SYSTEM 801
Principles of Operation

Version 2 November 21, 1975
Updated | December 12, 1975
* January 16, 1976

[redacted]

TABLE OF CONTENTS

1. System Architecture 1
1.1 Central Processing Unit 3
1.2 Register Organization 4
1.2.1 The Instruction Address Register 6
1.2.2 The MQ Register 6
1.2.3 The Condition Register 6
1.3 Instruction Formats 10
1.4 Interrupts 11
1.5 Input and Output 14
1.5.1 The External Interrupt Adapter 14
1.5.2 Input/Output Interface 14

2. Storage Access 15
2.1 Instructions 15

3. Address Computation 19
3.1 Instructions 19

4. Branching 20
4.1 Invalid Branch Address 20
4.2 Branch With Execute Instructions 21
4.3 Instructions 21

5. Traps 26
5.1 Instructions 26

6. Moves and Inserts 28
6.1 Instructions 28

7. Arithmetic 36
7.1 Instructions 36

8. Logical Operations 43
8.1 Instructions 43

9. Shifts 47
9.1 Instructions 47

10. System Control 53
10.1 Locking 53
10.2 Cache Control Operations 53
10.3 Instructions 54

11. Input/Output Control 58
11.1 Instructions 58

12. Index By Code 59

13. Index By Mnemonic 63

1. System Architecture

Logically an 801 system consists of main storage, a central processing unit, low-speed and high-speed input/output devices, and an interrupt adapter. This structure is shown in Figure 1.

Main storage provides the system with directly accessable fast access storage of data. Both data and programs must be loaded into main storage (from input devices) before they can be processed.

No processing of data occurs in main storage, either implicitly or explicitly. All data must be loaded into high-speed CPU storage called registers before it can be operated upon. Main storage may be either physically integrated with the CPU or may be constructed as stand-alone units. Additionally, main storage may be composed of large volume storage and faster access buffer storages, called caches. A CPU may have no cache, a cache for data references, a cache for instruction references, or individual caches for instruction and data references.

Unlike many systems, the fetching of instructions and the fetching and storing of data are not tightly coupled. Whether or not a cache for instructions is present, the CPU always attempts to prefetch one or more instructions. Hence, modification of an instruction by a program may not be seen when that instruction is fetched for execution, unless explicit steps have been taken to ensure that all pre-fetched instructions have been invalidated. For the purpose special instructions to control the caches have been provided.

Again, unlike many systems with caches, the data cache is not tightly coupled to the flow of data to or from input/output devices. For low-speed devices, which communicate directly with the CPU through its registers, this creates no problem. For high-speed input/output devices which may access memory directly, the program must ensure that, where necessary, updated data in cache is placed in storage prior to output, and that storage updated by input is correctly reflected in the data cache. Again, the cache control instructions can be used to guarantee correct results in these situations.

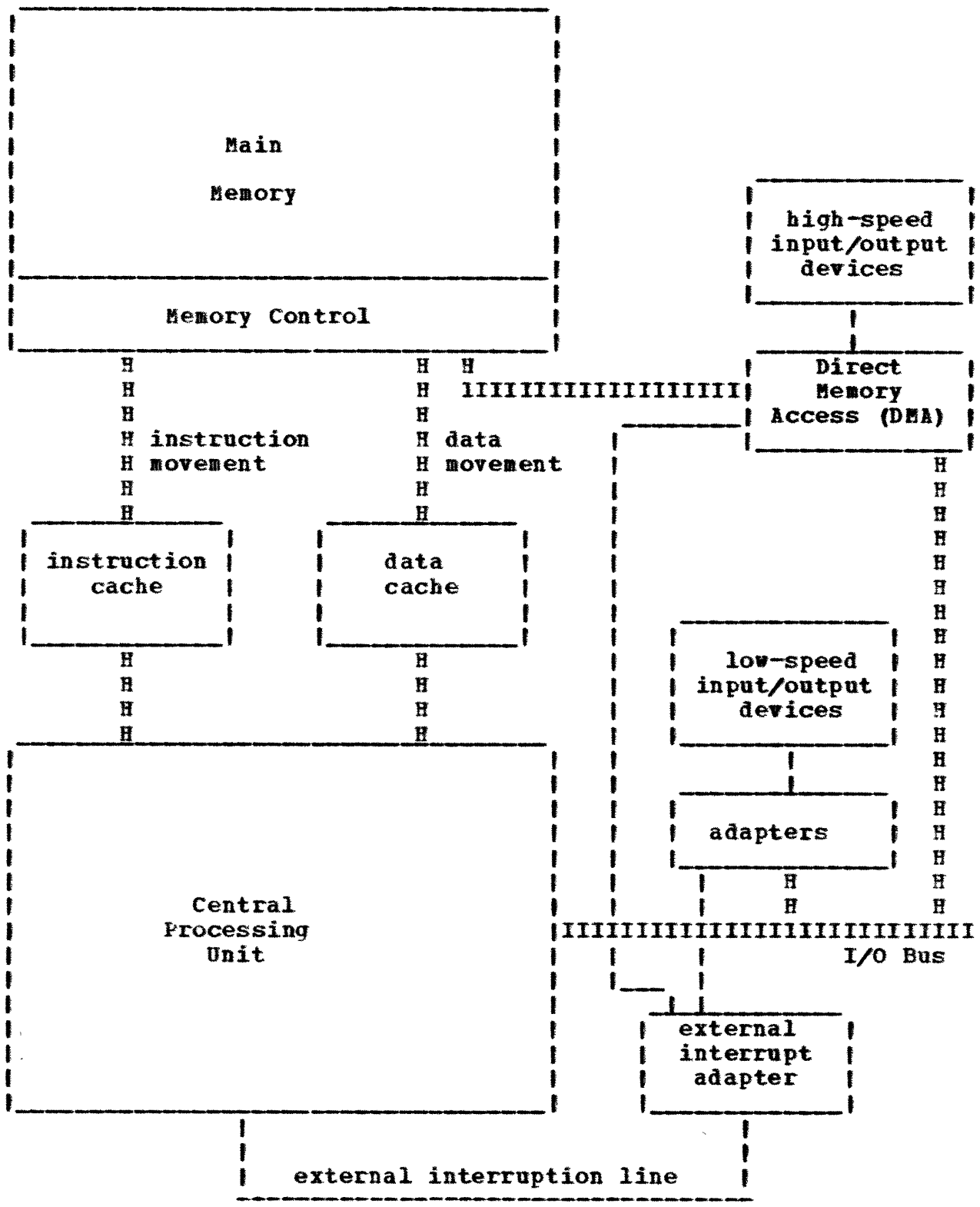


Figure 1 - SYSTEM ORGANIZATION

Fetching and storing of data by the CPU are not affected by any concurrent direct memory access input/output data transfer. When concurrent requests to a main storage location occur, access is normally granted in a predetermined sequence that assigns highest priority to input/output requests. If the first reference changes the contents, any subsequent storage fetches obtain the new contents, although, as noted above, care must be taken to ensure the synchronization of data and instruction caches where concurrent accesses are possible.

Main storage may be volatile or nonvolatile. If it is volatile, the contents of main storage are not preserved when power is turned off. If it is nonvolatile, turning power on or off does not affect the contents of main storage provided the CPU is in the stopped state and no references are made to main storage by input/output devices which can directly access storage. The organization of storage is shown in Figure 2.

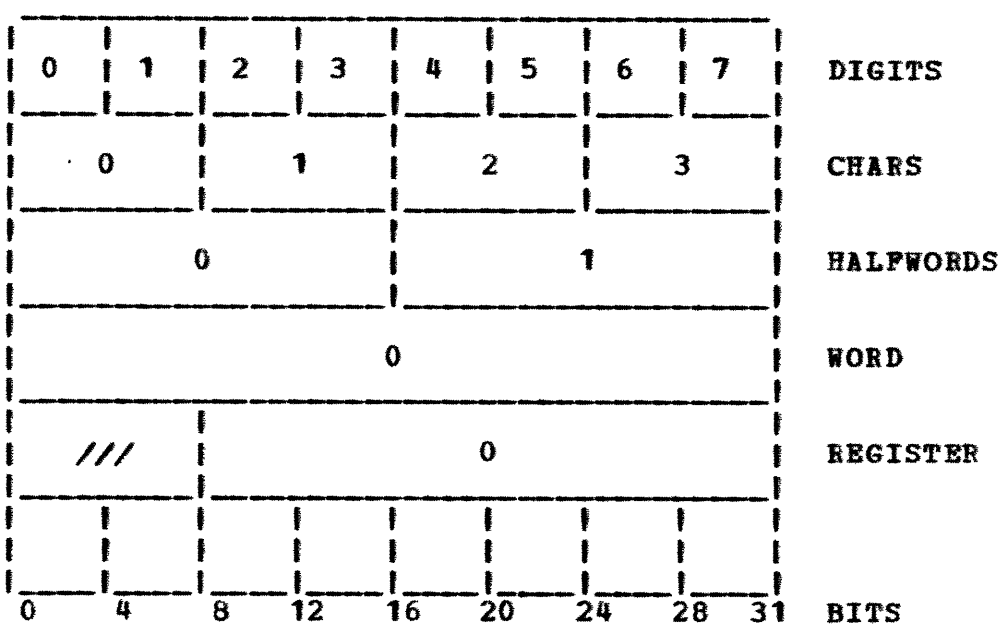


Figure 2 - STORAGE ORGANIZATION

1.1 Central Processing Unit

The central processing unit (CPU) is the controlling center of the system. It contains the sequencing and processing controls for instruction execution, interruption action, initial program loading, and other system-related functions.

The CPU, in processing instructions, attempts to

achieve the greatest instruction processing rate possible. Instructions which do not require storage access may be executed while storage is being accessed for some previous instruction, and the CPU attempts to pre-fetch instructions whenever possible, overlapping such fetches with the execution of other instructions. Since a successful branch instruction requires the fetching of an instruction out of sequential order, means are provided to permit execution of one instruction while fetching the target instruction of the branch. Control is maintained over such pre-fetching so that instruction execution appears to take place in the order intended by the programmer.

Instructions which the CPU can execute fall into ten classes; storage access, address computation, branching, trap, move, arithmetic, logical operation, shift, system control, and input/output instructions. A separate section is devoted to each instruction class.

1.2 Register Organization

All manipulation of information is done in high-speed storage internal to the central processing unit (CPU). The principal storage internal to the CPU is a set of sixteen registers. Each register consists of 24 bits. These registers contain a prefix of eight bits and a half of sixteen bits. The half consists of two characters, C0 and C1, of eight bits each. The half is also considered to consist of four digits, D0, D1, D2, and D3, each containing four bits. The register organization is shown in Figure 3.

To avoid the destruction of operands in certain operations, some instructions provide for the result of the operation to be placed in the twin of one of the source operand registers. The register twin to a given register has the name, in binary, of the given register, with the low-order bit inverted. Thus, the twin of register 5 (binary 0101) is register 4 (binary 0100), and the twin of register 12 (binary 1100) is register 13(binary 1101).

For computational purposes, registers are treated as signed quantities, unsigned positive quantities, or, where required, as unstructured logical quantities. A register may be treated as (a) an algebraic quantity consisting of a sign bit and twenty-three low-order integer bits, in two's complement representation, or (b) an unsigned positive integer of twenty-four bits.

