.)

SEGMENTATION ON THE H6180

This section describes the segmentation hardware of the H6180. In most respects, the mechanism is quite similar to the H645 appending hardware with the addition of some refinements to improve the performance of the system software. The single substantial deviation from the H645 specification is the addition of hardware to implement the Multics protection ring mechanism.

1. The Descriptor Segment Base Register (DSBR)

As in the H645 the base of the descriptor segment is located through the DSBR which on the H6180 specifies the following information.



DSBR.ADDR — specifies the full 24-bit address of the first double word of either a page table (for a paged descriptor segment) or the first location of the descriptor segment itself (for an unpaged descriptor segment).

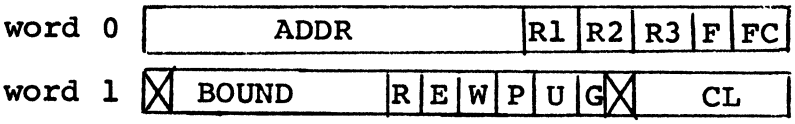
DSBR.BND — specifies the highest 0 mod 16 descriptor segment address which can be accessed without causing an out of bounds fault.

DSBR.U — specifies whether the descriptor segment is paged (= 0) or unpaged (= 1).

DSBR.STACK — is used in conjunction with the new CALL6 instruction. Whenever the H6180 processes a CALL6 instruction which requires the processor to switch to an inner ring (an inward call), the segment number of the stack for the inner (target) ring is computed by concatenating DSBR.STACK to the left of the ring number of the target ring. For example, if DSBR.STACK is 57 (octal) and the target ring is 3, the H6180 will compute the segment number of the stack for the target ring as 573 (octal). This facility is required to allow ring crossing to be fully automatic and places certain requirements on the segment numbers assigned to segments used as stacks.

2. The Segment Descriptor Word (SDW)

As in the H645 each segment (when active in main storage and known to a process) is described to the processor by a segment descriptor word (SDW) which is located by using the segment number of the segment as an index into the descriptor segment of the process. To accomodate the hardware-implemented ring crossing and argument validation and other changes, the SDW has been extended to the 72-bit double word described below.



SDW.ADDR (0-23) Is a full 24-bit absolute address and specifies the core address of either a page table (for a paged segment) or the first location of an unpaged segment.

SDW.R1 (24-26) Specifies the highest ring number of the read/write bracket for this segment (0-R1, see (3) on rings and ring brackets).

SDW.R2 (27-29) Specifies the highest ring number of the read/execute bracket of this segment (R1-R2, see (3)).

SDW.R3 (30-32) Specifies the highest ring number of the call bracket of this segment ((R2+1) - R3, see (3)).

SDW.F (33) Is a directed fault indicator and if off (= 0) specifies that the processor is to execute the directed fault specified in the FC field (see below).

SDW.FC (34-35) Indicates (if F is off) which of the 4 directed faults (DF0-DF3) the processor is to execute. (Note that the 645 provided for 8 directed faults.)

SDW.BOUND (1-14) Is the boundary field and indicates the highest 16-word block of the segment which can be addressed without causing an out-of-bounds fault. If the high order 14 bits of an address to this segment is greater than the value of the boundary field, an out-of-bounds fault is generated. (The boundary field could be maintained to the nearest word but special checks would have to be made for instructions which reference two or more continguous words.) A boundary field of 14 bits is chosen because some instructions (e.g., the new version of STB) reference up to 16 contiguous words. A further implication is that the software is expected to allocate unpaged segments on a zero mod 16 word boundary.

SDW.R (15) Is the read-permit indicator. Data fetches by other segments are permitted to this segment only if this indicator is on (= 1) and if the processor is executing in a ring less than or equal to R2 (i.e., within the read/write or read/execute bracket, see (3)).

SDW.E (16) Is the execute-permit indicator. Instruction fetches from this segment are permitted only if this indicator is on (= 1) and if the processor is executing in a ring greater than or equal to R1 and less than or equal to R2 (i.e., within the read/execute bracket, see below). Note that when the E indicator is on and the R indicator is off, the segment is to be treated as an "execute-only" procedure segment. An execute-only procedure segment is permitted to reference data within itself (i.e., within the same segment) in spite of the lack of the read indicator. However, read permission is denied to other segments. (Note: targets of XEC and XED instructions can exist in segments with only the R bit on.)

SDW.W (17) Is the write-permit indicator. Attempts to store into this segment are honored only if this indicator is on (= 1) and if the processor is executing in a ring less than or equal to R1 (i.e., within the read/write bracket, see below).

SDW.P (18) Is the privileged mode indicator. If this indicator is on (= 1) and if the processor is executing in ring 0, the procedure segment is permitted to execute privileged instructions and inhibit interrupts under control of bit 28. Privileged procedures need no further powers and are subject to all other access checking (read, write permission bounds checking, etc.). Since privileged procedures can only be executed in ring 0, it is no longer necessary to limit calls to privileged procedures to enter via word 0 of the segment.

SDW.U (19) Indicates whether the segment is paged (= 0) or unpaged (= 1). If the segment is unpaged, ADDR is the full absolute address of the first word (word 0) of the segment. If the segment is paged, ADDR is the full absolute address of the beginning of the page table for the segment.

SDW.G (20) Is the gate indicator and if off (= 0) any call to this segment from a different segment must be directed to an address value less than the value of the CL field (see below).

SDW.CL (22-35) Is the call limiter. If G is off any external transfer to this segment via the new CALL6 instruction (described below) must be directed to a word number less than the value of this field.

3. Rings and Ring Brackets

A Multics process consists of procedure and data segments which are all directly addressable through the descriptor segment of that process. However, a process may only access a segment when the process is running at an appropriate level of privilege. For example, all of the segments of the hardcore supervisor are shared and accessible to all Multics processes but only when executing at the highest level of privilege.

The Multics system allows segments to be grouped into an ordered set of collections called rings in which segments requiring the highest level of privilege to reference can only be accessed from within the innermost ring of the set. Each ring is identified with a ring number designating the required level of privilege necessary to access segments in that ring. In Multics, the ring with the highest privilege is ring 0 which contains the procedures and data bases of the hardcore supervisor. Each user process has at least two rings, one for the hardcore supervisor and one for user programs and data. The user process may generate more rings (levels of lesser privilege) if desired.

Frequently, it is useful to allow a segment to be accessible in more than one ring. For example, it is often useful for a data base which is writeable in an inner ring to be readable in an outer ring. For this reason, the concept of ring brackets were introduced.

The access of a user to a specific segment is controlled by two quantities: the access attributes (e.g., read, execute, write) and the ring brackets. The ring brackets of a segment are specified by three integers (R1, R2 and R3) each of which must be greater than or equal to the preceding number. The first number (R1) specifies the top (highest ring number) of the read/write bracket, the second number (R2) specifies the top of the read/execute bracket and the last number (R3) specifies the top of the call bracket.

3.1 The Read/Write Bracket (Rings 0-R1)

Attempts to read or write a segment by a procedure executing in a ring within the read/write bracket are allowed if the appropriate (read or write) access indicators are on for the segment being referenced. Execution of a procedure in a ring within this bracket is permitted only at the top of the read/write bracket (R1) which is also the bottom of the read/execute bracket. Note that the highest ring from which a segment can be written is specified by R1. As a result, the data in the segment is no more reliable than the procedure segments which operate in that ring.

3.2 The Read/Execute Bracket (Rings R1-R2)

Attempts to read or execute (transfer to) a segment by a procedure executing in a ring within the read/execute bracket are allowed if the appropriate (read and execute) indicators are on for the segment being referenced. Writing of a segment within its read/execute bracket is permitted only from the ring at the bottom of the bracket (R1) which is also the top of the read/write bracket. If R2 is equal to R1 the read/execute bracket specifies a single ring (R1).

3.3 The Call Bracket (Rings (R2 + 1) - R3)

Attempts to call a (procedure) segment from another segment executing in a ring above the read/execute bracket, but within the call bracket of the procedure to be called, are allowed if the execute indicator of the procedure to be called is on and if the new CALL6 instruction (described below) is used. When the CALL6 instruction is directed to a procedure in an inner ring which has the appropriate execute access and call bracket, the processor will automatically switch to the ring specified as the R2 of the procedure being called. The call bracket and the CALL6 instruction are the only means (except for faults) by which control can be passed from an outer ring to an inner (more privileged) ring. If R3 is equal to R2, the call bracket is null and the procedure cannot be called from an outer ring.

3.4 Summary

Assuming that the appropriate (read, execute or write) indicators are on, the following list summarizes the effects of the three ring brackets.

3.4.1 Writing is permitted from a ring within the read/write bracket only (i.e., if ring≤R1).

3.4.2 Reading is permitted from a ring within the read/write bracket or the read/execute bracket (i.e., ring≤R2).

3.4.3 Execution (or transfer of control) is permitted only from a ring within the read/execute bracket (i.e., R1≤ring≤R2).

3.4.4 Calling (via CALL6 only) is permitted from a ring within the read/execute or call brackets (i.e., R1≤ring≤R3).

3.4.5 The CALL6 instruction is the only instruction which may be used to access a segment in a ring within its call bracket (i.e., R2<ring≤R3).

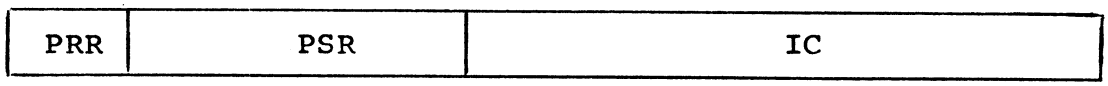
3.4.6 No access is permitted to a segment from a ring higher than the call bracket (i.e., ring>R3).

4. Processor Address Registers

Like the 645, the H6180 processor has 10 address or pointer registers (PR's). Eight of these pointer registers can be directly accessed and modified by the software, one is used to locate the current instruction and one is used exclusively by the processor for effective address calculations. Unlike the 645, each of the eight program addressable pointer registers specifies a full segmented address including the segment number and the word number in a single pointer register.

4.1 The Procedure Pointer Register

The procedure pointer register (PPR) is used by the processor to locate the current instruction and may be modified by the software to effect a transfer of control. The PPR is actually an extension of the PBR and IC of the 645. The contents of the 36-bit PPR are outlined below.



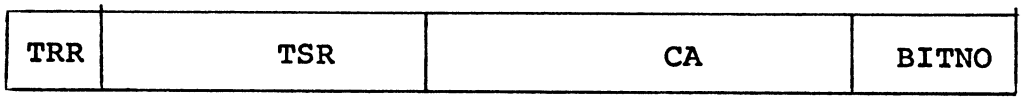
PPR.PRR (3 bits) Is the procedure ring register and specifies the ring (level of privilege) in which the processor is currently executing. PRR may be set to a higher value only by an RTCD or RCU instruction. It may be set to a lower value only by a CALL6 instruction (see below) or by a fault or interrupt.

PPR.PSR (15 bits) Is the procedure segment register (same as the PBR in the 645) and specifies the segment number of the current procedure segment.

PPR.IC (18 bits) Is the instruction counter (same as in the 645).

4.2 The Temporary Pointer Register

The temporary pointer register (TPR) is used exclusively by the processor for operand address calculations and serves the same general purpose as the TBR and computed address (CA) of the 645. It should be noted here that the addition of multiple-operand instructions (e.g., for string manipulation) requires additional TPR's (one for each additional operand). The contents of the 42-bit TPR are outlined below.



TPR.TRR (3 bits) Is the temporary ring register and is used to maintain the lowest level of privilege (i.e., highest ring number) encountered during operand address calculation. The TRR is initialized with the value of the PRR field of the PPR at the beginning of each instruction. During the operand address calculation, TRR is used to record the highest value of SDW.R1 (the top of the read/write bracket) of any segment used in the address calculation. For example, if an indirect address is fetched from segment X, TRR is set to the larger of TRR (its current value) and the R1 field in the SDW for segment X. Note that during the operand address calculation, the value of TRR may get larger but never smaller.

TPR.TSR (15 bits) Is the temporary segment register (same as the TBR in the 645) and is initialized with the value of the PSR field of the PPR at the beginning of each instruction. During operand address calculation, TSR contains the segment number portion of the current address calculation.

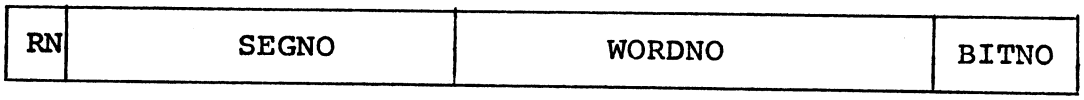
TPR.CA (18 bits) Is the computed address and serves the same function as the 645 register of the same name. The computed address is initialized at the beginning of each instruction with the contents of the instruction counter of the PPR. During operand address calculation, CA contains the word number portion of the current address calculation.

TPR.BITNO (6 bits) Is a bit offset relative to the first bit in the word specified by CA.

Once an operand address calculation is complete, the value of TRR is compared with the ring brackets of the segment containing the operand address to determine if the operation is to be allowed. For example, if the instruction intends to store into this operand, the value of TRR must be less than or equal to the R1 (in the SDW) of the segment to be modified.

4.3 The Eight Pointer Registers

The H6180 processor contains eight program-accessible pointer registers (PR's) which replace the 8 address base registers (ABR's) of the 645. The PR's of the 6180 processor differ from the 645 base registers in that each PR contains both a segment number and a word number portion. In effect, each PR of the 6180 processor behaves as a 645 base register pair. The contents of each 42-bit PR is outlined below.



RN (3 bits) Is used by the software to specify the level of privilege (i.e., ring number) at which the processor is to treat the address contained in the address register. When the processor uses the contents of a PR for address modification (e.g., bit 29 in the instruction word is on) the value of TRR is set to the larger of TRR (its current value) and the RN field of the specified PR. The use of the RN field of a PR allows the software to save the TRR of an operand address calculation (e.g., through the use of an EPP (effective pointer to pointer, formerly EAP) instruction.

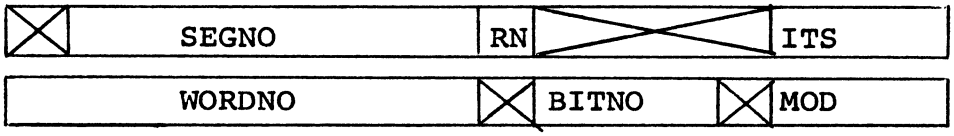
SEGNO (15 bits) Specifies the segment number portion of the segmented address.

WORDNO (18 bits) Specifies the word number portion of the segmented address.

BITNO (6 bits) Specifies a bit offset relative to WORDNO.

4.4 ITS Pointers

The software may store the contents of a PR into an ITS (indirect to segment) word pair with the use of an SPRI (store pointer in ITS format, formerly STP) instruction. The software may then address indirectly through the ITS indirect word rather than using the original PR. Alternatively, the software may reload another PR from the ITS word pair through the use of the EPP instruction. In either case, it is necessary to save the value of the RN of the PR in the ITS word pair so that the privilege level of the original operand address calculation is not lost. As a result, the ITS word pair is modified to include a ring number field as outlined below.



SEGNO (3-17) Is the segment number field (as in the 645). Note that bits 0-2 of the ITS word pair are set to zero for compatibility with 645 programs expecting an 18-bit segment number in the upper half of the first word.

RN (18-20) Is set to the value of the RN field of the PR during the SPRI instruction. If the processor attempts to indirect through an ITS word pair, TRR is set to the larger of TRR, RN (of the ITS), and R1 of the segment containing the ITS. Note that an improper value of RN in an ITS word pair has no ill effect since the processor always takes the maximum of TRR and RN. In other words, it is impossible for an ITS word pair to specify a higher privilege than the segment in which it resides.

ITS (30-35) Specifies the modifier code (octal 43) for the ITS modifier (same as in the 645).

WORDNO (0-17) Is the word number portion of the saved PR (same as in the 645).

BITNO (21-26) Is the bit offset of the PR saved by the SPRI instruction.

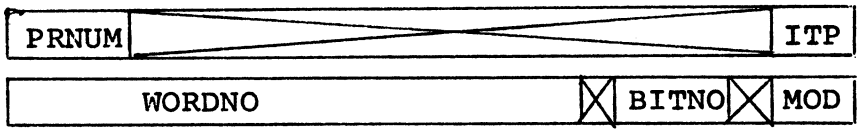
MOD (30-35) Is set to zero by the SPRI instruction but may be set by the software to specify further address modification (same as in the 645).

Since most Multics compilers (notably PL/I) calculate addresses via an EPP instruction, compiler-generated code can take full advantage of the hardware protection mechanism with little or no modification. If all addresses of all input parameters are calculated and saved (for use as outgoing argument pointers) via the use of the EPP and SPRI instructions it will be possible for a procedure operating in ring 1 to pass to ring 0 a parameter given to the procedure from ring 2 without checking the address of the parameter. The access checking is fully automatic as long as the TRR of the original address calculation continues to be maintained and passed along as the RN field of a PR or ITS word pair.

The STCD (store control double) instruction has been modified to store the PRR in the same manner as SPRI stores the RN field of a PR. The PRR is stored by the STCD to allow an RTCD (return control double) instruction to return to the proper ring.

4.5 ITP Pointers

To accomodate the possibility of a bit offset, the indirect word for ITP modification (called ITB on the 645) has been modified as follows.



PRNUM (0-2) Is the number of the PR to be used in the indirection.

ITP (30-35) Is the "Indirect Through Pointer" modifier (octal 41, the same as for ITB in the 645).

WORDNO (0-17) Is added to the WORDNO field of the specified PR during indirection to form a new computed address.

BITNO (21-26) Is the bit offset which overrides the BITNO field of the specified PR during indirection.

MOD (30-35) May specify further address modification.

When an ITP modifier is found in the (even) word of an indirect reference, a new effective address is computed in the TPR as follows:

```
C(PRn.WORDNO) + C(ITP.WORDNO) + C(R) => TPR.CA
C(PRn.SEGNO) => C(TPR.TSR)
C(ITP.BITNO) => C(TPR.BITNO)
MAX(PRn.RN,SDW.R1,TPR.TRR) => C(TPR.TRR)
```

where PRn is ITP.PRNUM. Further indirection is possible if ITP.MOD so indicates. C(R) is specified by a previous IR tag.

5. The CALL6 Instruction

The new CALL6 instruction is provided as the only means by which a procedure segment may call a procedure in an inner ring (i.e., set PRR to a lower value). The CALL6 instruction is to be used in all standard inter-procedure calls and is intended to replace the transfer instruction as the last instruction of the standard Multics calling sequence.

The "6" of the CALL6 instruction specifies that the pair of pointer registers (6-7) is to be involved in the execution of the CALL6. (This use of a pair of PR's involves two full PR's (RN, SEGNO, WORDNO, and BITNO) and should not be confused with a 645 base register pair.) When the CALL6 instruction is used to transfer control to another ring, the segment number of the stack for the target ring is formed by concatenating DSBR.STACK to the left of the ring number of the target ring. The CALL6 instruction behaves as a TRA (transfer) instruction with the following exceptions.

5.1 The access checking for a CALL6 instruction allows PRR to be set to a lower value provided that the call is made from a ring within the call bracket of the target segment.

5.2 If an attempt is made to call a procedure in an outer ring (a relatively rare case) an access violation occurs. Due to the necessity of copying all arguments the standard call, save and return sequences cannot handle calls to outer rings without excessive software overhead. Therefore, calls to outer rings' procedures will continue to cause a fault to allow the system software to interpret the call.

5.3 At the beginning of the CALL6, the contents of PR6 are assumed to point to a location (i.e., the current stack frame) within the stack segment of the calling ring. During execution of the CALL6, the processor sets the contents of PR7 to point to word 0 of the stack segment of the target ring in one of two ways. (TPR.TRR contains the target ring number.)

5.3.1 If the call is to a procedure in the same ring PR7 is set to point to word zero of the current stack segment as follows.

```
PR6.SEGNO => PR7.SEGNO
00...00 => PR7.WORDNO, PR7.BITNO
TPR.TRR => PR7.RN
```

5.3.2 If the call is to an inner ring (TPR.TRR<PPR.PRR) PR7 is set to point to word 0 of the stack segment of the target ring as follows.

```
DSBR.STACK||TPR.TRR => PR7.SEGNO
00...00 => PR7.WORDNO, PR7.BITNO
TPR.TRR => PR7.RN
```

5.3.3 If an attempt is made to use CALL6 to transfer to an outer ring (TPR.TRR>PPR.PRR) an access violation fault will be generated. The only way control can be transferred to an outer ring is via the RTCD (Return Control Double) instruction (i.e., the hardware only permits inward calls and outward returns). Calls to outer rings can of course be hand coded to look like outward returns to the hardware.

The H6180 INSTRUCTION SET

The instruction set of the H6180 processor is basically a merger of the 645 and H6080 instruction sets with several new instructions added for dealing with the extended pointer (address) registers of the H6180 processor. The definition of the H6180 instructions set is constrained to meet two distinctly different objectives. The first objective is to provide H6080 object code compatibility sufficient to allow the running of complete H6080 operating systems (e.g., GCOS) on a stand-alone basis and to allow the efficient operation of H6080 slave programs under Multics. The second objective is to maintain sufficient compatibility with H645 object code to allow Multics, with minimal software changes, to run effectively on both the current H645 and the H6180 processors.

1. Compatibility of Pointer Registers with 645 Address Base Registers

The H6180 processor contains eight pointer registers (PRs) in place of the eight address base registers (ABRs) of the H645. By Multics convention, the ABRs of the 645 are always used as four internal/external ABR pairs. The control fields are set so that ABR0/ABR1, ABR2/ABR3, ABR4/ABR5, and ABR6/ABR7 function as "pointer registers," each capable of holding a complete word number/segment number address. By convention, the even-numbered ABR's are used as the internal ABR of the pair. Each PR on the H6180 processor behaves like a 645 ABR pair, and can hold a complete segmented address. This doubling of the number of pointer registers makes code generation by the language translators more straightforward, and makes it possible to produce object programs that are more efficient. It means, however, that a careful mapping of 645 ABR-manipulating order codes to the H6180 processor was required to maintain upward compatibility of object programs.

Two aspects of the compatibility problem must be considered: the use of these registers in the formation of instruction operand addresses, and the effect of instructions which manipulate them.

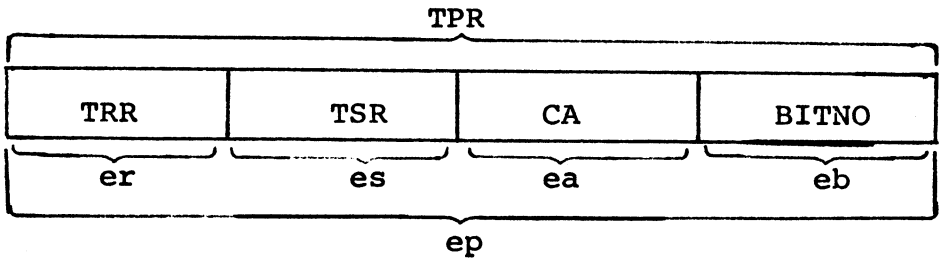
ABR-relative addressing is specified on the H645 by setting bit 29 of an instruction word to "1" and placing an ABR number in the segment tag field of the instruction (bits 0-2). The operand address calculation depends on whether the segment tag names the internal or the external ABR in a pair. If it names the internal ABR in some pair (i.e., 0, 2, 4, or 6), then both the word number from that ABR and the segment number from the paired external ABR participate in the formation of the operand address for the instruction. The effect is addressing relative to the complete address in the "pointer register" formed by the pair. If the segment tag names the external ABR in some pair (i.e., 1, 3, 5, or 7), then only the segment number from that external ABR is used and a word number of zero is assumed. The effect is addressing relative to the beginning of the segment since the word number portion in the internal ABR is ignored.

These two uses of ABR-pair "pointer registers" in operand address formation are possible because two ABR numbers are associated with each of the four "pointer registers" formed by pairs. Because there is not enough room in an instruction word to increase the size of the segment tag from three to four bits, only one number can be associated with each PR of the H6180 processor. Thus, addressing relative to the beginning of a segment through a PR which points to an arbitrary word within that segment is no longer possible. To address relative to the beginning of a segment on the H6180 processor a PR must first be explicitly loaded with a pointer to it, i.e., with the complete address of word zero of the segment.

To maintain upward compatibility of object programs, it was necessary to be able to construct object code that will do addressing relative to the beginning of a segment through a "pointer register" which will execute properly on both the H645 and the H6180 processor. The mapping of 645 ABR-manipulating order codes to the H6180 processor defined below shows how this goal was met with code that executes efficiently on both machines.

The second aspect of the compatibility problem was the effect of instructions which manipulate ABR-pair "pointer registers" on the 645. These instructions may also be applied to either the internal or the external ABR in a pair. Generally, the effect of the instruction on the "pointer register" formed by the pair is different in the two cases. For example, the "Add to Address Base Register n" (ADBn) instruction has the different effects of adding to the word number and adding to the segment number in the "pointer register" formed by the pair, depending on whether the internal or external ABR of the pair is specified as "n", respectively. The other 645 instructions with this property are EABn, EAPn, LBRn, SBRn, STPn, and TSBn. In cases where both functions performed by one of these instructions are used by Multics, instructions are provided on the H6180 processor to perform both on PRs. In several cases, however, both functions are not used, and only one or the other is provided on the H6180 processor. The same order code is used to represent the same function on both machines in all cases.

Before proceeding to a description of the proposed order code mapping, it is necessary to define some notation. The term "effective address" is used with respect to the H645 to mean the word number from the complete address of the ultimate operand of an instruction. Because Multics users think in terms of complete addresses, however, there is a tendency for them to use this term to mean the complete address of the operand, rather than just the word number from the complete address. The "Effective Address to Pair n" (EAPn) instruction suggests this usage is also acceptable. To eliminate user confusion on this matter, the following definitions are made: the contents of the temporary pointer register (TPR) at the conclusion of an operand address formation is called the effective pointer (ep). The ring number (TRR), segment number (TSR), word number (CA), and bit number in the TRR are called the effective ring (er), effective segment (es), effective address (ea), and effective bit (eb), respectively. The following diagram summarizes this notation.