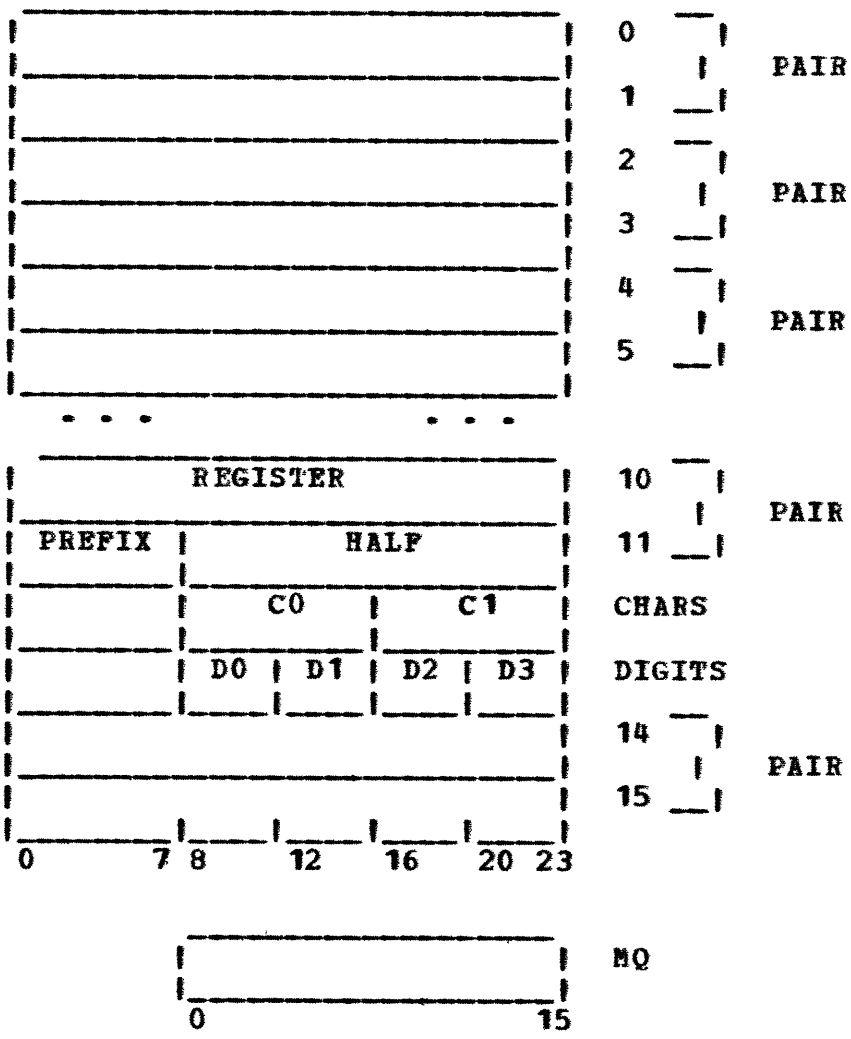


Figure 3 - REGISTER ORGANIZATION

Three additional special-purpose registers exist within the CPU. These registers are known as (a) the instruction address register (IAR), (b) the MQ-register, and (c) the condition register (CR). These registers are implicitly or explicitely addressed for a particular action or operation, and they may be accessed, used, or altered by a multiplicity of instructions.

1.2.1 The Instruction Address Register

The instruction address register (IAR), as shown in Figure 4, is a 24-bit register which normally contains the address of the next instruction to be executed. Since all instructions are constrained to lie on half-word boundaries, the low-order bit (bit 23) of the instruction address register is constrained to be zero.

Figure 4 - INSTRUCTION ADDRESS REGISTER

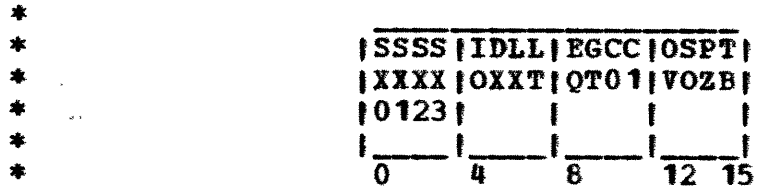
As a rule, the content of the instruction address register is incremented by the length of the current instruction during the process of decoding that instruction. Should this instruction be a successful branch instruction, the content of the instruction address register will be changed to the address of the branch target instruction, as given by the branch instruction.

1.2.2 The MQ Register

The MQ-register (MQ) is a 16-bit register whose primary use is to provide a register extension to accomodate the product for the Multiply Step instruction and the dividend for the Divide Step instruction.

1.2.3 The Condition Register

The condition register (CR) is a 16-bit register used to reflect the effect of certain operations, to provide a mechanism for testing (and branching) on a bit or condition, and to provide a 'parity stack' to indicate which bytes of the last four half words loaded were addressed. The condition register is shown in Figure 5.



SX0 -- PARITY STACK ZERO
SX1 -- PARITY STACK ONE
SX2 -- PARITY STACK TWO
SX3 -- PARITY STACK THREE

IO -- I/O Busy
DX -- DECIMAL EXCEPTION
LX -- LOCK EXCEPTION LATCH
LT -- COMPARES LESS THAN, NEG VALUE

EQ -- COMPARES EQUAL, ZERO VALUE
GT -- COMPARES GREATER THAN, POS VALUE
C0 -- CARRY FROM BIT0 OR COMPARE C0 =
C1 -- CARRY FROM BIT8 OR COMPARE C1 =

OV -- OVERFLOW LATCH
SO -- SUMMARY OVERFLOW LATCH
PZ -- PERMANENT ZERO BIT
TB -- TEST BIT

Figure 5 - THE CONDITION REGISTER

The first four bits of the condition register are used for the parity stack (SX). Bit 0 is known as SX0, bit 1 as SX1, bit 2 as SX2, and bit 3 as SX3. An effect of an instruction that loads the half of a register is to push the parity stack down one position. The lowest bit of the stack (SX3) is lost, while the low-order bit of the storage address is placed on the top of the stack, above the previous top three stack elements.

Bit 4 of the condition register is set by the IOR and IOW instructions. It is set to one if the I/O adapter selected by one of these instructions cannot accept the command. It is set to zero if the adapter accepts the command.

Bit 5 of the condition register is the decimal exception latch (DX). If the decimal feature is not installed, this bit is also reserved, and is set to zero whenever the condition register is loaded. If the decimal feature is installed, this bit is set by the decimal instructions (Add and Subtract Decimal) to one or zero if an exception condition is or is not, respectively, detected.

Bit 6 of the condition register is the lock exception latch (LX). If the lock feature is not installed, this bit is also reserved, and is set to zero whenever the condition register is loaded. If the lock feature is installed, the use of this bit is described under the definition of the establish lock instruction.

Bit 7 of the condition register is the less-than latch (LT). It is set to one by comparison operations if the first comparand is less than the second comparand. It is set to one by certain other arithmetic and logical operations if the result is negative or if the high-order bit of the result is one.

Bit 8 of the condition register is the equal latch (EQ). It is set to one by comparison operations if the first comparand equals the second comparand. It is set to one by certain other logical and arithmetic operations if the result is zero, or if all bits of the result are zeros.

Bit 9 of the condition register is the greater-than latch (GT). It is set to one by comparison operations if the first comparand is greater than the second comparand. It is set to one by certain other arithmetic and logical operations if the result is positive or if the high-order bit of a non-zero result is zero.

Bit 10 of the condition register is the carry-zero latch (C0). It is set to one by certain arithmetic instructions if the operation generates a carry out of bit position zero. It also functions as a special-purpose indicator for the Divide Step and Multiply Step instructions. This latch is set by logical compare instructions to show equality/inequality of character C0 of the comparands.

Bit 11 of the condition register is the carry-one latch (C1). It is set to one by certain arithmetic instructions if the operation generates a carry out of bit position eight. This latch is also set by logical compare instructions to show equality/inequality of character C1 of the comparands.

Bit 12 of the condition register is the overflow latch (OV), which is set to one when arithmetic and certain shift operations overflow. It also functions as a special purpose indicator for the Divide Step instruction.

Bit 13 of the condition register is the summary-overflow latch (SO). Whenever an instruction sets the overflow latch, it resets the summary-overflow latch to the OR of the overflow-latch with the old value of the summary-overflow latch.

Bit 14 of the condition register is the permanent-zero

bit (PZ). It is set to zero whenever the condition register is loaded, and it cannot be reset to one. Its presence provides for a guaranteed branch in the BI format by use of the Branch On Not-Bit instruction, where the permanent zero bit is specified.

Bit 15 of the condition register is the test bit (TB). A bit may be moved to or from an arbitrary bit position in a half from or to the test bit of the condition register through use of the Move From/To Test Bit Indexed/Value instructions.

All bits of the condition register, save those required to be zeros, can be arbitrarily set through use of the Move To Condition Register instruction. Additionally, any individual bit of the condition register may be set to an arbitrary value by use of the Insert Condition Bit Immediate instruction, except, of course, those bits required to be zero.

1.3 Instruction Formats

The five instruction formats, X, R, D, BI, and BA, are shown in Figure 6. For X and D instructions that refer to storage, and for I/O instructions, address calculation is according to the formulas:

X-Format (RB) + 0/(RC)

D-Format 0/(RC) + (0(bits 0-7) || I)

where 0/(RC) indicates the value 0 if RC is specified as 0, else the contents of register RC. I is treated as an unsigned 16 bit integer.

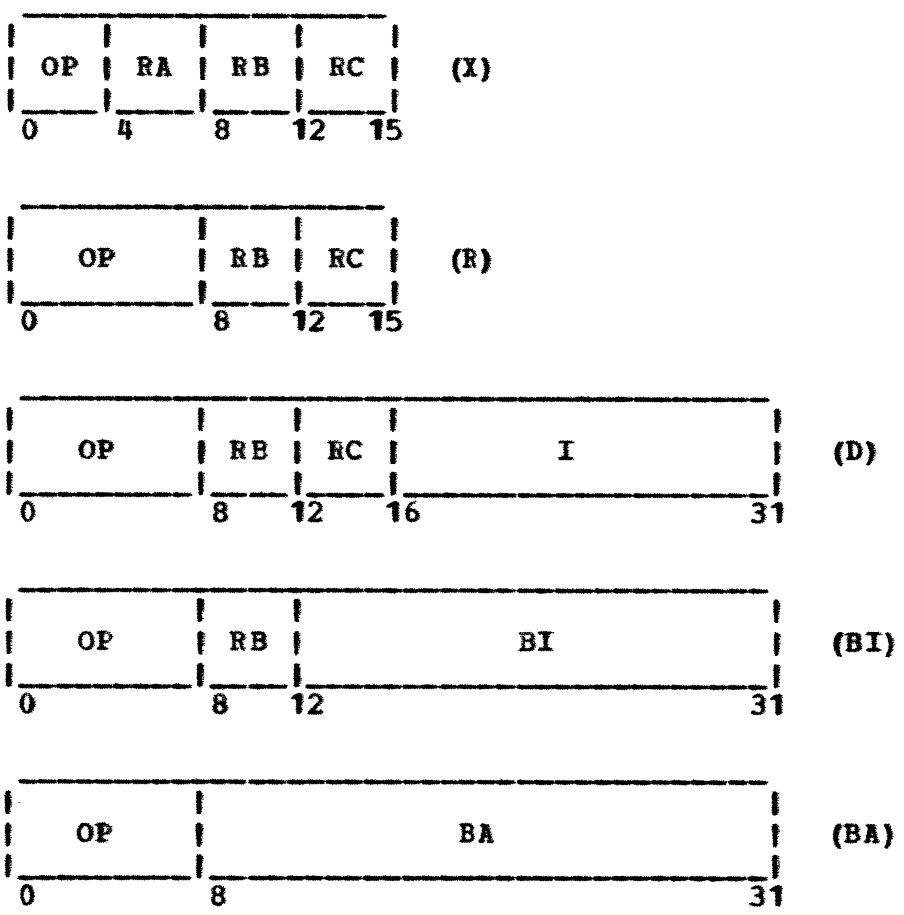


Figure 6 - INSTRUCTION FORMATS

1.4 Interrupts

An interrupt consists of storing the instruction address (IA) of the next instruction in a particular location in main memory and resetting the IA to a fixed value. Each interrupt type has area of 32 bytes for storing the old IA and other information associated with the interrupted state, and an area of 32 bytes (8 words) to which control is transferred when the interrupt occurrs. Both of these areas may or may not be in the data or instruction cache, respectively, when the interupt occurs. The organization of these interrupt areas is shown in Figure 7. The 32 byte areas called NEW are the locations to which control is passed when an interrupt occurrs. Presumably, a branch instruction will be placed in an area. The word labeled OLD IA is the location into which the old IA is stored. The reserved word is saved for possible future use by the hardware to provide more information about the interrupt. The remaining six words are usable by the software.

Normally, an interrupted program can be resumed at the location whose address has been stored as the old IA. However, when errors are being reported, this may not be possible. In general, the attempt will be to suppress the erroneous operation and report its location as the old IA. In some cases, a confused machine state may exist at interrupt. Particular difficulties are caused by branch and execute instructions and by load/store instructions. If errors arise either in the branch and execute or the instruction following it (the subject instruction), the old IA will identify the branch, and software analysis will be required. (It may not be possible to completely recover from a failure in the subject instruction of a branch, link and execute.) Load and store instructions are overlapped with instruction execution, and thus an imprecise interrupt scheme will be needed to report memory failures. The following is a description of each of the interrupt types.

IPL - This interrupt is used to initiate program execution after an IPL. THe processor is disabled after an IPL interrupt.

Machine Error - All processor and I/O machine failures are reported with this interrupt. After the interrupt, the processor is disabled.

Program Error - Program errors are reported with this interrupt. These include:

- Out of range load/store address
- Out of range instruction address
- Undefined operation
- Illegal subject instruction following a branch and

execute.
After the interrupt, the processor retains its former enable/disable status.

Trap - The trap instructions use this interrupt to report a successful comparison. The old IA will be the instruction following the trap. The processor enable/disable state is retained.

External - A signal from the external interrupt adapter while the processor is enabled causes this interrupt. The interrupt will never occur between a branch and execute, and its following instruction, which are treated as a single operation. The processor is disabled after the interrupt. (Note that the processor must have been enabled before the interrupt.)

I/O Check - A time-out or an error in the adapter interface during an I/O read or write will cause the operation to be suppressed and an I/O check interrupt to be taken. The old IA location will contain the address of the I/O instruction which failed. The enable state of the processor is retained after this interrupt.

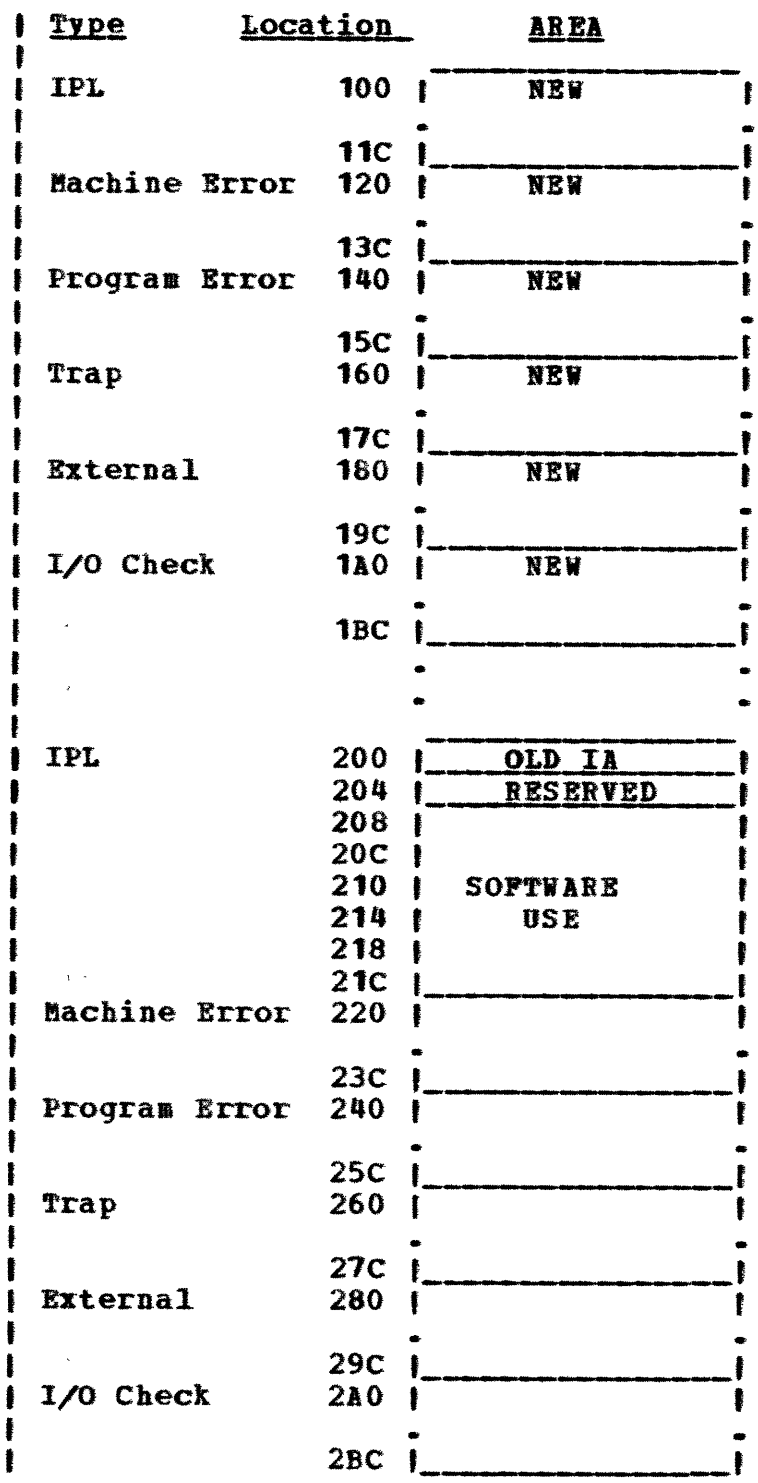


Figure 7. Interrupt Areas

1.5 Input and Output

Input/Output (I/O) operations involve the transfer of information between main storage or the CPU and an I/O adapter. I/O adapters attach I/O defices to the CPU via an I/O Bus which operates at approximately 801 speed. These adapters also connect to a special adapter, called the External Interrupt Adapter, that collects all the interrupt requests from the other adapters, and presents them to the 801 through the external interrupt line. Some high-speed I/O devices also attach through Direct Memory Access (DMA) directly to main memory for the direct transfer of data at high data rates.

A complete description of the I/O structure, program architecture, and functional characteristics, appears in the "801 I/O Subsystem Definition" document.

1.5.1 The External Interrupt Adapter

The interrupt adapter is the common link between all I/O for all interrupts from the I/O adapters to the CPU. It accepts requests for service from the various attached I/O devices, and, when the CPU is enabled for external interruptions, presents them to the CPU in some priority sequence.

The External Interrupt Adapter itself appears to the CPU like an I/O device, with various functions depending upon the particular model of the adapter. It may contain interval timers, real time clocks, device selection and/or masking mechanisms, and other features.

1.5.2 Input/Output Interface

Communications between an adapter and the CPU or memory is under program control. While all adapters, including the External Interrupt Adapter, attach to the common I/O bus, control sequences are, in general, unique to a particular adapter. These sequences, and their responses, are provided through the instructions Input/Output Read and Input/Output Write, which transmit to a specified adapter a 24 bit address/command field, and attempt to accept or transmit 16 bits of data or control information in a specified register. Hence, apart from the commonality of the I/O bus and the interrupt adapter, the interface between each device and the CPU is essentially a programmed interface.

For the Direct Memory Access that attach high-speed I/O defices directly to memory, a given control sequence may initiate the direct transfer of a block of data between main memory and the device. In such circumstances the device

* shares memory with other active directly attached devices
* and the CPU.

2. Storage Access

Storage is organized as a sequence of 32-bit words, each consisting of four 8-bit bytes. Bytes in storage are consecutively numbered, left to right, starting with zero. Each number is considered the address of the corresponding byte.

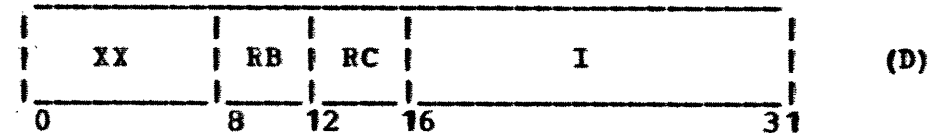
All addresses are computed as byte addresses. Storage addressing wraps around from the maximum byte address, 16,777,215, to address zero. If less than the maximum amount of storage is installed, an attempt to utilize a byte from a non-existent storage location will result in an address exception condition.

All storage accesses are for a word or multiples thereof. Accesses for a register fetch or store the three low-order bytes of a word. Accesses for a half fetch or store the high- or low-order half-word of a word, as required. Accesses for instructions may require the fetching of a word, a half-word, or the low-order half-word of a word followed by the high-order half-word of the next consecutive word in storage. Half-word or word addresses are generated, respectively, by ignoring the low-order one or two bits of a byte address.

If a cache memory for data references is installed, accesses to or from cache to storage occur in multiples of words. Because instruction fetch, storage access, and register access are overlapped in the execution of load and store instructions, interrupts, such as for a bad effective address, may be imprecise.

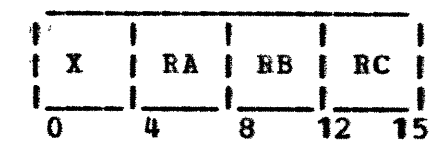
2.1 Instructions

LHAD -- load half algebraic,D-form



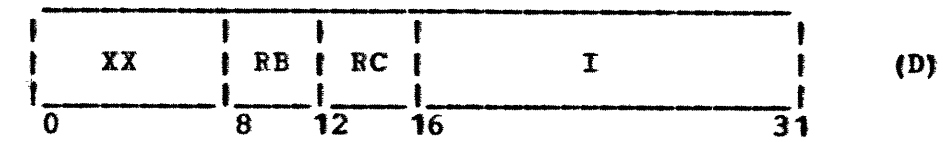
The half (chars C0 and C1) of the register specified by RB is replaced by the half word of storage addressed by 0/(RC) + I. The resulting sign bit is extended through the prefix of register RA. The parity stack in the condition register is pushed down, and the condition register bit SX3 is lost. Condition register bit SX0 assumes the value of the low-order bit of the storage address.

LHAX -- load half algebraic,X-form



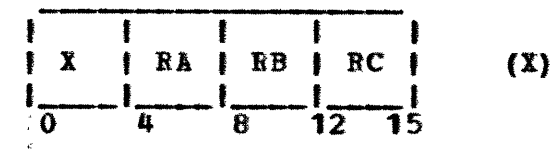
The half (chars C0 and C1) of the register specified by RA is replaced by the half word of storage addressed by (RB) + 0/(RC). The resulting sign bit is extended through the prefix of register RA. The parity stack in the condition register is pushed down, and the condition register bit SX3 is lost. Condition register bit SX0 assumes the value of the low-order bit of the storage address.