The mapping of order codes for manipulating pointer registers can now be considered. The PRs of the H6180 processor are numbered PR0 through PR7, and the order code mapping will associate PR0, PR2, PR4, and PR6 with the ABR0/ABR1, ABR2/ABR3, ABR4/ABR5, and ABR6/ABR7 "pointer registers" of the 645, respectively. The 645 instruction set includes 59 order codes for manipulating paired and unpaired ABRs. These are grouped as 10 mnemonic instructions. They are mapped to 82 order codes grouped as 12 instructions on the H6180 processor. The ten 645 instructions are considered in alphabetic order.

ADBn - Add to Address Base Register n

Summary: $C(Y)_{0-17} + C(ABRn)_{0-17} \Rightarrow C(ABRn)_{0-17}$

If n is even, then this instruction adds to the word number in the internal ABR of a pair. If n is odd, then it adds to the segment number in the external ABR of a pair. The first function is used by Multics and is duplicated on the H6180 processor. The second function is meaningless with respect to Multics and is not duplicated. The four order codes thus freed are reused to provide the add-to-word-number function for the four additional PRs on the H6180 processor. The result is the H6180 processor instruction:

ADWPn - Add to Word Number Field of PRn (old ADBn order codes)

$C(Y)_{0-17}$ + C(PRn.WORDNO) => C(PRn.WORDNO)
00...0 => C(PRn.BITNO)

EABn - Effective Address to Base n

Summary: EA => $C(ABRn)_{0-17}$

If n is even, then this instruction replaces the word number in the internal ABR of a pair with the effective address (word number from the complete address) of the operand of the instruction. If n is odd, then it replaces the segment number in the external ABR of a pair with the effective address. Both functions are used by Multics. They provide the best way of loading a pair with a complete address represented as a pair of integers. For example, if a segment number is contained in word "temp" as an integer and a word number is contained in word "temp+1" as an integer, then the sequence:

```
ldaq    temp      place the integers in the A and Q
eab0    0,ql      put second integer in internal ABR of pair 0/1
eab1    0,al      put first integer in external ABR of pair 0/1
```

loads the ABR0/ABR1 "pointer register" with the complete address represented by the segment number and the word number. Both functions of the EABn instruction are provided on the 6180 processor. The order codes for EAB0, EAB2, EAB4, and EAB6 become EAWP0, EAWP2, EAWP4 and EAWP6, respectively. This instruction, "Effective Address to Word Number Field of PRn", is described below. New order codes are provided for EAWP1, EAWP3, EAWP5, and EAWP7. The order codes for EAB1, EAB3, EAB5, and EAB7, become EASP0, EASP2, EASP4, and EASP6, respectively. This instruction, "Effective Address to Segment Number Field of PRn", is described below. New order codes are provided for EASP1, EASP3, EASP5, and EASP7. The mapping of, say, EAB1 to EASP0 is correct, for EAB1 updates the segment number portion of the ABR0/ABR1 "pointer register" and EASP0 does the same to the corresponding PR (which is PR0) of the H6180 processor. With this mapping, the object form of the instruction sequence given in the example above performs the same function on both machines. A summary of these two H6180 processor instruction is:

EASPn - Effective Address to Segment Number Field of PRn (EAB1, 3, 5, 7 and 4 new codes)

EA => C(PRn.SEGNO)

EAWPn - Effective Address to Word Number Field of PRn (EAB0, 2, 4, 6 and 4 new codes)

EA => C(PRn.WORDNO)

EB => C(PRn.BITNO)

EAPn - Effective Address to Pair n

Summary: If $C(ABRn)_{21}$ = "1" then ES => $C(ABRn)_{0-17}$

If $C(ABRn)_{21}$ = "0" then EA => $C(ABRn)_{0-17}$ and
ES => $C(ABRm)_{0-17}$

where "m" is the external ABR associated with the internal ABRn

If n is even, then $C(ABRn)_{21}$ = "0" will be true and this instruction replaces the word number/segment number in the ABRn/ABRn+1 "pointer register" with the word number and segment number from the complete address of the operand of the instruction. If n is odd, then $C(ABRn)_{21}$ = "1" will be true and only the segment number from the complete operand address replaces the segment number in the specified external ABR of a pair. The first function of the EAPn instruction is frequently used in Multics object programs, and is provided on the H6180 processor. The second function is infrequently used and is not duplicated on the follow-on processor. Instead, the order codes for EAP1, EAP3, EAP5, and EAP7 are used to solve the compatibility problem caused by lack of room in which to expand the segment tag field of an instruction from three bits to four bits to match the increased number of pointer registers (see above in the introduc- to this section).

Taking these ideas one at a time, the order codes for EAP0, EAP2, EAP4, and EAP6 become EPP0, EPP2, EPP4, and EPP6, respectively, on the H6180 processor. This instruction, "Effective Pointer to PRn", is described below. New order codes are provided for EPP1, EPP3, EPP5, and EPP7.

Next, before using the order codes of EAP1, EAP3, EAP5, and EAP7 to provide a different function on the H6180 processor, it must be shown that the function provided by them can be duplicated with other instructions. Consider an example of the occurrence of the instruction:

```
eap3    <operand>    replace segment number in the 2/3 pointer
                     register with the segment number of
                     operand
```

This instruction can be replaced with three instruction sequence:

```
eaa     ABR2|0       word number from 2/3 to au
eap2    <operand>    complete address of operand to the 2/3
                       pointer register
eab2    0,au         replace word number with number saved in
                       au
```

The same sort of thing could be done using one of the index registers rather than the accumulator. Because the EAPn instruction is very seldom used with n specified as odd, the inefficiency of replacing one instruction with three is of no concern. The order code for EAPn with n odd is used to provide a different function on the H6180 processor.

The different function is one that is not provided by a single instruction on the H645. The new instruction is EPBPn: Effective Pointer at Segment Base to PRn. The ring number and segment number from the effective pointer at the instruction's operand are placed in the RINGNO and SEGNO fields of PRn and the WORDNO and BITNO fields are set to zero. This generates in PRn a pointer at the beginning of the segment containing the operand of the instruction. The order codes for EAP1, EAP3, EAP5, and EAP7 are mapped to EPBP1, EPBP3, EPBP5, and EPBP7, respectively. New order codes are provided for EPBP0, EPBP2, EPBP4, and EPBP6. Note that the mapping here is different than for the case of EABn for n odd. EAB1 mapped to EASP0, but EAP1 maps to EPBP1. The reason for this is so that the EPBPn instruction can solve the compatibility problem. As has been described, code sequences of the form (where n is even):

```
eapn      <operand>      pointer at operand to pair n/n+1
 .
 .
 .
<opcode>  ABRn+1|m,<modifier>      reference relative to
                                   the beginning of the
                                   segment pointed into by
                                   ABRn/ABRn+1
```

will no longer work on the H6180 processor. (The object form of the second instruction in the sequence will generate a reference relative to PRn+1 and the first instruction in the sequence loaded PRn.) This sequence can be made to work on both machines by inserting a third instruction. The modified 645 instruction sequence is:

```
eapn      <operand>
 .
 .
 .
eapn+1    ABRn|0         effectively does nothing on the 645
 .
 .
 .
<opcode>  ABRn+1|m,<modifier>
```

On the H6180 processor the object form of the above sequence is the same as that generated by the following sequence of the H6180 processor instructions:

```
eppn      <operand>     pointer at operand to PRn
 .
 .
 .
epbpn+1   PRn|0         pointer at beginning of operand seg-
 .                      ment to PRn+1
 .
 .
<opcode> PRn+1|m,<modifier>  addressing relative to beginning
                              of segment
```

The effect of the sequence is the same on both machines and although the mnemonics are different, the order codes used are, in fact, the same. Thus, the strange mapping of the EAP1, EAP3, EAP5, and EAP7 instructions solves this problem of compatibility between the 645 and the H6180 processor.

The two instructions for the H6180 processor generated from EAPn are summarized as:

EPBPn - Effective Pointer at Segment Base to PRn (EAP1, 3, 5, 7, and 4 new codes)

er => C(PRn.RINGNO)
es => C(PRn.SEGNO)
00...0 => C(PRn.WORDNO), C(PRn.BITNO)

EPPn - Effective Pointer to PRn (EAP0, 2, 4, 6 and 4 new codes)

ep => C(PRn)

LBRn - Load Base Register n

Summary: $C(Y)_{0-23}$ => C(ABRn)

This instruction manipulates single ABRs, and ignores the pairing specified by the control fields. It is infrequently used and is not duplicated on the H6180 processor. The order codes thus freed are used for a new function on the H6180 processor: loading a pointer register from a segment number and word number stored in a single word as a "packed pair". This new instruction is summarized below:

LPRPn - Load Pointer Register from Packed Pointer into PRn (old codes for LBRn)

er => C(PRn.RINGNO)
$C(Y)_{6-17}$ => C(PRn.SEGNO)
$C(Y)_{18-35}$ => C(PRn.WORDNO)
$C(Y)_{0-5}$ => C(PRn.BITNO)

LDB - Load Bases

Summary: $C(Y,Y+1,...,Y+7)_{0-23}$ => C(ABR0,ABR1,...,ABR7)

The analog of this instruction on the H6180 processor loads the eight PRs from eight consecutive ITS pairs. The decision to load from ITS pairs is made because each pointer register is longer than 36 bits and thus requires two words of storage in any case, and storing and loading the PRs from ITS pairs allows the stored images to be used as indirect addresses. The new instruction has the following summary:

LPRI - Load Pointers from ITS Pointers into Pointer Registers (old LDB)

"lpri loc" is equivalent to the following sequence assuming "loc" is the address (zero mod 16) of a vector of eight ITS pairs:

```
epp0    loc,*
epp1    loc+2,*
 .
 .
 .
epp7    loc+14,*
```

LDCF - Load Control Field

This instruction has no use with respect to PRs, and is dropped. The order code is freed for other use.

SBRn - Store Address Base Register n

This instruction is the inverse of the 645 LBRn instruction, and is mapped to the inverse of the H6180 processor LPRPn instruction.

SPRPn - Store PRn in Packed Pointer format (old codes for SBRn)

$C(PRn.BITNO) \Rightarrow C(Y)_{0-5}$

$C(PRn.SEGNO) \Rightarrow C(Y)_{6-17}$

$C(PRn.WORDNO) \Rightarrow C(Y)_{18-35}$

STB - Store Bases

This instruction is the inverse of the 645 LDB instruction and is mapped to the inverse of the H6180 processor LPRI instruction.

SPRI - Store Pointers Register in ITS format (old STB)

"spri loc" is equivalent to the following sequence assuming "loc" is a zero mod 16 address:

```
spri0     loc
spri1     loc+2
 .
 .
 .
spri7     loc+14
```

STPn - Store Pair n

Summary:

If $C(ABRn)_{21}$ = "0" then

$C(ABRm)_{0-17}$ => $C(Y)_{0-17}$, 00...0 => $C(Y)_{18-29}$,
ITS tag (100011) => $C(Y)_{30-35}$

$C(ABRn)_{0-17}$ => $C(Y+1)_{0-17}$, 00...0 => $C(Y+1)_{18-35}$

If $C(ABRn)_{21}$ = "1" then

$C(ABRn)_{0-17}$ => $C(Y)_{0-17}$, 00...0 => $C(Y)_{18-29}$,
ITS tag (100011) => $C(Y)_{30-35}$

00...0 => C(Y+1)

where ABRm is the external base paired with ABRn and Y is a zero mod 2 address.

This instruction is another of those with two functions, depending whether the internal or external ABR of pair is specified by "n". If n is even, then $C(ABRn)_{21}$ = "0" will be true and the instruction causes the contents of the ABR pair "pointer register" including this ABR to be stored as an ITS pair. The ITS pair will contain the complete address from the pair. On the other hand, if n is odd then $C(ABRn)_{21}$ = "1" will be true and an ITS pair is produced that contains only the segment number from the specified external ABR in a pair and a word number of zero. Both functions are provided on the H6180 processor. The order codes for STP0, STP2, STP4, and STP6 become SPRI0, SPRI2, SPRI4, and SPRI6 on the H6180 processor, respectively. This instruction, "Store Pointer from PRn" in ITS format, is described below. New order codes are provided for SPRI1, SPRI3, SPRI5, and SPRI7. The order codes for STP1, STP3, STP5, and STP7 become SPBP0, SPBP2, SPBP4, and SPBP6 on the H6180 processor. This instruction, "Store Pointer at Segment Base from PRn", is described below. New order codes are provided for SPBP1, SPBP3, SPBP5, and SPBP7. The mapping here is similar to the mapping described earlier for the 645 EABn instruction.