LHZD -- load half zero,D-form



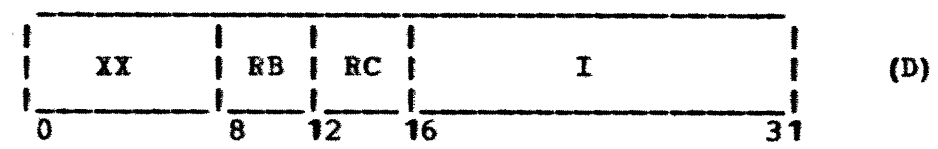
The half (chars C0 and C1) of the register specified by RB is replaced by the half word of storage addressed by 0/(RC) + I. The prefix of register RB is set to zeros. The parity stack in the condition register is pushed down, and the condition register bit SX3 is lost. Condition register bit SX0 assumes the value of the low-order bit of the storage address.

LHZX -- load half zero,X-form



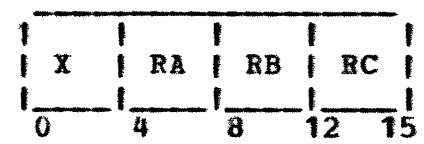
The half (chars C0 and C1) of the register specified by RA is replaced by the half word of storage addressed by (RB) + 0/(RC). The prefix of register RA is set to zeros. The parity stack in the condition register is pushed down, and the condition register bit SX3 is lost. Condition register bit SX0 assumes the value of the low-order bit of the storage address.

| LD -- load, D-form



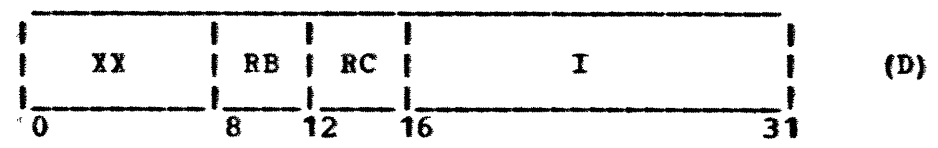
* The content of the register specified by RB is replaced by characters 1,2 and 3 of the word in storage addressed by 0/(RC) + I.
| Note: This instruction does not affect the condition | register in order to be able to preserve the machine state | when processing an interrupt.

| LX -- load, X-form



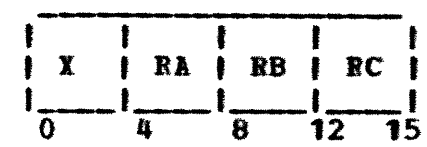
* The content of the register specified by RA is replaced by characters 1,2 and 3 of the word in storage addressed by (RB) + 0/(RC).
| Note: This instruction does not affect the condition | register in order to be able to preserve the machine state | when processing an interrupt.

STCD -- store char,D-form



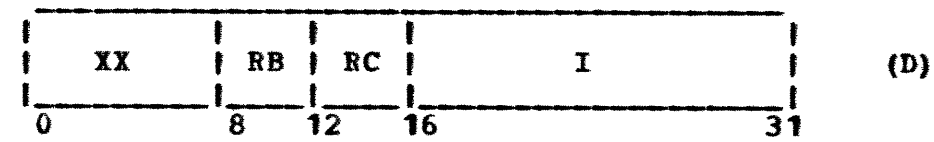
The char of storage addressed by 0/(RC) + I is replaced by char C1 of the register specified by RB.

STCX -- store char,x-form



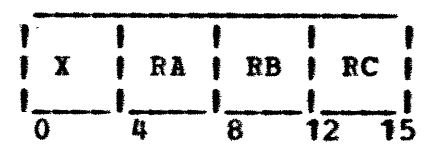
The char of storage addressed by (RB) + 0/(RC) is replaced by char C1 of the register specified by RA.

STHD -- store half,D-form



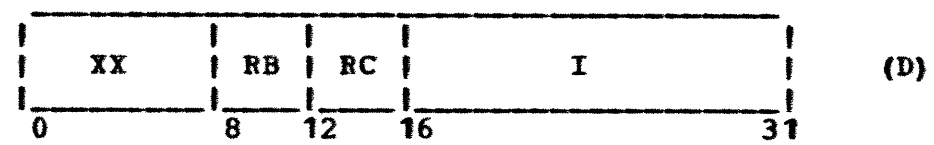
The half word of storage addressed by 0/(RC) + I is replaced by the half of the register specified by RB.

STHX -- store half,X-form



The half word of storage addressed by (RB) + 0/(RC) is replaced by the half of the register specified by RA.

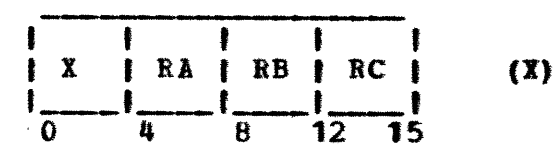
| STD -- store, D-form



Chars 1, 2, and 3 of the word in storage addressed by 0/(RC) + I is replaced by the content of the register specified by RB.

| Note: This instruction does not affect the condition register in order to be able to preserve the machine state when processing an interrupt.

| STX -- store, X-form



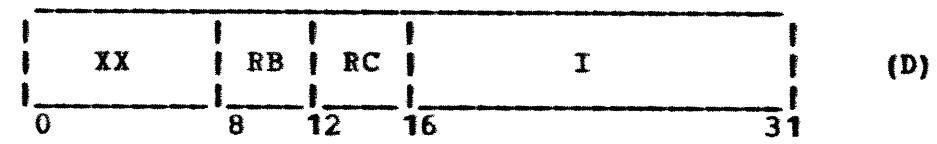
Chars 1, 2, and 3 of the word in storage addressed by (RB) + 0/(RC) is replaced by the content of the register specified by RA.

| Note: This instruction does not affect the condition
| register in order to be able to preserve the machine state
| when processing an interrupt.

3. Address Computation

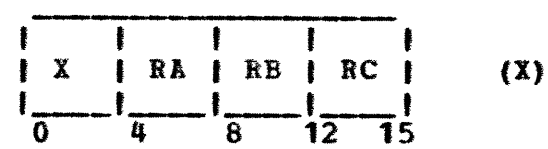
3.1 Instructions

CAD -- compute address,D-form



The address specified by 0/(RC) + I replaces the content of register RB. No storage references for operands occur, and the address is not inspected for address exceptions.

CAX -- compute address,X-form



The address specified by (RB) + 0/(RC) replaces the content of register RA. No storage references for operands occur, and the address is not inspected for address exceptions.

4. Branching

The normal sequential execution of instructions may be changed by the use of the branch instructions in order to perform subroutine linkage, decision making, and loop control.

Subroutine linkage is provided by branch and link instructions:

branch and link absolute (with execute) -- BALA,BALAX
branch and link (with execute), R-form -- BALR,BALRX
branch and link (with execute), I-form -- BALI,BALIX

These instructions permit not only the introduction of a new instruction address, but also preservation of the return address in an implicitly or explicitly designated register. In every case, the new instruction address, the address of the branch target instruction, is computed before the return address is saved. For the regular forms of the instruction, the return address is the address of the byte immediately following the Branch And Link instruction; for the execute forms of the instruction, the return address is the full word boundary on or preceeding the the location six bytes beyond the instruction immediately following the branch and link with execute instruction. In the latter case, the register containing the return address is available to the subject instruction. Note that when 0 is specified as register RC in the R-form Branch And Link instructions, the branch address is taken from register 0. A separate instruction, Move From Instruction Address Register, is provided for obtaining the current instruction address.

Facilities for decision making are provided by the conditional branch instructions:

branch on bit (and execute), I-form -- BB,BBX
branch on bit (and execute), R-form -- BBR,BBRX
branch on not-bit (and execute), I-form -- BNB,BNBX
branch on not-bit (and execute), R-form -- BNBR,BNBRX

These instructions provide the capability of branching or not according to any specified state of any bit of the condition register. Loop control can also be accomplished through use of these instructions to test the outcome of address arithmetic and counting operations.

4.1 Invalid Branch Addresses

If a branch specifies an invalid storage location as the address of the branch target instruction, the address exception condition is not recognized until an attempt is made to execute the branch target instruction.

4.2 Branch With Execute Instructions

For every branch instruction, there is a corresponding branch with execute form of the instruction. The instruction immediately following a branch with execute instruction is called the subject instruction. Whether or not the branch is taken, the subject instruction is executed. The execution of the branch and of the subject instruction is considered as a single unit. If an interrupt occurs at any time during the execution of the branch and its subject instruction, the machine state will be left as if the subject instruction did not execute and the old instruction address will be that of the branch. An interruption during a branch, link, and execute, however, may or may not leave the link address in the specified register.

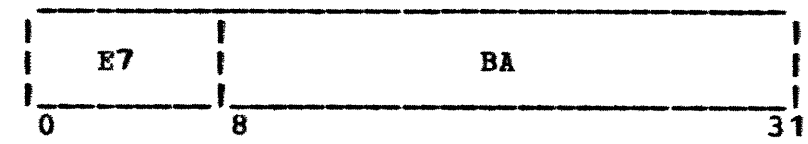
Certain instructions are not allowed to follow a branch and execute instruction. These are branch instructions, trap instructions, cache control instructions, and I/O instructions.

4.3 Instructions

BALA — branch absolute and link

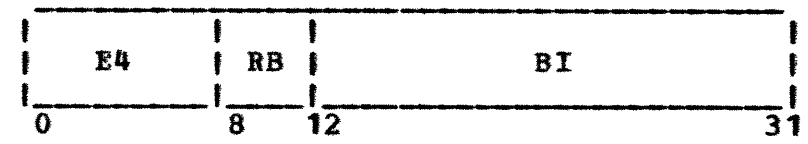
The content of register 15 is replaced by the updated instruction address, and then the updated instruction address is replaced by BA, with its low order bit (bit 23) forced to zero.

BALAX -- branch absolute and link with execute



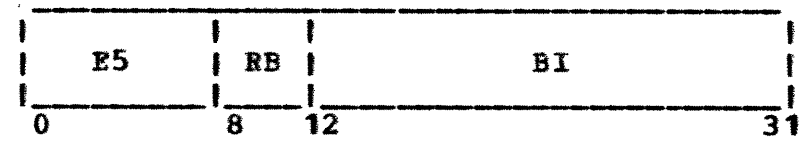
The content of register 15 is replaced by the updated instruction address incremented by six and set to the preceeding full word boundary, the instruction immediately following the branch instruction is executed while the the updated instruction address is replaced by BA, with its low order bit (bit 23) forced to zero.

BALI -- branch and link,I-form



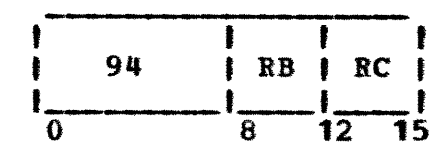
The content of register RB is replaced by the updated instruction address, and then bits 3-22 of the updated instruction address are replaced by BI.

BALIX -- branch and link with execute,I-form



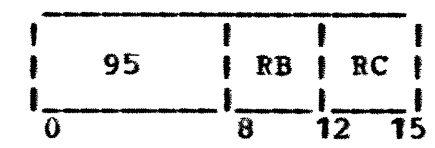
The content of register RB is replaced by the updated instruction address incremented by six and set to the preceeding full word boundary, the instruction immediately following the branch instruction is executed while bits 3-22 of the updated instruction address are replaced by BI.