SPRIn - Store Pointer from PRn in ITS format (STP0, 2, 4, 6 and 4 new codes)

000 => $C(Y)_{0-2}$
PRn.SEGNO => $C(Y)_{3-17}$
PRn.RN => $C(Y)_{18-20}$
00...0 => $C(Y)_{21-29}$
ITS flag (100011) => $C(Y)_{30-35}$
PRn.WORDNO => $C(Y+1)_{0-17}$
00...0 => $C(Y+1)_{18-20}$
PRn.BITNO => $C(Y+1)_{21-26}$
00...0 => $C(Y+1)_{27-35}$

SPBPn - Store Pointer at Segment Base from PRn (STP1, 3, 5, 7, and 4 new codes). Same as SPRIn except:

00...0 => C(Y+1)

TSBn - Transfer and Set Base n

Summary: If $C(ABRn)_{21}$ = "0" then C(IC) + 00...01 => $C(ABRn)_{0-17}$
C(PBR) => $C(ABRm)_{0-17}$, EA => C(IC),ES => C(PBR)

where n and m are the designated internal and linked external ABR's, respectively

IF $C(ABRn)_{21}$ = "1" then C(PBR) => $C(ABRn)_{0-17}$,
EA => C(IC),ES => C(PBR)

Of the two functions performed by this instruction, only that associated with n even is useful under Multics. The function performed when n is odd is not duplicated on the H6180 processor. The order codes thus freed are used to provide the first function for the additional four PRs. Thus, the following instruction is defined:

TSPn - Transfer and Set PRn (old TSBn order codes)

C(PRR) => C(PRn.RINGNO)
C(PSR) => C(PRn.SEGNO)
C(IC) + 1 => C(PRn.WORDNO)
000000 => C(PRn.BITNO)
es => C(PSR)
ea => C(IC)

2. Other Modifications to the H645 Instruction Set

With the exception of the instructions discussed earlier and those listed below, all H645 and H6080 instructions are interpreted by the H6180 as specified in their respective processor manuals. The instructions listed below have been modified from their H645 versions. Detailed instruction writeups of all modified unprivileged instructions are provided in section 4.

LBAR - Load Base Address Register

This instruction causes a "635 Compatibility" fault on the H645. On the H6180 the BAR is loaded with the specified values. LBAR is an unprivileged instruction when the H6180 is in Multics mode but is privileged when the H6180 is in GCOS (H6080) mode.

RCCL - Read Calendar Clock

The manner in which the processor port number is specified by the RCCL is greatly simplified.

RET, STC1, STI, and LDI Instructions

These instructions have all been modified to deal with the expanded number of indicators which are maintained in the H6180.

RTCD and STCD Instructions

These instructions no longer save and reload the indicators on the H6180. (Indicators are no longer saved accross a call by the standard CALL, SAVE, and RETURN macros.) In addition RTCD is modified to handle a return to an outer ring.

STAC - Store A Conditionally

This instruction is implemented using the limited read-alter-rewrite (RAR) facility of the H6180. On the H645 an RAR instruction would delay any instruction until it completed its RAR cycle. On the H6180 an RAR instruction will only delay other RAR instructions but have no control over non-RAR instructions.

TSS - Transfer and Set Slave

The TSS instruction allows an unprivileged user program to enter Multics BAR mode and transfer to H600/6000 slave programs to be run in BAR mode within the Multics environment.

3. New Instructions

In addition to the EIS instructions the following new unprivileged instructions have been added to the H6180.

EPAQ - Effective Pointer to AQ

The EPAQ instruction loads the full effective pointer (TRR, TSR, WORDNO, and bit number) into the AQ in a form such that each value is available for subsequent AU, AL, QU, or QL indexing operations.

STACQ - Store A Conditional on Q

The STACQ instruction is designed as an eventual replacement for STAC as a locking instruction. If the contents of the operand word are equal to the contents of the Q register, the A register is stored in the operand word and the zero indicator is set on. Otherwise the operand is not modified and the zero indicator is set off.

4. Writeups of New and Modified Instructions

This section gives detailed instruction write-ups of all new or modified instructions with the exception of the EIS instructions which are documented in standard Honeywell 6000 line publications. Note that the H6000 line operation code field now includes bit 27 giving a 10-bit opcode field (600 line opcodes were restricted to 9 bits, 18-26). In the instruction writeups the opcode is given as XXX (Y) where XXX is the octal value of the first 9 bits of the field and Y is 0 or 1 depending on the value of bit 27 in the opcode. Thus the opcode for the H635 instruction LDA (H635 opcode 235) is written as 235 (0).

4.1 ADWPn - Add to Word Register of PRn

Mnemonic	Name of Instruction	Op Code (octal)
ADWPn	Add to Word Register of PRn	050 (0) 051 (0) 052 (0) 053 (0) 150 (0) 151 (0) 152 (0) 153 (0)

Summary: $C(Y)_{0-17}$ + C(PRn.WORDNO) => C(PRn.WORDNO)
00 ... 00 => C(PRn.BITNO)

The contents of memory location Y is added to the contents of the WORDNO field of Pointer Register n and the result replaces the contents of the WORDNO field. Zeros replace the contents of the BIT field of Pointer Register n. The rest of Pointer Register n remains unchanged.

Modifications: All except CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: CI, SC, SCR

4.2 CALL6 - Call Instruction

Mnemonic	Name of Instruction	Op Code (octal)
CALL6	Call (using PR6-7)	713 (0)

Summary: C(TPR) => C(PPR)

where: if C(TPR.TRR) < C(PPR.PRR),
C(DSBR.STACK) || C(TPR.TRR) => C(PR7.SEGNO)
if C(TPR.TRR) = C(PPR.PRR), C(PR6.SEGNO) => C(PR7.SEGNO)
00 ... 00 => C(PR7.WORDNO)
00 ... 00 => C(PR7.BITNO)
C(TPR.TRR) => C(PPR.PRR)
C(TPR.TSR) => C(PPR.PSR)
C(TPR.CA) => C(PPR.IC)
C(TPR.TRR) => C(PR7.RN)

The SEGNO field of PR7 is replaced according to the TRR and PRR fields (see above) and the effective ring number, segment number and segment address replace the corresponding fields of the procedure pointer register.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

Note: If C(TRR) > C(PRR) an access violation fault is initiated and the instruction is not executed.

4.3 EASPn - Effective Address to Segment Number Field of PRn

Mnemonic	Name of Instruction	Op Code (octal)
EASPn	Effective Address to Segment Number Field of PRn	311 (0) 310 (1) 313 (0) 312 (1) 331 (0) 330 (1) 333 (0) 332 (1)

Summary: EA_{3-17} => C(PRn.SEGNO)

The least significant 15 bits of the effective address register replaces the contents of the SEGNO field of Pointer Register n. The rest of Pointer Register n remains unchanged.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

Note: This and EAWPn replace the 645 Effective Address to Base n instruction.

4.4 EAWPn - Effective Address to Word/Bit Number Field of PRn

Mnemonic	Name of Instruction	Op Code (octal)
EAWPn	Effective Address to Word/Bit Number Field of PRn	310 (0) 311 (1) 312 (0) 313 (1) 330 (0) 331 (1) 332 (0) 333 (1)

Summary: C(TPR.CA) => C(PRn.WORDNO)
C(TPR.BITNO) => C(PRn.BITNO)

The calculated address and bit number replaces the contents of the word and bit number fields of Pointer Register n. The remaining fields of Pointer Register n remain unchanged.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

Note: This and EASPn replace the 645 Effective Address to Base n instruction.

4.5 EPAQ - Effective Pointer to AQ Register

Mnemonic	Name of Instruction	Op Code (octal)
EPAQ	Effective Pointer to AQ Register	212 (0)

Summary: $C(TPR) \Rightarrow C(AQ)_{00-17}$

where:

$00 \ldots 00 \Rightarrow C(AQ)_{0-2}$
$C(TPR.TSR) \Rightarrow C(AQ)_{3-17}$
$00 \ldots 00 \Rightarrow C(AQ)_{18-32}$
$C(TPR.TRR) \Rightarrow C(AQ)_{33-35}$
$C(TPR.CA) \Rightarrow C(AQ)_{36-53}$
$00 \ldots 00 \Rightarrow C(AQ)_{54-65}$
$C(TPR.BITNO) \Rightarrow C(AQ)_{66-71}$

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected:

Zero	If C(AQ) = 0, then ON; otherwise OFF

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

4.6 EPBPn - Effective Pointer at Base to PRn

Mnemonic	Name of Instruction	Op Code (octal)
EPBPn	Effective Pointer at Base to PRn	350 (1) 351 (0) 352 (1) 353 (0) 370 (1) 371 (0) 372 (1) 373 (0)

Summary: C(TPR.TRR) => C(PRn.RN)
C(TPR.TSR) => C(PRn.SEGNO)
00 ... 00 => C(PRn.WORDNO)
00 ... 00 => C(PRn.BITNO)

The ring number and segment number portion of the Effective Pointer (TPR) replace the contents of the RN and SEGNO fields of Pointer Register n. The contents of the WORDNO and BITNO fields of Pointer Register n are forced to zero.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

Note: This and EPPn replace the 645 Effective Address to Pair n instruction.

4.7 EPPn - Effective Pointer to PRn

Mnemonic	Name of Instruction	Op Code (octal)
EPPn	Effective Pointer to PRn	350 (0) 351 (1) 352 (0) 353 (1) 370 (0) 371 (1) 372 (0) 373 (1)

Summary: C(TPR) => C(PRn)

C(TPR.TRR) => C(PRn.RN)
C(TPR.TSR) => C(PRn.SEGNO)
C(TPR.CA) => C(PRn.WORDNO)
C(TPR.BITNO) => C(PRn.BITNO)

The ring number, segment number, bit number, and computed address of the Effective Pointer (TPR) replace the contents of the RN, SEGNO, WORDNO and BITNO fields of Pointer Register n.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

Note: This and EPBPn replace the 645 Effective Address to Pair n instruction.

4.8 LBAR - Load Base Address Register

Mnemonic	Name of Instruction	Op Code (octal)
LBAR	Load Base Address Register	230 (0)

Summary: $C(Y)_{0-8}$ => BAR.BASE
$C(Y)_{9-17}$ => BAR.BOUND

BAR.BASE is the high order 9 bits of an 18-bit relocation constant (with 9 low order zeros) used to relocate effective address calculations when the processor is placed in BAR mode. BAR.BOUND specifies the number of 512-word logical blocks within the simulated "core image." A reference to a logical block above this bound will cause a store fault. The LBAR instruction is unprivileged when executed while the processor is in Multics mode.

Modifications: all except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

4.9 LDI - Load Indicator Register

Mnemonic	Name of Instruction	Op Code (octal)
LDI	Load Indicator Register	634 (0)

Summary: $C(Y)_{18-30} = C(IR)$

The relationship between $C(Y)_{18-30}$ and the indicators is as follows:

Bit Position C(Y)	Indicator
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
29	Truncation
30	Multi-Word Inst. Int.

Modifications: All except CI, SC, SCR
The tally runout indicator will reflect $C(Y)_{25}$ regardless of what address modification is performed on the LDI instruction for tally operations.

Indicators Affected:

Parity Mask	If corresponding bit in C(Y) is 1, and processor is in Privileged Mode, then ON; otherwise OFF. Indicator is NOT AFFECTED in the Non-Privileged Mode.
Bar Mode	Not affected
Absolute Mode	Not affected
All other Indicators	If corresponding bit in C(Y) is ONE, then ON; otherwise OFF.

Illegal Procedure Fault:
Modifications: SC, SCR, CI

4.10 LPRI - Load Pointer Registers from ITS Pairs

Mnemonic	Name of Instruction	Op Code (octal)
LPRI	Load Pointer Registers from ITS Pairs	173 (0)

Summary: $C(Y, Y + 2, \ldots, Y + 14)_{00-71}$ => C(PR0, PR1, ..., PR7)

where:

MAX(Y+2n pair 18-20, SDW.R1, TPR.TRR) => C(PRn.RN)

$C(Y + 2n\ pair)_{3-17}$ => C(PRn.SEGNO)

$C(Y + 2n\ pair)_{36-53}$ => C(PRn.WORDNO)

$C(Y + 2n\ pair)_{57-62}$ => C(PRn.BITNO)

Starting at location Y, the contents of eight word pairs (in ITS pair format) replace the contents of Pointer Registers 0 thru 7 as shown. The hardware assumes Y_{14-17}=0000 (zero modulo 16) and addressing will be incremented accordingly. No check is made. The contents of the ITS word pairs loaded is unchanged.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

Note: This replaces the 645 Load Base Register instruction.

4.11 LPRPn - Load Pointer Register n Packed

Mnemonic	Name of Instruction	Op Code (octal)
LPRPn	Load Pointer Register n Packed	76n (0)

Summary: $C(Y)_{0-35}$ => C(PRn)

where:

C(TPR.TRR) => C(PRn.RNR)
$C(Y)_{0-5}$ => C(PRn.BITNO)
X||$C(Y)_{6-17}$ => C(PRn.SEGNO)
$C(Y)_{18-35}$ => C(PRn.WORDNO)

If the $C(Y)_{6-17}$ is all ones, X will be 111111; if $C(Y)_{6-17}$ is not all ones, X will be 000000.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

4.12 RCCL - Read Calendar Clock

Mnemonic	Name of Instruction	Op Code (octal)
RCCL	Read Calendar Clock	633 (0)

Summary: 00 ... 00 => $C(AQ)_{0-19}$
$C(Calendar\ Clock) => C(AQ)_{20-71}$
$C(Y)_{0-2}$ specify which Processor Port is to be used.

The contents of the clock in one of the System Controllers replaces the contents of the AQ register as shown. The contents of the clock is unchanged. The contents of the three most significant bits of the C(Y) specify which Processor Port (i.e., which System Controller) is to be used.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

4.13 RET - Return

Mnemonic	Name of Instruction	Op Code (octal)
RET	Return	630 (0)

Summary: $C(Y)_{0-17}$ => C(IC); $C(Y)_{18-31}$ => C(IR)

The contents of the location specified by Y replace the contents of the Instruction Counter and Indicator Register. The Tally Runout indicator reflects the state of $C(Y)_{25}$ prior to address modification and is unaffected by any subsequent modification performed on the RET instruction. The relationship between $C(Y)_{18-31}$ and the indicators are as follows:

Bit Position C(Y)	Indicator
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Not (BAR Mode)
29	Truncation
30	Multi-word Inst. Int.
31	Absolute Mode

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected:

Parity Mask	Not affected
Bar Mode	Not affected
Absolute Mode	Not affected
All other Indicators	If corresponding bit in C(Y) is 1, then ON; otherwise OFF.

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

4.14 RTCD - Return Control Double

Mnemonic	Name of Instruction	Op Code (octal)
RTCD	Return Control Double	610 (0)

Summary: C(Y pair) => C(PPR)
$C(Y\ pair)_{3-17}$ => C(PPR.PSR)
$Max(C(Y)_{18-20}, C(TPR.TRR), C(SDW.R1_Y))$ => C(PRR)
$C(Y\ pair)_{36-53}$ => C(PPR.IC)

If TPR.TRR>PPR.PRR, then TPR.TRR→PR0-7.RN, otherwise no change to RN

The contents of the word pair starting at location Y replaces the contents of the procedure segment and ring number. The hardware assumes Y_{17} = 0. No check is made. The contents of the word pair starting at location Y is unchanged.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

4.14 SPBPn - Store Segment Base Pointer of PRn

Mnemonic	Name of Instruction	Op Code (octal)
SPBPn	Store Segment Base Point of PRn	250 (1) 251 (0) 252 (1) 253 (0) 650 (1) 651 (0) 652 (1) 653 (0)

Summary: 000 => C(Y pair)$_{0 - 2}$
C(PRn.SEGNO) => C(Y pair)$_{3 - 17}$
C(PRn.RN) => C(Y pair)$_{18 - 20}$
00 ... 00 => C(Y pair)$_{21 - 29}$
43_8 => C(Y pair)$_{30 - 35}$
00 ... 00 => C(Y pair)$_{36-71}$

The ring number and segment descriptor number currently in the Pointer Register n replace the contents of word pair starting at location Y as shown. The hardware assumes $Y_{17} = 0$. No check is made. The contents of Pointer Register n is unchanged.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

Note: This and SPRIn replace the 645 Store Pair n instruction.

4.15 SPRI - Store Pointer Registers as ITS Pairs

Mnemonic	Name of Instruction	Op Code (octal)
SPRI	Store Pointer Registers as ITS Pairs	254 (0)

Summary: C(PR0, PR1,...,PR7) => $C(Y, Y+2,...,Y+14)_{0 - 71}$

where:

$000 \Rightarrow C(Y+2n \text{ pair})_{0 - 2}$
$C(PRn.SEGNO) \Rightarrow C(Y+2n \text{ pair})_{3 - 17}$
$C(PRn.RN) \Rightarrow C(Y+2n \text{ pair})_{18 - 20}$
$00 \ldots 00 \Rightarrow C(Y+2n \text{ pair})_{21 - 29}$
$43_8 \Rightarrow C(Y+2n \text{ pair})_{30 - 35}$
$C(PRn.WORDNO) \Rightarrow C(Y+2n \text{ pair})_{36 - 53}$
$000 \Rightarrow C(Y+2n \text{ pair})_{54 - 56}$
$C(PRn.BITNO) \Rightarrow C(Y+2n \text{ pair})_{57 - 62}$
$00 \ldots 00 \Rightarrow C(Y+2n \text{ pair})_{63 - 71}$

Starting at location Y, the contents of Pointer Registers 0 - 7 replace the contents of eight word pairs (in ITS pair format). The hardware assumes $Y_{14 - 17} = 0000$ and addressing is incremented accordingly. No check is made. The contents of the Pointer Registers are unchanged.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

Note: This replaces the 645 Store Bases instruction.

4.16 SPRIn - Store Pointer Register n as ITS Pair

Mnemonic	Name of Instruction	Op Code (octal)
SPRIn	Store Pointer Register n as ITS Pair	250 (0) 251 (1) 252 (0) 253 (1) 650 (0) 651 (1) 652 (0) 653 (1)

Summary: C(PRn) => $C(Y)_{0 - 71}$

where:

$000 \Rightarrow C(Y\ pair)_{0 - 2}$

$C(PRn.SEGNO) \Rightarrow C(Y\ pair)_{3 - 17}$

$C(PRn.RN) \Rightarrow C(Y\ pair)_{18 - 20}$

$00 \ldots 00 \Rightarrow C(Y\ pair)_{21 - 29}$

$43_8 \Rightarrow C(Y\ pair)_{30 - 35}$

$C(PRn.WORDNO) \Rightarrow C(Y\ pair)_{36 - 53}$

$000 \Rightarrow C(Y\ pair)_{54 - 56}$

$C(PRn.BITNO) \Rightarrow C(Y\ pair)_{57 - 62}$

$00 \ldots 00 \Rightarrow C(Y\ pair)_{63 - 71}$

The contents of Pointer Register n replaces the contents of the word pair (in ITS format) starting at location Y. The hardware assumes $Y_{17} = 0$. No check is made. The contents of the Pointer Register n are unchanged.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

Note: This and SPBPn replace the 645 Store Pair n instruction.

4.17 SPRPn - Store Pointer Register n Packed

Mnemonic	Name of Instruction	Op Code (octal)
SPRPn	Store Pointer Register n Packed	54n (0)

Summary: $C(PRn) \Rightarrow C(Y)_{0-35}$

where:

$$C(PRn.BITNO) \Rightarrow C(Y)_{0-5}$$
$$C(PRn.SEGNO)_{3-14} \Rightarrow C(Y)_{6-17}$$
$$C(PRn.WORDNO) \Rightarrow C(Y)_{18-35}$$

The contents of Pointer Register n replace the contents of location Y as shown. The contents of Pointer Register n are unchanged.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

If the 3 most significant bits of C(PRn.SEGNO) $\neq$ 000 and C(PRn.SEGNO) $\neq$ to all one's (Null Pointer).

Note: This replaces the 645 Store Address Base Register n instruction.

4.18 SREG - Store Registers

Mnemonic	Name of Instruction	Op Code (octal)
SREG	Store Registers	753 (0)

Summary: C(X0,X1,...,X7,A,Q,E,TR) = C(Y,Y+1,...,Y+7)
where Y must be a 0 modulo 8 address.*

The contents of the Index (X0-X7), A, Q, E, and TR registers are stored in successive locations beginning at Y and ending at Y+7 in the following format:

$C(X0) \Rightarrow C(Y)_{0-17}$
$C(X1) \Rightarrow C(Y)_{18-35}$
$C(X2) \Rightarrow C(Y+1)_{0-17}$
$C(X3) \Rightarrow C(Y+1)_{18-35}$
$C(X4) \Rightarrow C(Y+2)_{0-17}$
$C(X5) \Rightarrow C(Y+2)_{18-35}$
$C(X6) \Rightarrow C(Y+3)_{0-17}$
$C(X7) \Rightarrow C(Y+3)_{18-35}$

$C(A) \Rightarrow C(Y+4)_{0-35}$
$C(Q) \Rightarrow C(Y+5)_{0-35}$
$C(E) \Rightarrow C(Y+6)_{0-7}$
$00 \ldots 0 \Rightarrow C(Y+6)_{18-35}$
$C(TR) \Rightarrow C(Y+7)_{0-26}$
$00 \ldots 0 \Rightarrow C(Y+7)_{27-32}$
$C(RALR) \Rightarrow C(Y+7)_{33-35}$

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

*The hardware assumes $Y_{15-17} = 000$ and addressing is incremented accordingly. No check is made.

4.19 STAC - Store A Conditional

Mnemonic	Name of Instruction	Op Code (octal)
STAC	Store A Conditional	354 (0)

Summary: Test C(Y) Then:

1) if C(Y) = 0, C(A) = C(Y), Zero indicator set ON
2) if C(Y) ≠ 0, Zero indicator set OFF

If the initial C(Y) is non-zero then C(Y) are not changed by this instruction.

Modifications: All types except DU, DL, CI, SC, SCR

Indicators Affected:

Zero	If initial C(Y) = 0, then ON; otherwise OFF

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

4.20 STACQ - Store A Conditional on Q

Mnemonic	Name of Instruction	Op Code (octal)
STACQ	Store A Conditional-C(Y)=C(Q)	654 (0)

Summary: Test C(Y) Then:

1) if C(Y) = C(Q), C(A) => C(Y), Zero indicator set ON
2) if C(Y) ≠ C(Q), Zero indicator set OFF

If the initial C(Y) is ≠ C(Q) then C(Y) are not changed by this instruction.

Modifications: All types except DU, DL, CI, SC, SCR

Indicators Affected:

Zero	If initial C(Y) = C(Q), then ON; otherwise OFF

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

4.21 STC1 - Store Instruction Counter Plus 1

Mnemonic	Name of Instruction	Op Code (octal)
STC1	Store Instruction Counter plus 1	554 (0)

Summary: $C(IC) + 0 \ldots 01 = C(Y)_{0-17}$
$C(IR) \Rightarrow C(Y)_{18-31}$
$00 \ldots 0 = C(Y)_{32-35}$

The contents of the Instruction Counter and the Indicator Register are stored in $C(Y)_{0-17}$ and $C(Y)_{18-31}$ respectively after modification. The $C(Y)_{25}$ reflects the state of the Tally Runout indicator prior to modification. The relationship between the $C(Y)_{18-31}$ and the indicators are as follows:

Bit Position C(Y)	Indicators
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Not (BAR Mode)
29	Truncation
30	Multi-Word Inst. Int.
31	Absolute Mode

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifiations: DU, DL, CI, SC, SCR

4.22 STCD - Store Control Double

Mnemonic	Name of Instruction	Op Code (octal)
STCD	Store Control Double	357 (0)

Summary: C(PPR) => C(Y pair)

where:

000 => $C(Y\ pair)_{0-2}$
$C(PPR.PSR) => C(Y\ pair)_{3-17}$
$C(PPR.PRR) => C(Y\ pair)_{18-20}$
00 ... 00 => $C(Y\ pair)_{21-29}$
$43_8 => C(Y\ pair)_{30-35}$
$C(PPR.IC)+2 => C(Y\ pair)_{36-53}$
00 ... 00 => $C(Y\ pair)_{54-71}$

The procedure segments, ring number and the instruction counter are stored in the word pair starting at location Y as shown. The hardware assumes $Y_{17} = 0$. No check is made.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

4.23 STI - Store Indicator Register

Mnemonic	Name of Instruction	Op Code (octal)
STI	Store Indicator Register	754 (0)

Summary: C(IR) => $C(Y)_{18-31}$

The contents of the Indicator Register are stored in $C(Y)_{18-31}$ after modification, but the $C(Y)_{25}$ reflects the state of the Tally Runout indicator prior to modification. The relationship between $C(Y)_{18-31}$ and the indicators are as follows:

Bit Position C(Y)	Indicator
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Not (BAR Mode)
29	Truncation
30	Multi-Word Inst. Int.
31	Absolute Mode

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

4.24 TSPn - Transfer and Set PRn

Mnemonic	Name of Instruction	Op Code (octal)
TSPn	Transfer and Set PRn	270 (0) 271 (0) 272 (0) 273 (0) 670 (0) 671 (0) 672 (0) 673 (0)

Summary: C(PPR) => C(PRn)

where:

C(PPR.PRR) => C(PRn.RN)
C(PPR.PSR) => C(PRn.SEGNO)
C(PPR.IC)+1 => C(PRn.WORDNO)
0 ... 00 => C(PRn.BITNO)
C(TPR.CA) => C(PPR.IC)
C(TPR.TSR) => C(PPR.PSR)

The contents of the procedure pointer register replace the contents of Pointer Register n. The effective segment address and a segment number (see above) replace the instruction counter and segment number of the procedure pointer register.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, CI, SC, SCR

Note: This replaces the 645 Transfer and Set Base n instruction.