BALR -- branch and link,R-form



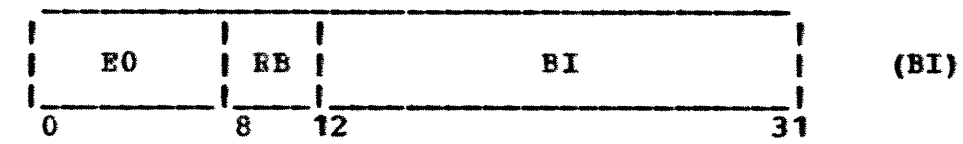
The content of register RB is replaced by the updated instruction address. The updated instruction address is replaced by the content of register RC, with its low-order bit (bit 23) set to zero.

BALRX -- branch and link with execute,R-form



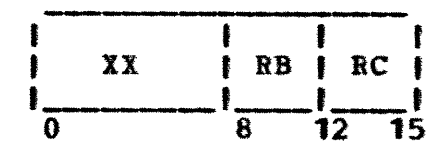
The content of register RB is replaced by the updated instruction address incremented by six and set to the preceeding full word boundary. The instruction immediately following the branch instruction is executed while the the updated instruction address is replaced by the content of register RC, with its low-order bit (bit 23) set to zero.

BB -- branch on bit



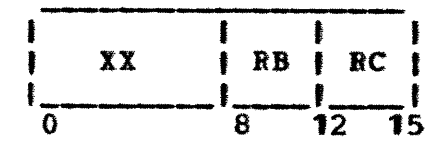
If the condition register bit specified by BB is one, bits 3-22 of the updated instruction address are replaced by BI. If the condition bit is zero, the updated instruction address is unaltered, and no branch occurs.

BBR -- branch on bit, R-form



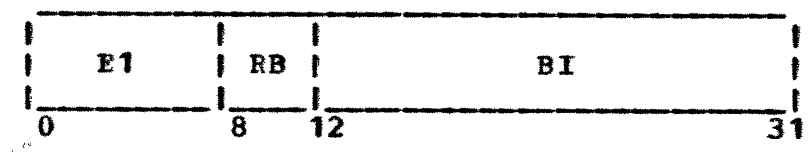
If the condition register bit specified by RB is one, the updated instruction address is replaced by the content of the register specified by RC, and the low-order bit is
* forced to zero. If the condition bit is zero, the updated
* instruction address is unaltered, and no branch occurs.

BBRX -- branch on bit and execute, R-form



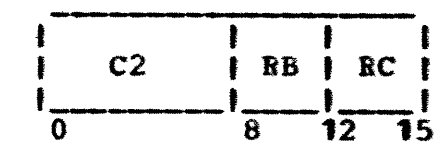
If the condition register bit specified by RB is one, the following instruction is executed while the updated instruction address is replaced by the content of the register specified by RC, with the low-order bit forced to
* zero. If the condition bit is zero, the updated instruction
* address is unaltered, and no branch occurs.

BBX -- branch on bit and execute



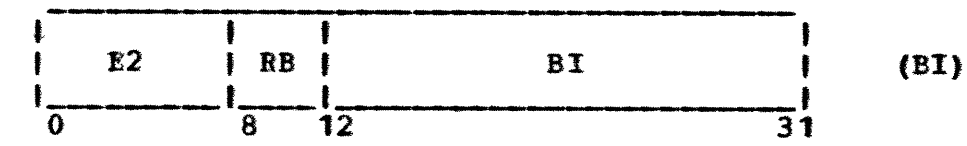
If the condition register bit specified by RB is one, the following instruction is executed while bits 3-22 of the updated instruction address are replaced by BI. If the condition bit is zero, the updated instruction address is unaltered, and no branch occrs.

BEX -- branch,execute and enable



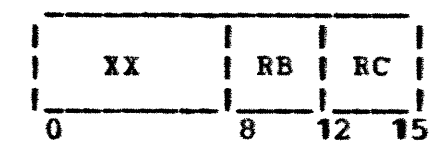
The instruction immediately following the BEX instruction, this following instruction called the subject instruction, is executed while the updated instruction address register is replaced by the content of the register specified by RC, with the low-order bit forced to zero. Upon completion of the subject instruction the machine becomes enabled.

BNB -- branch on not bit



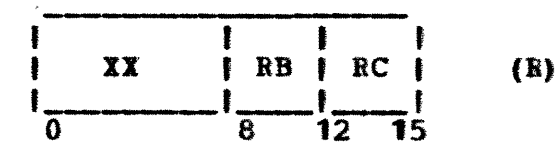
If the condition register bit specified by RB is zero, bits 3-22 of the updated instruction address are replaced by BI. If the condition bit is one, the updated instruction address is unaltered, and no branch occurs.

BNBR -- branch on not-bit, R-form



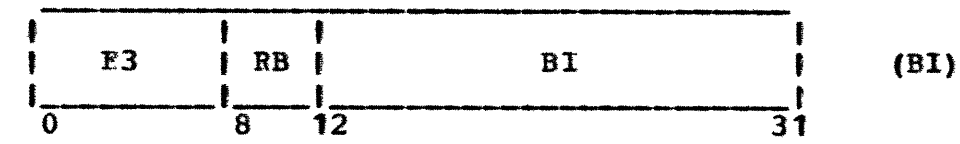
If the condition register bit specified by RB is zero, the updated instruction address is replaced by the content of the register specified by RC, with the low-order bit forced to zero. If the condition bit is one, the updated instruction address is unaltered, and no branch occurs.

BNBRX -- branch on not-bit and execute, R-form



If the condition register bit specified by RB is zero, the following instruction is executed while the updated instruction address is replaced by the content of the register specified by RC, with the low-order bit forced to
* zero. If the condition bit is one, the updated instruction
* address is unaltered, and no branch occurs.

BNBX -- branch on not bit and execute



If the condition register bit specified by RB is zero, the following instruction is executed while bits 3-22 of the updated instruction address are replaced by BI. If the condition bit is one, the updated instruction address is unaltered, and no branch occurs.

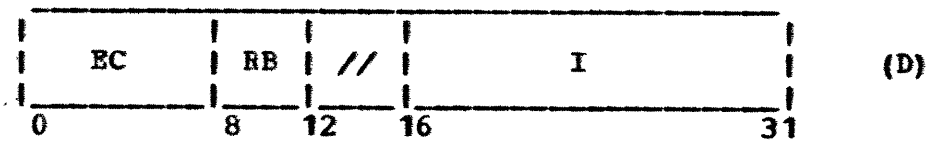
5. Traps

The trap instructions are provided to test for a specified set of conditions. If the conditions tested by a trap instruction are met, a trap exception condition is generated, and an interruption occurs. Control is transfered to the special interrupt area associated with trap interrupts (see section 1.4). If the tested conditions are not met, instruction execution continues with the next sequential instruction. Trap instructions may not appear as the subject instruction of a branch with execute.

The comparisons are performed on operands treated as 24 bit unsigned integers.

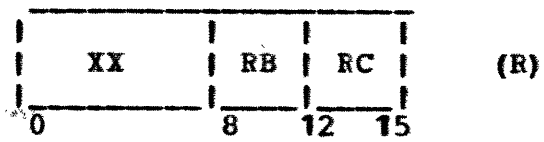
5.1 Instructions

TGTI -- trap if register greater than immed



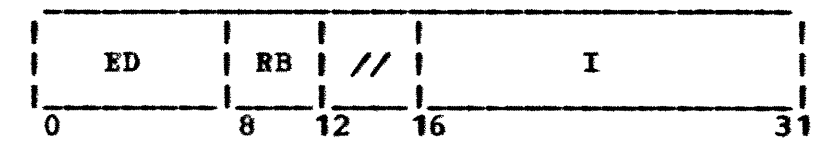
If the content of the register specified by RB is greater than the value of the field I, extended on the left with eight zeros, a trap exception condition is generated.

TLT -- trap if register less than



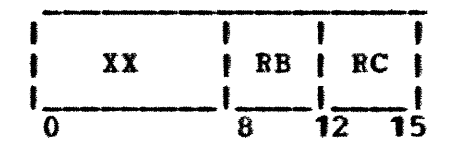
If the content of register RB is less than the content of register RC, a trap exception condition is generated.

TLTI -- trap if register less than immediate



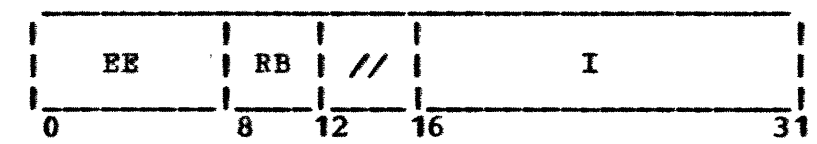
If the content of the register specified by RB is less than the value of the field I, extended on the left with eight zeros, a trap exception condition is generated.

TNE -- trap if register not equal



If the content of register RB is not equal the content of register RC, a trap exception condition is generated.

TNEI -- trap if register not equal immediate



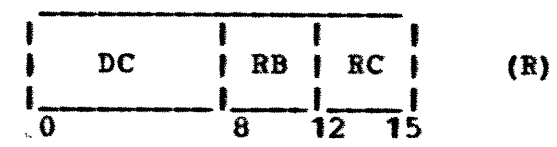
If the content of the register specified by RB is not equal to the value of the field I, extended on the left with eight zeros, a trap exception condition is generated.

6. Moves And Inserts

This group of instructions is concerned solely with the movement of data between registers and with the insertion of data from the immediate field of an instruction into a register. Except when data is moved or inserted into the condition register, none of these instructions alter the condition register or generate exception conditions.

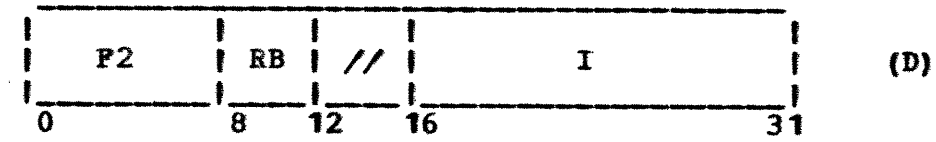
6.1 Instructions

ICBI -- insert condition bit immediate



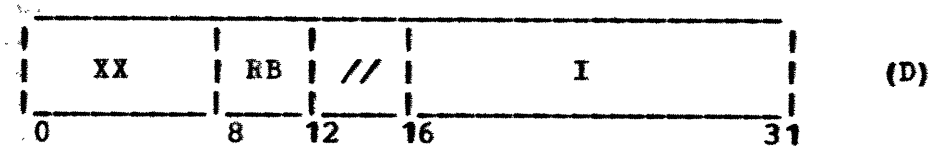
The condition register bit specified by RB is set equal to the low order bit of RC. If the bit specified by RB is a reserved or permanent-zero bit, the value of the bit is unchanged.

IPI -- insert prefix immediate



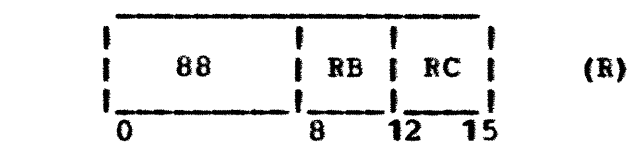
The prefix of the register specified by RB is replaced by the eight low order bits of I (bits 24-31 of the instruction.

IPIZ -- insert prefix immediate and zero



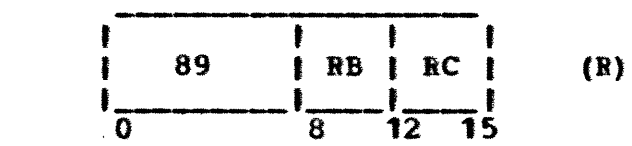
The prefix of the register specified by RB is replaced by the eight low order bits of I (bits 24-31 of the instruction. The half of the register specified by RB is set to zeros.