4.25 TSS - Transfer and Set Slave

Mnemonic	Name of Instruction	Op Code (octal)
TSS	Transfer and Set Slave	715 (0)

Summary: EA => C(PPR.IC), C(TPR.TSR) => C(PPR.PSR)

The new Effective Address replaces the C(PPR.IC), and the new Pointer in TPR.TSR formed during the Appending process for the transfer address replaces C(PPR.PSR).

If this instruction is executed while in the Multics operational mode, then the Absolute indicator is turned off and the Master Mode indicator will reset (BAR mode). The BAR is then used in the final address preparation of the transfer and will be applied to all subsequent instructions until a fault or interrupt occurs.

If this instruction is executed with the Master Mode indicator reset (in BAR Mode) then it functions as a TRA instruction.

Modifications: All except DU, DL, CI, SC, SCR

Indicators Affected: None

Illegal Procedure Fault:
Modifications: DU, DL, SC, CI, SCR

APPENDIX A

Alphabetic H-6180 Instruction Listing (Includes EIS)

Instruction			
502	(1)	A4BD	Add 4-bit character displacement to AR
501	(1)	A6BD	Add 6-bit character displacement to AR
500	(1)	A9BD	Add 9-bit character displacement to AR
560	(1)	AAR0	Alphanumeric descriptor to AR0
561	(1)	AAR1	Alphanumeric descriptor to AR1
562	(1)	AAR2	Alphanumeric descriptor to AR2
563	(1)	AAR3	Alphanumeric descriptor to AR3
564	(1)	AAR4	Alphanumeric descriptor to AR4
565	(1)	AAR5	Alphanumeric descriptor to AR5
566	(1)	AAR6	Alphanumeric descriptor to AR6
567	(1)	AAR7	Alphanumeric descriptor to AR7
503	(1)	ABD	Add bit displacement to AR
202	(1)	AD2D	Add using 2 decimal operands
222	(1)	AD3D	Add using 3 decimal operands
212	(0)	ABSA	Absolute Address to Accumulator
075	(0)	ADA	Add to Accumulator
077	(0)	ADAQ	Add to A-Q
415	(0)	ADE	Add to Exponent Register
033	(0)	ADL	Add Low to A-Q
035	(0)	ADLA	Add Logical to Accumulator
037	(0)	ADLAQ	Add Logical to A-Q
036	(0)	ADLQ	Add Logical to Quotient
020	(0)	ADLX0	Add logical to Index 0
021	(0)	ADLX1	Add Logical to Index 1
022	(0)	ADLX2	Add Logical to Index 2
023	(0)	ADLX3	Add Logical to Index 3
024	(0)	ADLX4	Add Logical to Index 4
025	(0)	ADLX5	Add Logical to Index 5
026	(0)	ADLX6	Add Logical to Index 6
027	(0)	ADLX7	Add Logical to Index 7
076	(0)	ADQ	Add to Quotient Register
050	(0)	ADWP0	Add to Word Number Field of PR0
051	(0)	ADWP1	Add to Word Number Field of PR1
052	(0)	ADWP2	Add to Word Number Field of PR2
053	(0)	ADWP3	Add to Word Number Field of PR3
150	(0)	ADWP4	Add to Word Number Field of PR4
151	(0)	ADWP5	Add to Word Number Field of PR5
152	(0)	ADWP6	Add to Word Number Field of PR6
153	(0)	ADWP7	Add to Word Number Field of PR7
060	(0)	ADX0	Add to Index 0
061	(0)	ADX1	Add to Index 1
062	(0)	ADX2	Add to Index 2
063	(0)	ADX3	Add to Index 3
064	(0)	ADX4	Add to Index 4
065	(0)	ADX5	Add to Index 5
066	(0)	ADX6	Add to Index 6
067	(0)	ADX7	Add to Index 7

Instruction

775 (0)	ALR	Accumulator Left Rotate
735 (0)	ALS	Accumulator Left Shift
375 (0)	ANA	And to Accumulator
377 (0)	ANAQ	And to A-Q
376 (0)	ANQ	And to Quotient
355 (0)	ANSA	And to Storage Accumulator
356 (0)	ANSQ	And to Storage Quotient
340 (0)	ANSX0	And to Storage Index 0
341 (0)	ANSX1	And to Storage Index 1
342 (0)	ANSX2	And to Storage Index 2
343 (0)	ANSX3	And to Storage Index 3
344 (0)	ANSX4	And to Storage Index 4
345 (0)	ANSX5	And to Storage Index 5
346 (0)	ANSX6	And to Storage Index 6
347 (0)	ANSX7	And to Storage Index 7
360 (0)	ANX0	And to Index 0
361 (0)	ANX1	And to Index 1
362 (0)	ANX2	And to Index 2
363 (0)	ANX3	And to Index 3
364 (0)	ANX4	And to Index 4
365 (0)	ANX5	And to Index 5
366 (0)	ANX6	And to Index 6
367 (0)	ANX7	And to Index 7
054 (0)	AOS	Add One to Storage
540 (1)	ARA0	AR0 to Alphanumeric Descriptor
541 (1)	ARA1	AR1 to Alphanumeric Descriptor
542 (1)	ARA2	AR2 to Alphanumeric Descriptor
543 (1)	ARA3	AR3 to Alphanumeric Descriptor
544 (1)	ARA4	AR4 to Alphanumeric Descriptor
545 (1)	ARA5	AR5 to Alphanumeric Descriptor
546 (1)	ARA6	AR6 to Alphanumeric Descriptor
547 (1)	ARA7	AR7 to Alphanumeric Descriptor
771 (0)	ARL	Accumulator Right Logical
640 (1)	ARN0	AR0 to Numeric Descriptor
641 (1)	ARN1	AR1 to Numeric Descriptor
642 (1)	ARN2	AR2 to Numeric Descriptor
643 (1)	ARN3	AR3 to Numeric Descriptor
644 (1)	ARN4	AR4 to Numeric Descriptor
645 (1)	ARN5	AR5 to Numeric Descriptor
646 (1)	ARN6	AR6 to Numeric Descriptor
647 (1)	ARN7	AR7 to Numeric Descriptor
731 (0)	ARS	Accumulator Right Shift
055 (0)	ASA	Add Stored to Accumulator
056 (0)	ASQ	Add Stored to Quotient
040 (0)	ASX0	Add Stored to Index 0
041 (0)	ASX1	Add Stored to Index 1
042 (0)	ASX2	Add Stored to Index 2
043 (0)	ASX3	Add Stored to Index 3
044 (0)	ASX4	Add Stored to Index 4

Instruction			
045	(0)	ASX5	Add Stored to Index 5
046	(0)	ASX6	Add Stored to Index 6
047	(0)	ASX7	Add Stored to Index 7
071	(0)	AWCA	Add with Carry to Accumulator
072	(0)	AWCQ	Add with Carry to Quotient
507	(1)	AWD	Add Word Displacement to AR
505	(0)	BCD	Binary to Binary-Coded-Decimal
301	(1)	BTD	Binary to Decimal Convert
713	(1)	CALL6	Call
532	(1)	CAMP	Clear Associative Memory Paged
532	(0)	CAMS	Clear Associative Memory Segmented
315	(0)	CANA	Comparative And With Accumulator
317	(0)	CANAQ	Comparative And With A-Q
316	(0)	CANQ	Comparative And With Quotient
300	(0)	CANX0	Comparative And With Index 0
301	(0)	CANX1	Comparative And With Index 1
302	(0)	CANX2	Comparative And With Index 2
303	(0)	CANX3	Comparative And With Index 3
304	(0)	CANX4	Comparative And With Index 4
305	(0)	CANX5	Comparative And With Index 5
306	(0)	CANX6	Comparative And With Index 6
307	(0)	CANX7	Comparative And With Index 7
015	(0)	CIOC	Connect I/O Channel
405	(0)	CMG	Compare With Magnitude
211	(0)	CMK	Compare Masked
115	(0)	CMPA	Compare With Accumulator
066	(1)	CMPB	Compare Bit Strings
106	(1)	CMPC	Compare Alphanumeric Character String
117	(0)	CMPAQ	Compare with A-Q
303	(1)	CMPN	Compare Numeric
116	(0)	CMPQ	Compare With Quotient Register
100	(0)	CMPX0	Compare With Index 0
101	(0)	CMPX1	Compare With Index 1
102	(0)	CMPX2	Compare With Index 2
103	(0)	CMPX3	Compare With Index 3
104	(0)	CMPX4	Compare With Index 4
105	(0)	CMPX5	Compare With Index 5
106	(0)	CMPX6	Compare With Index 6
107	(0)	CMPX7	Compare With Index 7
215	(0)	CNAA	Comparative Not With Accumulator
217	(0)	CNAAQ	Comparative Not With A-Q
216	(0)	CNAQ	Comparative Not With Quotient
200	(0)	CNAX0	Comparative Not With Index 0
201	(0)	CNAX1	Comparative Not With Index 1
202	(0)	CNAX2	Comparative Not With Index 2
203	(0)	CNAX3	Comparative Not With Index 3
204	(0)	CNAX4	Comparative Not With Index 4
205	(0)	CNAX5	Comparative Not With Index 5
206	(0)	CNAX6	Comparative Not With Index 6
207	(0)	CNAX7	Comparative Not With Index 7

Instruction

111	(0)	CWL	Compare With Limits
060	(0)	CSL	Combine Bit Strings Left
061	(0)	CSR	Combine Bit Strings Right
477	(0)	DFAD	Double Precision Floating Add
427	(0)	DFCMG	Double Precision Floating Compare Magnitude
517	(0)	DFCMP	Double Precision Floating Compare
527	(0)	DFDI	Double Precision Floating Divide Inverted
567	(0)	DFDV	Double Precision Floating Divide
433	(0)	DFLD	Double Precision Floating Load
463	(0)	DFMP	Double Precision Floating Multiply
473	(0)	DFRD	Double Precision Floating Round
577	(0)	DFSB	Double Precision Floating Subtract
457	(0)	DFST	Double Precision Floating Store
472	(0)	DFSTR	Double Precision Floating Store Round
616	(0)	DIS	Delay Until Interrupt Signal
506	(0)	DIV	Divide Integer
002	(0)	DRL	Derail
305	(1)	DTB	Decimal to Binary Convert
437	(0)	DUFA	Double Precision Unnormalized Floating Add
423	(0)	DUFM	Double Precision Unnormalized Floating Multiply
537	(0)	DUFS	Double Precision Unnormalized Floating Subtract
207	(1)	DV2D	Divide Using 2 Decimal Operands
227	(1)	DV3D	Divide Using 3 Decimal Operands
507	(0)	DVF	Divide Fraction
635	(0)	EAA	Effective Address to Accumulator
636	(0)	EAQ	Effective Address to Q
311	(0)	EASP0	Effective Address to Segment Number Field of PR0
310	(1)	EASP1	Effective Address to Segment Number Field of PR1
313	(0)	EASP2	Effective Address to Segment Number Field of PR2
312	(1)	EASP3	Effective Address to Segment Number Field of PR3
331	(0)	EASP4	Effective Address to Segment Number Field of PR4
330	(1)	EASP5	Effective Address to Segment Number Field of PR5
333	(0)	EASP6	Effective Address to Segment Number Field of PR6
332	(1)	EASP7	Effective Address to Segment Number Field of PR7
310	(0)	EAWP0	Effective Address to Word Number Field of PR0
311	(1)	EAWP1	Effective Address to Word Number Field of PR1
312	(0)	EAWP2	Effective Address to Word Number Field of PR2
313	(1)	EAWP3	Effective Address to Word Number Field of PR3
330	(0)	EAWP4	Effective Address to Word Number Field of PR4
331	(1)	EAWP5	Effective Address to Word Number Field of PR5
332	(0)	EAWP6	Effective Address to Word Number Field of PR6
333	(1)	EAWP7	Effective Address to Word Number Field of PR7
620	(0)	EAX0	Effective Address to Index 0
621	(0)	EAX1	Effective Address to Index 1
622	(0)	EAX2	Effective Address to Index 2
623	(0)	EAX3	Effective Address to Index 3
624	(0)	EAX4	Effective Address to Index 4
625	(0)	EAX5	Effective Address to Index 5
626	(0)	EAX6	Effective Address to Index 6
627	(0)	EAX7	Effective Address to Index 7
213	(0)	EPAQ	Effective Pointer to A-Q

Instruction

350	(1)	EPBP0	Effective Pointer at Base to PR0
351	(0)	EPBP1	Effective Pointer at Base to PR1
352	(1)	EPBP2	Effective Pointer at Base to PR2
353	(0)	EPBP3	Effective Pointer at Base to PR3
370	(1)	EPBP4	Effective Pointer at Base to PR4
371	(0)	EPBP5	Effective Pointer at Base to PR5
372	(1)	EPBP6	Effective Pointer at Base to PR6
373	(0)	EPBP7	Effective Pointer at Base to PR7
350	(0)	EPP0	Effective Pointer to PR0
351	(1)	EPP1	Effective Pointer to PR1
352	(0)	EPP2	Effective Pointer to PR2
353	(1)	EPP3	Effective Pointer to PR3
370	(0)	EPP4	Effective Pointer to PR4
371	(1)	EPP5	Effective Pointer to PR5
372	(0)	EPP6	Effective Pointer to PR6
373	(1)	EPP7	Effective Pointer to PR7
675	(0)	ERA	Exclusive Or to Accumulator
677	(0)	ERAQ	Exclusive Or to A-Q
676	(0)	ERQ	Exclusive Or to Quotient
655	(0)	ERSA	Exclusive Or to Storage Accumulator
656	(0)	ERSQ	Exclusive Or to Storage Quotient
640	(0)	ERSX0	Exclusive Or to Storage Index 0
641	(0)	ERSX1	Exclusive Or to Storage Index 1
642	(0)	ERSX2	Exclusive Or to Storage Index 2
643	(0)	ERSX3	Exclusive Or to Storage Index 3
644	(0)	ERSX4	Exclusive Or to Storage Index 4
645	(0)	ERSX5	Exclusive Or to Storage Index 5
646	(0)	ERSX6	Exclusive Or to Storage Index 6
647	(0)	ERSX7	Exclusive Or to Storage Index 7
660	(0)	ERX0	Exclusive Or to Index 0
661	(0)	ERX1	Exclusive Or to Index 1
662	(0)	ERX2	Exclusive Or to Index 2
663	(0)	ERX3	Exclusive Or to Index 3
664	(0)	ERX4	Exclusive Or to Index 4
665	(0)	ERX5	Exclusive Or to Index 5
666	(0)	ERX6	Exclusive Or to Index 6
667	(0)	ERX7	Exclusive Or to Index 7
475	(0)	FAD	Floating Add
425	(0)	FCMG	Floating Compare Magnitude
515	(0)	FCMP	Floating Compare
525	(0)	FDI	Floating Divide Inverted
565	(0)	FDV	Floating Divide
431	(0)	FLD	Floating Load
461	(0)	FMP	Floating Multiply
513	(0)	FNEG	Floating Negate
573	(0)	FNO	Floating Normalize
471	(0)	FRD	Floating Round
575	(0)	FSB	Floating Subtract
455	(0)	FST	Floating Store
470	(0)	FSTR	Floating Store Rounded
430	(0)	FSZN	Floating Set Zero and Negative Indicators
774	(0)	GTB	Gray to Binary

Instruction

760	(1)	LAR0	Load AR0
761	(1)	LAR1	Load AR1
762	(1)	LAR2	Load AR2
763	(1)	LAR3	Load AR3
764	(1)	LAR4	Load AR4
765	(1)	LAR5	Load AR5
766	(1)	LAR6	Load AR6
767	(1)	LAR7	Load AR7
463	(1)	LAREG	Load Address Registers
230	(0)	LBAR	Load Base Address Register
335	(0)	LCA	Load Complement into A
337	(0)	LCAQ	Load Complement into A-Q
674	(0)	LCPR	Load Central Processor Registers
336	(0)	LCQ	Load Complement into Q
320	(0)	LCX0	Load Complement into Index 0
321	(0)	LCX1	Load Complement into Index 1
322	(0)	LCX2	Load Complement into Index 2
323	(0)	LCX3	Load Complement into Index 3
324	(0)	LCX4	Load Complement into Index 4
325	(0)	LCX5	Load Complement into Index 5
326	(0)	LCX6	Load Complement into Index 6
327	(0)	LCX7	Load Complement into Index 7
235	(0)	LDA	Load Accumulator
034	(0)	LDAC	Load Accumulator and Clear
237	(0)	LDAQ	Load A-Q Register
232	(0)	LDBR	Load Descriptor Base Register
411	(0)	LDE	Load Exponent Register
634	(0)	LDI	Load Indicator Register
236	(0)	LDQ	Load Quotient Register
032	(0)	LDQC	Load Q and Clear
627	(0)	LDT	Load Timer Register
220	(0)	LDX0	Load Index 0
221	(0)	LDX1	Load Index 1
222	(0)	LDX2	Load Index 2
223	(0)	LDX3	Load Index 3
224	(0)	LDX4	Load Index 4
225	(0)	LDX5	Load Index 5
226	(0)	LDX6	Load Index 6
227	(0)	LDX7	Load Index 7
777	(0)	LLR	Long Left Rotate
737	(0)	LLS	Long Left Shift
467	(1)	LPL	Load Pointers and Lengths
173	(0)	LPRI	Load Pointer Register From ITS
760	(0)	LPRP0	Load Pointer Register 0 Packed
761	(0)	LPRP1	Load Pointer Register 1 Packed
762	(0)	LPRP2	Load Pointer Register 2 Packed
763	(0)	LPRP3	Load Pointer Register 3 Packed
764	(0)	LPRP4	Load Pointer Register 4 Packed
765	(0)	LPRP5	Load Pointer Register 5 Packed
766	(0)	LPRP6	Load Pointer Register 6 Packed
767	(0)	LPRP7	Load Pointer Register 7 Packed

Instruction

257	(1)	LPTP	Load Page Table Pointers
173	(1)	LPTR	Load Page Table Registers
774	(1)	LRA	Load Ring Alarm
073	(0)	LREG	Load Registers
773	(0)	LRL	Long Right Logical
733	(0)	LRS	Long Right Shift
257	(0)	LSDP	Load Segment Descriptor Pointers
232	(1)	LSDR	Load Segment Descriptor Registers
720	(0)	LXL0	Load Index 0 from Lower
721	(0)	LXL1	Load Index 1 from Lower
722	(0)	LXL2	Load Index 2 from Lower
723	(0)	LXL3	Load Index 3 from Lower
724	(0)	LXL4	Load Index 4 from Lower
725	(0)	LXL5	Load Index 5 from Lower
726	(0)	LXL6	Load Index 6 from Lower
727	(0)	LXL7	Load Index 7 from Lower
100	(1)	MLR	Move Alphanumeric Left to Right
001	(0)	MME1	Master Mode Entry 1
004	(0)	MME2	Master Mode Entry 2
005	(0)	MME3	Master Mode Entry 3
007	(0)	MME4	Master Mode Entry 4
206	(1)	MP2D	Multiply Using 2 Decimal Operands
226	(1)	MP3D	Multiply Using 3 Decimal Operands
401	(0)	MPF	Multiply Fraction
402	(0)	MPY	Multiply Integer
101	(1)	MRL	Move Alphanumeric Left to Right
020	(1)	MVE	Move Alphanumeric Edited
300	(1)	MVN	Move Numeric
024	(1)	MVNE	Move Numeric Edited
160	(1)	MVT	Move Alphanumeric with Translation
660	(1)	NAR0	Numeric Descriptor to AR0
661	(1)	NAR1	Numeric Descriptor to AR1
662	(1)	NAR2	Numeric Descriptor to AR2
663	(1)	NAR3	Numeric Cescriptor to AR3
664	(1)	NAR4	Numeric Descriptor to AR4
665	(1)	NAR5	Numeric Descriptor to AR5
666	(1)	NAR6	Numeric Descriptor to AR6
667	(1)	NAR7	Numeric Descriptor to AR7
531	(0)	NEG	Negate Accumulator
533	(0)	NEGL	Negate Long
011	(0)	NOP	No Operation
275	(0)	ORA	Or to Accumulator
277	(0)	ORAQ	Or to A-Q
276	(0)	ORQ	Or to Quotient
255	(0)	ORSA	Or to Storage Accumulator
256	(0)	ORSQ	Or to Storage Quotient
240	(0)	ORSX0	Or to Storage Index 0
241	(0)	ORSX1	Or to Storage Index 1
242	(0)	ORSX2	Or to Storage Index 2

Instruction

243	(0)	ORSX3	Or to Storage Index 3
244	(0)	ORSX4	Or to Storage Index 4
245	(0)	ORSX5	Or to Storage Index 5
246	(0)	ORSX6	Or to Storage Index 6
247	(0)	ORSX7	Or to Storage Index 7
260	(0)	ORX0	Or to Index 0
261	(0)	ORX1	Or to Index 1
262	(0)	ORX2	Or to Index 2
263	(0)	ORX3	Or to Index 3
264	(0)	ORX4	Or to Index 4
265	(0)	ORX5	Or to Index 5
266	(0)	ORX6	Or to Index 6
267	(0)	ORX7	Or to Index 7
776	(0)	QLR	Quotient Left Rotate
736	(0)	QLS	Quotient Left Shift
772	(0)	QRL	Quotient Right Logical
732	(0)	QRS	Quotient Right Shift
633	(0)	RCCL	Read Calendar Clock
613	(0)	RCU	Restore Control Unit
630	(0)	RET	Return
233	(0)	RMCM	Read Memory Controller Mask
560	(0)	RPD	Repeat Double
500	(0)	RPL	Repeat Link
520	(0)	RPT	Repeat
413	(0)	RSCR	Read System Controller Registers
231	(0)	RSW	Read Switches
610	(0)	RTCD	Return Control Double
522	(1)	S4BD	Subtract 4-bit Character Displacement from AR
521	(1)	S6BD	Subtract 6-bit Character Displacement from AR
520	(1)	S9BD	Subtract 9-bit Character Displacement from AR
740	(1)	SAR0	Store AR0
741	(1)	SAR1	Store AR1
742	(1)	SAR2	Store AR2
743	(1)	SAR3	Store AR3
744	(1)	SAR4	Store AR4
745	(1)	SAR5	Store AR5
746	(1)	SAR6	Store AR6
747	(1)	SAR7	Store AR7
440	(1)	SAREG	Store Address Registers
203	(1)	SB2D	Subtract Using 2 decimal operands
223	(1)	SB3D	Subtract Using 3 decimal operands
175	(0)	SBA	Subtract from Accumulator
350	(0)	SBAR	Store Base Address Register
177	(0)	SBAQ	Subtract From A-Q
523	(1)	SBD	Subtract Bit Displacement from AR
135	(0)	SBLA	Subtract Logical from Accumulator
137	(0)	SBLAQ	Subtract Logical from A-Q
136	(0)	SBLQ	Subtract Logical from Quotient

Instruction

120	(0)	SBLX0	Subtract Logical from Index 0
121	(0)	SBLX1	Subtract Logical from Index 1
122	(0)	SBLX2	Subtract Logical from Index 2
123	(0)	SBLX3	Subtract Logical from Index 3
124	(0)	SBLX4	Subtract Logical from Index 4
125	(0)	SBLX5	Subtract Logical from Index 5
126	(0)	SBLX6	Subtract Logical from Index 6
127	(0)	SBLX7	Sbutract Logical from Index 7
176	(0)	SBQ	Subtract From Quotient
160	(0)	SBX0	Subtract From Index 0
161	(0)	SBX1	Subtract From Index 1
162	(0)	SBX2	Subtract From Index 2
163	(0)	SBX3	Subtract From Index 3
164	(0)	SBX4	Subtract From Index 4
165	(0)	SBX5	Subtract From Index 5
166	(0)	SBX6	Subtract From Index 6
167	(0)	SBX7	Subtract From Index 7
120	(1)	SCD	Scan Character Double
121	(1)	SCDR	Scan Character Double in Reverse
124	(1)	SCM	Scan with Mask
125	(1)	SCMR	Scan with Mask in Reverse
452	(0)	SCPR	Store Control Processor Register
657	(0)	SCU	Store Control Unit
154	(0)	SDBR	Store Descriptor Base Register
553	(0)	SMCM	Set Memory Controller Mask
451	(0)	SMIC	Set Memory Interrupt Cells
250	(1)	SPBP0	Store Segment Base Pointer of PR0
251	(0)	SPBP1	Store Segment Base Pointer of PR1
252	(1)	SPBP2	Store Segment Base Pointer of PR2
253	(0)	SPBP3	Store Segment Base Pointer of PR3
650	(1)	SPBP4	Store SEgment Base Pointer of PR4
651	(0)	SPBP5	Store Segment Base Pointer of PR5
652	(1)	SPBP6	Store Segment Base Pointer of PR6
653	(0)	SPBP7	Store Segment Base Pointer of PR7
447	(1)	SPL	Store Pointer and Lengths
254	(0)	SPRI	Store Pointer Register as ITS
250	(0)	SPRI0	Store Pointer Register 0 as ITS
251	(1)	SPRI1	Store Pointer Register 1 as ITS
252	(0)	SPRI2	Store Pointer Register 2 as ITS
253	(1)	SPRI3	Store Pointer Register 3 as ITS
650	(0)	SPRI4	Store Pointer Register 4 as ITS
651	(1)	SPRI5	Store Pointer Register 5 as ITS
652	(0)	SPRI6	Store Pointer Register 6 as ITS
653	(1)	SPRI7	Store Pointer Register 7 as ITS
540	(0)	SPRP0	Store Pointer Register 0 Packed
641	(0)	SPRP1	Store Pointer Register 1 Packed
542	(0)	SPRP2	Store Pointer Register 2 Packed
543	(0)	SPRP3	Store Pointer Register 3 Packed
544	(0)	SPRP4	Store Pointer Register 4 Packed
545	(0)	SPRP5	Store Pointer Register 5 Packed
546	(0)	SPRP6	Store Pointer Register 6 Packed
547	(0)	SPRP7	Store Pointer Register 7 Packed