MC00 -- move character zero from zero



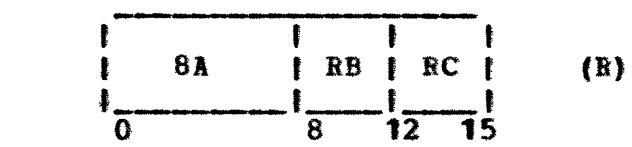
Char C0 of the register specified by RB is replaced by char C0 of the register specified by RC.

MC01 -- move character zero from one



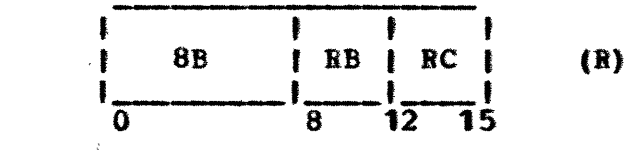
Char C0 of the register specified by RB is replaced by char C1 of the register specified by RC.

MC10 -- move character one from zero



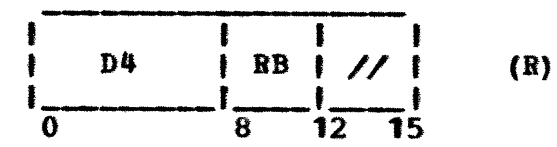
Char C1 of the register specified by RB is replaced by char C0 of the register specified by RC.

MC11 -- move character one from one



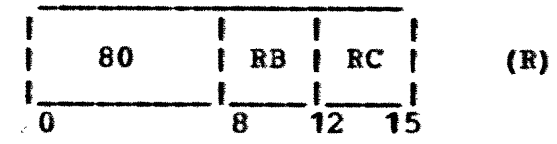
Char C1 of the register specified by RB is replaced by char C1 of the register specified by RC.

MFCR -- move from condition register



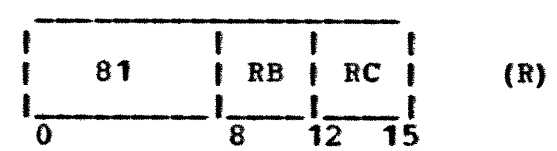
The content of the half of register RB is replaced by the content of the condition register. The prefix of register RB is set to zeros.

MFC0 -- move from character indexed by SX0



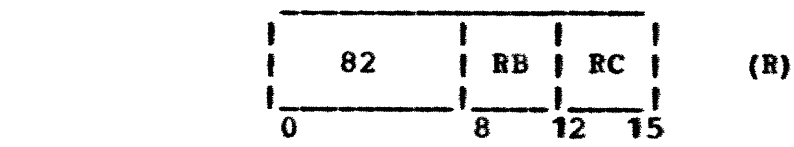
Char C1 of the register specified by RB is replaced by char C0 or char C1 of the register specified by RC, as condition register bit SX0 is, respectively, zero or one. Char C0 of the register specified by RB is set to zero.

MFC1 -- move from character indexed by SX1



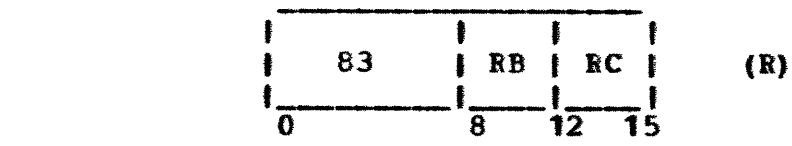
Char C1 of the register specified by RB is replaced by char C0 or char C1 of the register specified by RC, as condition register bit SX1 is, respectively, zero or one. Char C0 of the register specified by RB is set to zero.

MFC2 -- move from character indexed by SX2



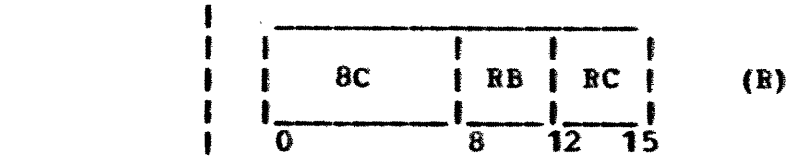
Char C1 of the register specified by RB is replaced by char C0 or char C1 of the register specified by RC, as condition register bit SX2 is, respectively, zero or one. Char C0 of the register specified by RB is set to zero.

MFC3 -- move from character indexed by SX3



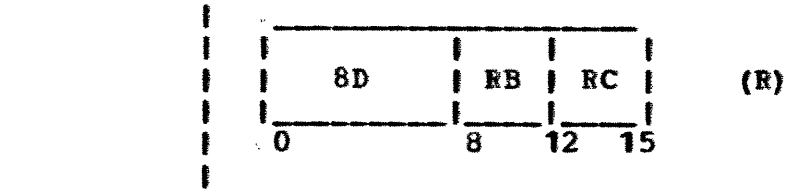
Char C1 of the register specified by RB is replaced by char C0 or char C1 of the register specified by RC, as condition register bit SX3 is, respectively, zero or one. Char C0 of the register specified by RB is set to zero.

MFD -- move from digit



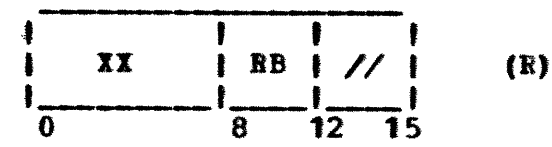
A digit of register RB is selected by bits 22-23 of register RC. This digit is placed in digit D3 of RB and the remainder of RB is set to zero.

MFDP -- move from digit paired



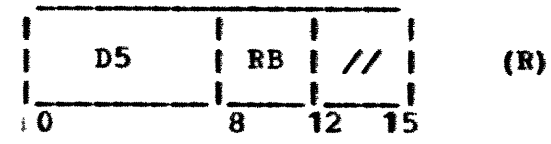
A digit of RB is selected by bits 22-23 of RC. This digit is placed in digit D3 of the twin, in a register pair, of RB and the remainder of the twin is set to zero.

MFIA -- move from instruction address



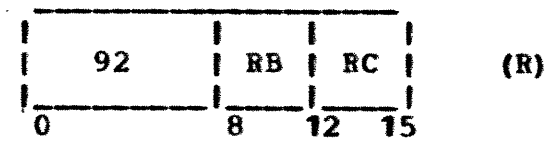
The content of register RB is replaced by the value of the current instruction address, i.e., the location of this instruction.

MFMQ -- move from MQ



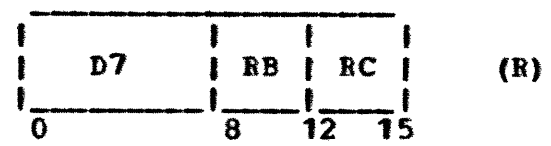
The content of the half of register RB is replaced by the content of the MQ register. The prefix of register RB is set to zeros.

MFP -- move from prefix



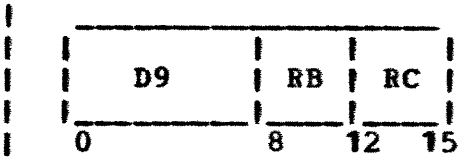
Char C1 of the register specified by RB is replaced by the prefix of the register specified by RC. The prefix and char C0 of the register specified by RB is set to zeros.

MFTB -- move from test bit



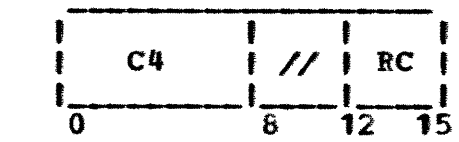
A particular bit of the half of register RB is set to the value of the condition register test bit. The particular bit of the half of register RB is specified by the value of digit D3 of register RC.

MFTBV -- move from test bit value



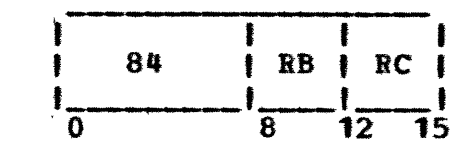
The bit of the half of register RB specified by RC is set to the value of the condition register test bit.

MTCR -- move to condition register



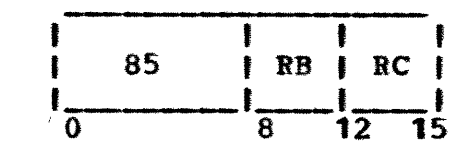
Those bits of the condition register not reserved and/or required to be zero are set to the values of the corresponding bits of the half of register RC.

MTC0 -- move to character indexed by SX0



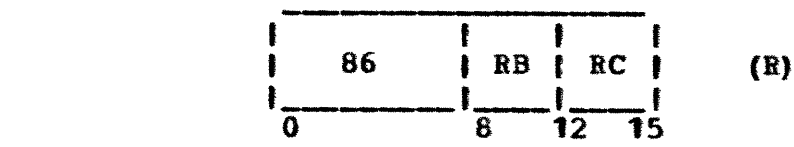
Char C0 or C1 of the register specified by RB is replaced by char C1 of register RC, as condition register bit SX0 is zero or one, respectively.

MTC1 -- move to character indexed by SX1



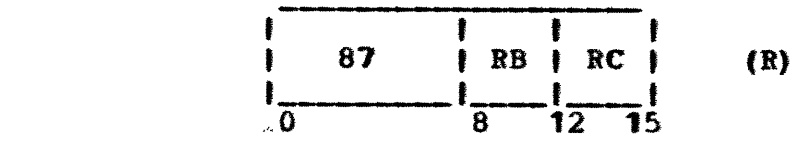
Char C0 or C1 of the register specified by RB is replaced by char C1 of register RC, as condition register bit SX1 is zero or one, respectively.

MTC2 -- move to character indexed by SX2



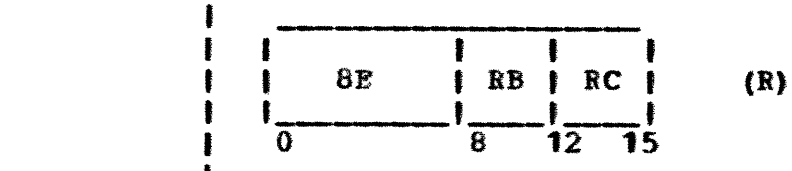
Char C0 or C1 of the register specified by RB is replaced by char C1 of register RC, as condition register bit SX2 is zero or one, respectively.

MTC3 -- move to character indexed by SX3



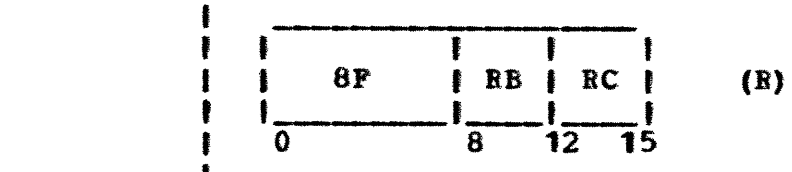
Char C0 or C1 of the register specified by RB is replaced by char C1 of register RC, as condition register bit SX3 is zero or one, respectively.

MTD -- move to digit



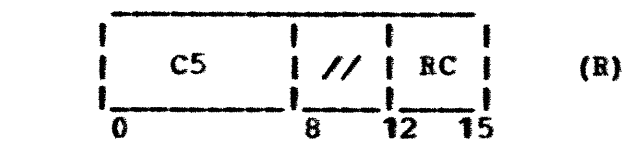
The digit of register RB specified by bits 22-23 of register RC is replaced by digit D3 of register RB.