Instruction

557	(1)	SPTP	Store Page Table Pointers
154	(1)	SPTR	Store Page Table Registers
754	(1)	SRA	Store Ring Alarm Register
753	(0)	SREG	Store Registers
155	(0)	SSA	Subtract Stored from Accumulator
057	(0)	SSCR	Set System Controller Register
557	(0)	SSDP	Store Segment Descriptor Pointers
254	(1)	SSDR	Store Segment Descriptor Registers
156	(0)	SSQ	Subtract Stored from Quotient
140	(0)	SSX0	Subtract Stored from Index 0
141	(0)	SSX1	Subtract Stored from Index 1
142	(0)	SSX2	Subtract Stored from Index 2
143	(0)	SSX3	Subtract Stored from Index 3
144	(0)	SSX4	Subtract Stored from Index 4
145	(0)	SSX5	Subtract Stored form Index 5
146	(0)	SSX6	Subtract Stored from Index 6
147	(0)	SSX7	Subtract Stored from Index 7
755	(0)	STA	Store Accumulator
354	(0)	STAC	Store A Conditional
654	(0)	STACQ	Store A Conditional on Q
757	(0)	STAQ	Store A-Q
551	(0)	STBA	Store 9 Bit Character of A
552	(0)	STBQ	Store 9 Bit Character of Q
554	(0)	STC1	Store Instruction Counter + 1
750	(0)	STC2	Store Instruction Counter + 2
751	(0)	STCA	Store 6 bit Character of A
357	(0)	STCD	Store Control Double
752	(0)	STCQ	Store 6 bit Character of Q
456	(0)	STE	Store Exponent Register
754	(0)	STI	Store Indicators
756	(0)	STQ	Store Quotient
454	(0)	STT	Store Timer Register
740	(0)	STX0	Store Index 0
741	(0)	STX1	Store Index 1
742	(0)	STX2	Store Index 2
743	(0)	STX3	Store Index 3
744	(0)	STX4	Store Index 4
745	(0)	STX5	Store Index 5
746	(0)	STX6	Store Index 6
747	(0)	STX7	Store Index 7
450	(0)	STZ	Store Zero
171	(0)	SWCA	Subtract with Carry from Accumulator
172	(0)	SWCQ	Subtract with Carry from Quotient
440	(0)	SXL0	Store Index 0 in Lower
441	(0)	SXL1	Store Index 1 in Lower
442	(0)	SXL2	Store Index 2 in Lower
443	(0)	SXL3	Store Index 3 in Lower
444	(0)	SXL4	Store Index 4 in Lower
445	(0)	SXL5	Store Index 5 in Lower
446	(0)	SXL6	Store Index 6 in Lower
447	(0)	SXL7	Store Index 7 in Lower

Instruction

234	(0)	SZN	Set Zero and Negative Indicators
214	(0)	SZNC	Set Zero and Negative Indicators and Clear
064	(1)	SZTL	Set Zero and Truncation Indicators with Bit String Left
065	(1)	SZTR	Set Zero and Truncation Indicators with Bit String Right
164	(1)	TCT	Test Character and Translate
165	(1)	TCTR	Test Character and Translate in Reverse
614	(0)	TEO	Transfer on Exponent Overflow
615	(0)	TEU	Transfer on Exponent Underflow
604	(0)	TMI	Transfer on Minus
604	(1)	TMOZ	Transfer on Minus or Zero
602	(0)	TNC	Transfer on No Carry
601	(0)	TNZ	Transfer on Not Zero
617	(0)	TOV	Transfer on Overflow
605	(0)	TPL	Transfer on Plus
602	(1)	TPNZ	Transfer on Plus and Nonzero
710	(0)	TRA	Transfer Unconditionally
603	(0)	TRC	Transfer on Carry
601	(1)	TRTF	Transfer on Truncation Indicator OFF
600	(1)	TRTN	Transfer on Truncation Indicator ON
270	(0)	TSP0	Transfer And Set Pointer Register 0
271	(0)	TSP1	Transfer And Set Pointer Register 1
272	(0)	TSP2	Transfer And Set Pointer Register 2
273	(0)	TSP3	Transfer And Set Pointer Register 3
670	(0)	TSP4	Transfer And Set Pointer Register 4
671	(0)	TSP5	Transfer And Set Pointer Register 5
672	(0)	TSP6	Transfer And Set Pointer Register 6
673	(0)	TSP7	Transfer And Set Pointer Register 7
715	(0)	TSS	Transfer And Set Slave
700	(0)	TSX0	Transfer And Set Index 0
701	(0)	TSX1	Transfer And Set Index 1
702	(0)	TSX2	Transfer And Set Index 2
703	(0)	TSX3	Transfer And Set Index 3
704	(0)	TSX4	Transfer And Set Index 4
705	(0)	TSX5	Transfer And Set Index 5
706	(0)	TSX6	Transfer And Set Index 6
707	(0)	TSX7	Transfer And Set Index 7
607	(0)	TTF	Transfer On Tally Indicator Off
606	(1)	TTN	Transfer On Tally Indicator On
600	(0)	TZE	Transfer on Zero
435	(0)	UFA	Unnormalized Floating Add
421	(0)	UFM	Unnormalized Floating Multiply
535	(0)	UFS	Unnormalized Floating Subtract
716	(0)	XEC	Execute
717	(0)	XED	Execute Double

	000	001	002	003	004	005	006	007	010	011	012	013	014	015	016	017
000		MME	DRL		(MME2)	(MME3)		(MME4)		NOP	PULS1	PULS2		CIOC		
020	ADLX0	ADLX1	ADLX2	ADLX3	ADLX4	ADLX5	ADLX6	ADLX7			LDQC	ADL	LDAC	ADLA	ADLQ	ADLAQ
040	ASX0	ASX1	ASX2	ASX3	ASX4	ASX5	ASX6	ASX7	(ADWP0)	(ADWP1)	(ADWP2)	(ADWP3)	AOS	ASA	ASQ	SSCR
060	ADX0	ADX1	ADX2	ADX3	ADX4	ADX5	ADX6	ADX7		AWCA	AWCQ	LREG		ADA	ADQ	ADAQ
100	CMPX0	CMPX1	CMPX2	CMPX3	CMPX4	CMPX5	CMPX6	CMPX7		CWL				CMPA	CMPQ	CMPAQ
120	SBLX0	SBLX1	SBLX2	SBLX3	SBLX4	SBLX5	SBLX6	SBLX7						SBLA	SBLQ	SBLAQ
140	SSX0	SSX1	SSX2	SSX3	SSX4	SSX5	SSX6	SSX7	(ADWP4)	(ADWP5)	(ADWP6)	(ADWP7)	(SDBR)	SSA	SSQ	
160	SBX0	SBX1	SBX2	SBX3	SBX4	SBX5	SBX6	SBX7		SWCA	SWCQ	(LPRI)		SBA	SBQ	SBAQ
200	CNAX0	CNAX1	CNAX2	CNAX3	CNAX4	CNAX5	CNAX6	CNAX7		CMK	(ABSA)	(EPAQ)	SZNC	CNAA	CNAQ	CNAAQ
220	LDX0	LDX1	LDX2	LDX3	LDX4	LDX5	LDX6	LDX7	LBAR	RSW	(LDBR)	RMCM	SZN	LDA	LDQ	LDAQ
240	ORSX0	ORSX1	ORSX2	ORSX3	ORSX4	ORSX5	ORSX6	ORSX7	(SPRI0)	(SPBP1)	(SPRI2)	(SPBP3)	(SPRI)	ORSA	ORSQ	(LSDP)
260	ORX0	ORX1	ORX2	ORX3	ORX4	ORX5	ORX6	ORX7	(TSP0)	(TSP1)	(TSP2)	(TSP3)		ORA	ORQ	ORAQ
300	CANX0	CANX1	CANX2	CANX3	CANX4	CANX5	CANX6	CANX7	(EAWP0)	(EASP0)	(EAWP2)	(EASP2)		CANA	CANQ	CANAQ
320	LCX0	LCX1	LCX2	LCX3	LCX4	LCX5	LCX6	LCX7	(EAWP4)	(EASP4)	(EAWP6)	(EASP6)		LCA	LCQ	LCAQ
340	ANSX0	ANSX1	ANSX2	ANSX3	ANSX4	ANSX5	ANSX6	ANSX7	(EPP0)	(EPBP1)	(EPP2)	(EPBP3)	(STAC)	ANSA	ANSQ	(STCD)
360	ANX0	ANX1	ANX2	ANX3	ANX4	ANX5	ANX6	ANX7	(EPP4)	(EPBP5)	(EPP6)	(EPBP7)		ANA	ANQ	ANAQ
400		MPF	MPY			CMG				LDE		RSCR		ADE		
420		UFM		DUFM		FCMG		DFCMG	FSZN	FLD		DFLD		UFA		DUFA
440	SXL0	SXL1	SXL2	SXL3	SXL4	SXL5	SXL6	SXL7	STZ	SMIC	SCPR		STT	FST	STE	DFST
460		FMP		DFMP					FSTR	FRD	DFSTR	DFRD		FAD		DFAD
500	RPL					BCD	DIV	DVF				FNEG		FCMP		DFCMP
520	RPT					FDI		DFDI		NEG	(CAMS)	NECL		UFS		DUFS
540	(SPRP0)	(SPRP1)	(SPRP2)	(SPRP3)	(SPRP4)	(SPRP5)	(SPRP6)	(SPRP7)	SBAR			SMCM	STC1			(SSDP)
560	RPD					FDV		DFDV		STBA	STBQ	FN0		FSB		DFSB
600	TZE	TNZ	TNC	TRC	TMI	TPL		TTF	(RTCD)			(RCU)	TEO	TEU	DIS	TOV
620	EAX0	EAX1	EAX2	EAX3	EAX4	EAX5	EAX6	EAX7	RET			(RCCL)	LDI	EAA	EAQ	LDT
640	ERSX0	ERSX1	ERSX2	ERSX3	ERSX4	ERSX5	ERSX6	ERSX7	(SPRI4)	(SPBP5)	(SPBI6)	(SPBP7)		ERSA	ERSQ	(SCU)
660	ERX0	ERX1	ERX2	ERX3	ERX4	ERX5	ERX6	ERX7	(TSP4)	(TSP5)	(TSP6)	(TSP7)	LCPR	ERA	ERQ	ERAQ
700	TSX0	TSX1	TSX2	TSX3	TSX4	TSX5	TSX6	TSX7	TRA			(CALL6)		TSS	XEC	XED
720	LXL0	LXL1	LXL2	LXL3	LXL4	LXL5	LXL6	LXL7		ARS	QRS	LRS		ALS	QLS	LLS
740	STX0	STX1	STX2	STX3	STX4	STX5	STX6	STX7	STC2	STCA	STCQ	SREG	STI	STA	STQ	STAQ
760	(LPRP0)	(LPRP1)	(LPRP2)	(LPRP3)	(LPRP4)	(LPRP5)	(LPRP6)	(LPRP7)		ARL	QRL	LRL	GTB	ALR	QLR	LLR

Left Half

Right Half

	000	001	002	003	004	005	006	007	010	011	012	013	014	015	016	017
000																
020	MVE				MVNE											
040																
060	CSL	CSR			SZTL	SZTR	CMPB									
100	MLR	MRL					CMPC									
120	SCD	SCDR			SCM	SCMR							(SPTR)			
140																
160	MVT				TCT	TCTR						(LPTR)				
200			AD2D	SB2D			MP2D	DV2D			(LSDR)					
220			AD3D	SB3D			MP3D	DV3D								
240									(SPBP0)	(SPRI1)	(SPBP2)	(SPRI3)	(SSDR)			(LPTP)
260																
300	MVN	BTD		CMPN		DTB			(EASP1)	(EAWP1)	(EASP3)	(EAWP3)				
320									(EASP5)	(EAWP5)	(EASP7)	(EAWP7)				
340								DVDR	(EPBP0)	(EPP1)	(EPBP2)	(EPP3)				
360									(EPBP4)	(EPP5)	(EPBP4)	(EPP7)				
400																
420																
440				SAREG				SPL								
460				LAREG				LPL								
500	A9BD	A6BD	A4BD	ABD				AWD								
520	S9BD	S6BD	S4BD	SBD				SWD			(CAMP)					
540	ARA0	ARA1	ARA2	ARA3	ARA4	ARA5	ARA6	ARA7								(SPTP)
560	AAR0	AAR1	AAR2	AAR3	AAR4	AAR5	AAR6	AAR7								
600	TRTN	TRTF			TMOZ	TPNZ	TTN									
620																
640	ARN0	ARN1	ARN2	ARN3	ARN4	ARN5	ARN6	ARN7	(SPBP4)	(SPRI5)	(SPBP6)	(SPRI7)				
660	NAR0	NAR1	NAR2	NAR3	NAR4	NAR5	NAR6	NAR7								
700																
720																
740	SAR0	SAR1	SAR2	SAR3	SAR4	SAR5	SAR6	SAR7					(SRA)		(SPTR)	
760	LAR0	LAR1	LAR2	LAR3	LAR4	LAR5	LAR6	LAR7					(LRA)			

Left Half

Right Half