MTDP -- move to digit paired



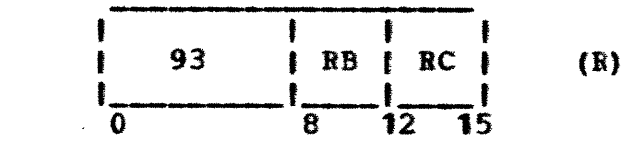
The digit of the twin, in a register pair, of RB selected by bits 22-23 of RC is replaced by digit D3 of register RB.

MTMQ -- move to MQ



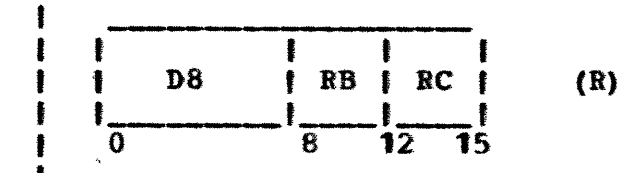
The content of the MQ register is replaced by the half of register RC.

MTP -- move to prefix



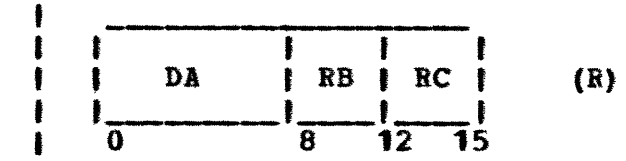
The prefix or the register specified by RB is replaced by char C1 of the register specified by RC.

MTTB -- move to test bit



The condition register test bit is set to the value of a bit of the half of register RB. This bit is specified by digit D3 of register RC.

MTTBV -- move to test bit value



The condition register test bit is set to the value of a bit of the half of register RB, this bit specified by RC.

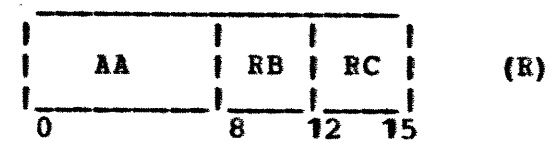
7. Arithmetic

The arithmetic operations, with the exception of the decimal operations Add Decimal and Subtract Decimal, treat registers as consisting of a sign bit followed by 23 integer bits. On these operations, the LT, EQ, and GT bits in the condition register are set to reflect the result: LT is set to one if the result has a negative sign, EQ if the result is zero, and GT if the result has a positive sign and is not zero. Condition register bit C0 is set to reflect the carry out of bit position zero and C1 the carry out of bit position eight. The overflow latch, OV, is set to one if the carry out of bit position one is not equal to the carry out of bit position zero. The summary overflow bit (SO) is set to the OR of the new value of OV with the old value of OV.

The extended operations use the value of the C1 bit to
* determine the result. The extended add instructions, AE and
* AEI, add the value of the C1 bit to the sum of the two operands to determine the result. In the extended subtract
* instructions, SE and SEI, the value of the first operand is added to the complement of the second operand and to this result is added the value of the C1 bit to determine the result.

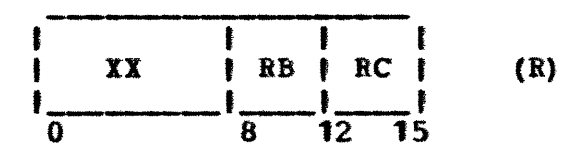
7.1 Instructions

| A -- add



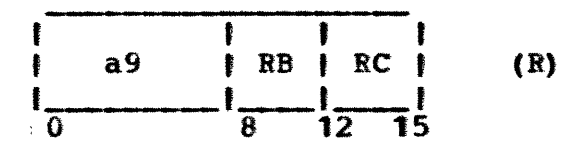
The contents of registers RB and RC are added and the result placed into register RB. Condition bits LT, EQ, GT, C0, C1, OV, and SO are set.

ABS -- absolute value



The content of register RB is replaced by the absolute value of the content of register RC. Condition bits LT, EQ, GT, OV, and SO are set.

AD -- add decimal



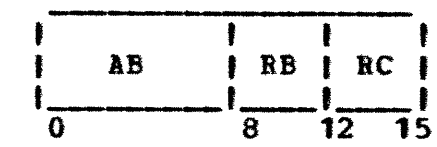
The two-digit decimal number in the low-order byte of the register specified by the second operand, augmented by the value of the condition register C0 bit, is added to the two-digit decimal number in the low-order byte of the register specified by the first operand. The result replaces the low-order byte of the register specified by the first operand, while the high-order bytes of this register remain unaltered.

The condition register bit C0 is set to one if a carry results from the operation, otherwise it is set to zero. The condition register bit EQ is set to one if both digits of the result are zero, otherwise it is set to zero.

If any digit is an invalid decimal digit; i.e. X'A, B, C, D, E, or F', the operation is suppressed, and the condition register decimal-exception-latch, DX, is set to one. Otherwise this latch is set to zero.

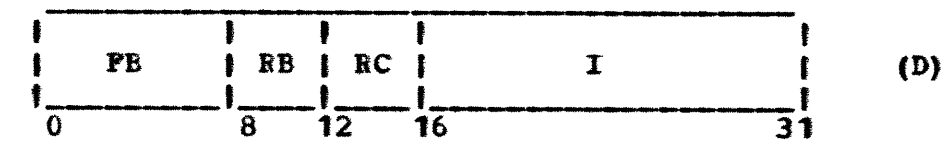
Condition register alterations:
DX, EQ, C0

| AE -- add extended



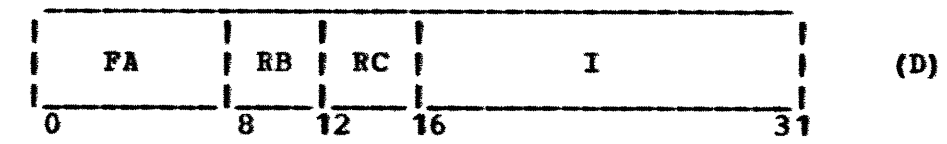
The content of register RB, the content of register RC, and the value of condition bit C1 are summed and the result placed into register RB. Condition bits LT, EQ, GT, C0, C1, OV, and SO are set.

| AEI -- add extended immediate



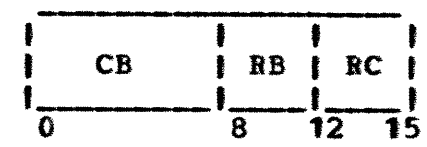
The field I, extended on the left with eight zeros, the contents of register RC, and the value of condition bit C1 are summed and the result placed in register RB. Condition bits LT, EQ, GT, C0, C1, OV, and SO are set.

| AI -- add immediate



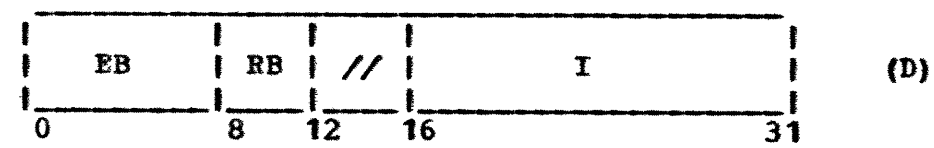
The field I, extended on the left with eight zeros, is added to the contents of register RC and the result placed in register RB. Condition bits LT, EQ, GT, C0, C1, OV, and SO are set.

| C -- compare



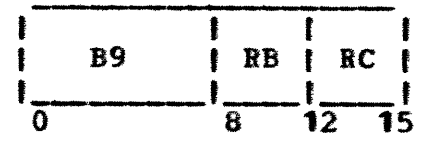
The contents of registers RB and RC, both treated as 24 bit signed integers, are compared. Condition bits LT, EQ, and GT
* are set according to how the value of register RB relates to
* that of register RC.

CI -- compare immediate



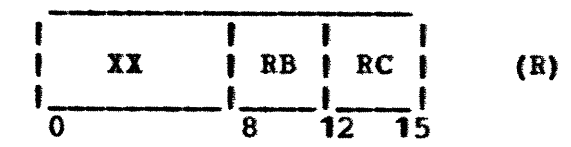
The content of register RB is compared to field I, extended on the left with eight zeros. Condition bits LT, EQ, and GT
* are set according to how the value of register RB relates to
* that of register RC.

DS -- divide step



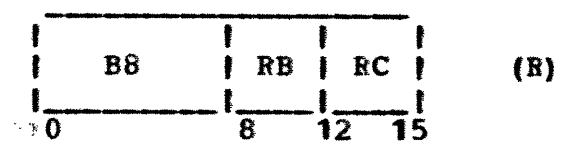
The content of register RC is added to or subtracted from (register RB)||(bit 0 of MQ) depending on whether the signs of registers RB and RC disagree or agree. The 24 low order bits of the sum replace register RB. The MQ is shifted left
* one position and bit 15 of the MQ is set to 1 if and only if S, the sign of the result, equals the sign of register RC. Condition bits CO and OV are set: CO=(RC(bit 0)=S) and OV=(RB(bit 0)=S).
Note: Condition bit SO is unaffected by this instruction.

EXTS -- extend sign



The content of the half of register RB is replaced by the half of register RC. Bits 0 - 7 of register RB are set to equal bit 8. Condition bits LT, EQ, and GT are set.

MS -- multiply step

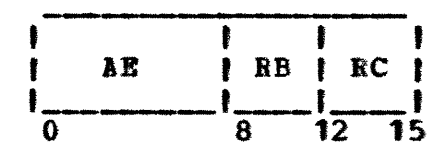


The incomplete product of register RC and bits 14-15 of the MQ are formed in (register RB)||MQ. A 26-bit sum is formed in accordance with the following table:

Condition Bit CO	MQ Bit 14	MQ Bit 15	Algebraic Sum
0	0	0	(RB)+(RC)
0	0	1	(RB)+2(RC)
0	1	0	(RB)-(RC)
0	1	1	(RB)+0
1	0	0	(RB)+0
1	0	1	(RB)+(RC)
1	1	0	(RB)-2(RC)
1	1	1	(RB)-(RC)

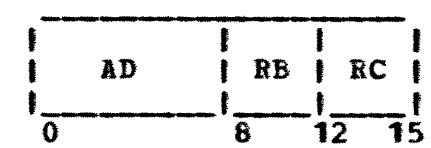
* The MQ is algebraically shifted right two positions, with the two low order bits of the sum replacing bits 0-1 of the MQ. Register RB is replaced with the 24 high order bits of the sum. Condition register bit CO is set to ¬bit 14 of the MQ (before shift).

S -- subtract



The content of register RC is subtracted from the content of register RB and the result placed into register RB. Condition bits LT, EQ, GT, C0, C1, OV, and SO are set.
Note: if RB=RC, the contents of this register is set to zero and bit C0 of the condition register is set to one.

SD -- subtract decimal



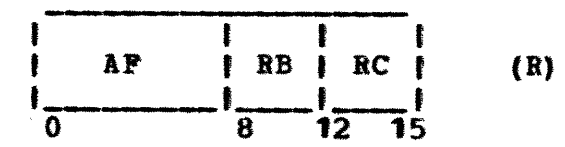
The two-digit decimal number in the low-order byte of the register specified by the second operand, augmented by the inverse of the condition register bit C0, is subtracted from the two-digit decimal number in the low-order byte of the register specified by the first operand. The result replaces the low-order byte of the register specified by the first operand, while the high-order bytes of this register are unaltered.

The condition register bit C0 is set to zero if a 'borrow' was required to perform the subtraction, otherwise it is set to one. The condition register equal-latch EQ is set to zero if both digits of the result are zero, otherwise it is set to one.

If any digit is an invalid decimal digit; i.e. X'A, B, C, D, E, or F', the operation is suppressed, and the condition register decimal-exception-latch DX is set to one. Otherwise this latch is set to zero.

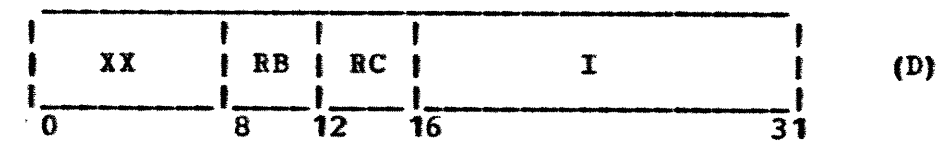
condition register alterations:
DX, EQ, C0

| SE -- subtract extended



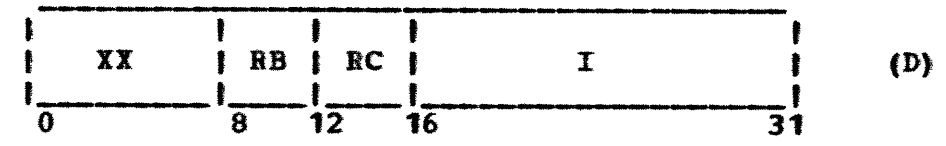
The complement of register RC is added to the content of register RB to which result is added the value of condition bit C1 and the result placed into register RB. Condition bits LT, EQ, GT, C0, C1, OV, and SO are set.

| SEI -- subtract extended immediate



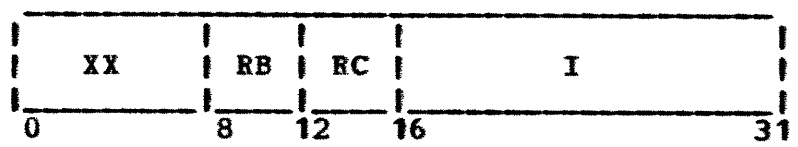
The field I, extended on the left with eight zeros, is comlemented and added to the content of register RC to which result is added the value of condition bit C1 and the result placed in register RB. Condition bits LT, EQ, GT, C0, C1, OV, and SO are set.

| SFI -- subtract from immediate



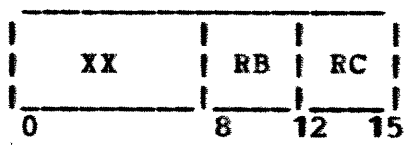
The content of register RB is replaced the by content of the register specified by RC subtracted from I. For the subtraction, I is extended on the left with eight zeros. The condition bits LT, EQ, GT, C0, C1, OV, and SO are set based on the result.

| SI -- subtract immediate



The content of register RB is replaced by I subtracted from the register specified by RC. For the subtraction, I was extended on the left with eight zeros. Condition bits LT, GT, C0, C1, OV, and SO are set based on the result.

TPO -- test prefix for overflow



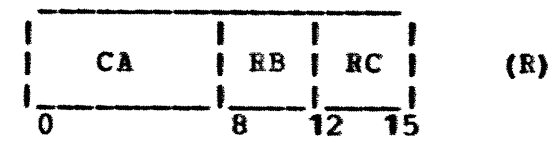
The content of register RB is replaced by the content of register RC. If any of bits 0 - 7 do not equal bit 8 (the sign bit) of register RB, then set OV and SO of the condition register to one; else set OV to zero.

8. Logical Operations

The logical operations treat registers as 24 bit unsigned integers. The exception is the instruction Count Leading Zeros, CLZ, which is applied to the half of a * register, i.e., the 16 low-order bits. The logical * operations that set the LT, EQ, and GT bits of the condition * register according to the result do so according to the * algebraic value of the result. If the result is a negative * value, LT is set to one; if it is zero, EQ is set to one; or * if it is positive and not zero, GT is set to one.

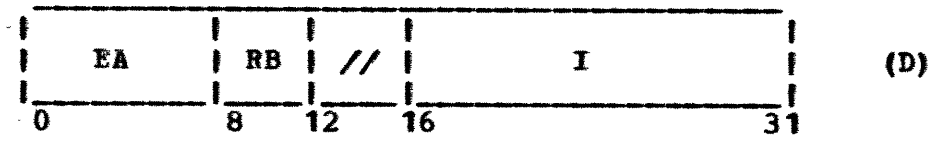
8.1 Instructions

| CL -- compare logical



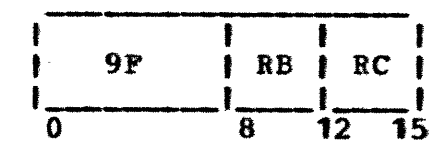
The content of register RB is compared with the content of register RC. Both comparands are treated as 24 bit unsigned quantities. Condition register bits LT, EQ, GT, C0 and C1 * are set. LT, EQ, and GT are set according to how the value * of register RB logically relates to that of register RC. C0 is set to one if the C0 characters in both comparands are equal, else it is set to zero. C1 is set to one if the C1 characters in both comparands are equal, else it is set to zero.

CLI -- compare logical immediate



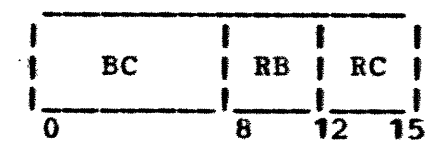
The content of register RB is compared with field I, extended to the left with 8 zeros. Both comparands are treated as 24 bit unsigned quantities. Condition register * bits LT, EQ, GT, C0 and C1 are set. LT, EQ, and GT are set * according to how the value of register RB logically relates * to that of register RC. C0 is set to one if the C0 characters in both comparands are equal, else it is set to zero. C1 is set to one if the C1 characters in both comparands are equal, else it is set to zero.

CLZ -- count leading zeros



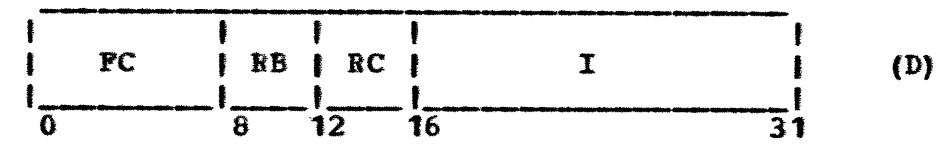
The content of the register specified by RB is replaced by the binary representation of the number of leading zeros in the half of the register specified by RC (i.e. The number of zeros to the left of the left-most one-bit of the half of register RC).

N -- and



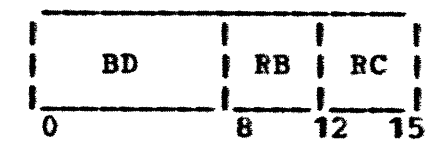
The "and" of the contents of the registers specified by RB and RC replace the content of the register specified by RB. Condition bits LT, EQ, and GT are set according to the result.

NI -- and immediate



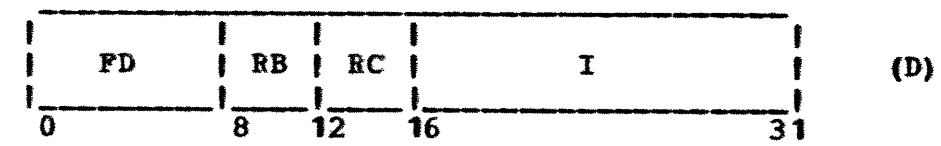
The "and" of field I, extended to the left with 8 zeros, and of the content of register RC replaces the content of register RB. Condition register bits LT, EQ, and GT are set. The connective "and" is applied bit by bit.

| O -- or



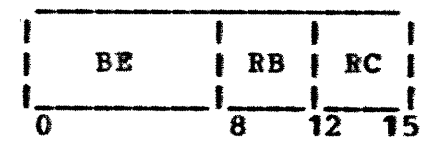
The "or" of the contents of the registers specified by RB and RC replace the content of the register specified by RB. Condition bits LT, EQ, and GT are set according to the result.

OI -- or immediate



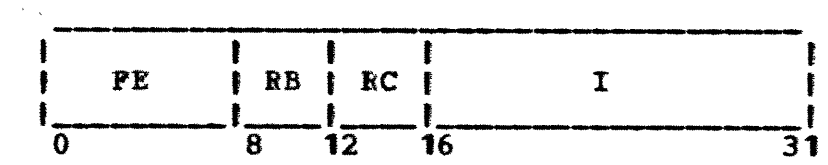
The "or" of field I, extended to the left with 8 zeros, and of the content of register RC replaces the content of register RB. Condition register bits LT, EQ, and GT are set. The connective "or" is applied bit by bit.

| X -- exclusive or



The "exclusive or" of the contents of the registers specified by RB and RC replace the content of the register specified by RB. Condition bits LT, EQ, and GT are set according to the result.

XI -- exclusive or immediate



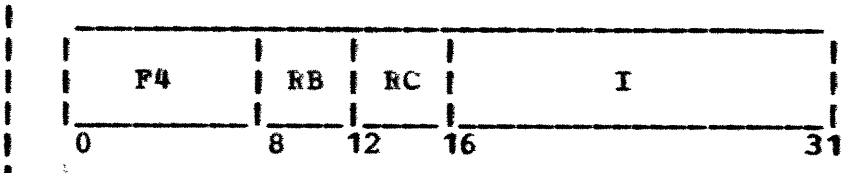
The "exclusive or" of field I, extended to the left with 8 zeros, and of the content of register RC replaces the content of register RB. Condition register bits LT, EQ, and GT are set. The connective exclusive or is applied bit by bit.

9. Shifts

Shift instructions operate either on 24 bits or on 16 bits (a half). The instructions shift a distance of from 0 to 15 bits either left or right. All shifts set the condition register to indicate if the value returned to the register is zero, positive, or negative. All except the algebraic right shift are logical in their treatment of the value shifted. On all left shifts, zeros are supplied to the vacated low order positions.

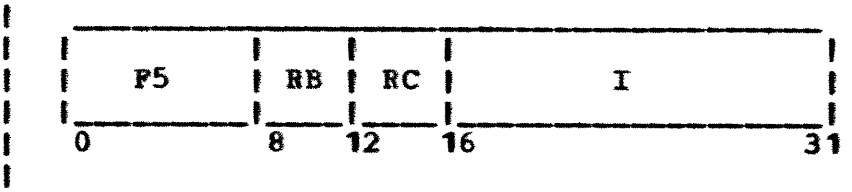
9.1 Instructions

NSLI -- and,then shift left immediate



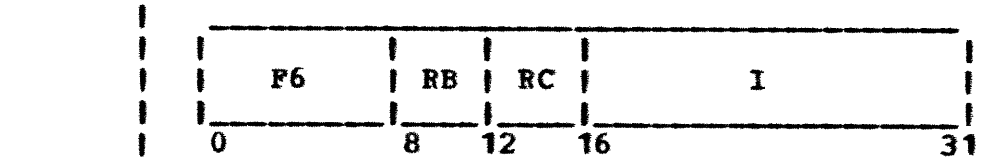
The "and" of field I, extended to the left by eight zeros, and the contents of register RB is shifted left the number of bits specified by RC and replaces the content of register RB. Condition bits LT, EQ, and GT are set.

NSLPI -- and,then shift left paired immediate



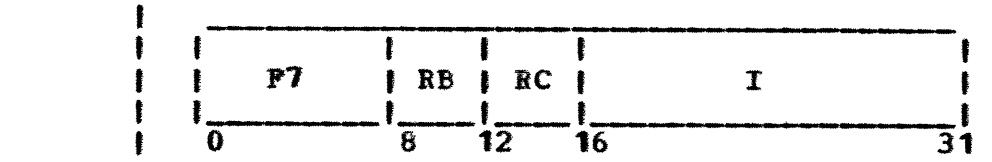
The "and" of field I, extended to the left by eight zeros, and the contents of register RB is shifted left the number of bits specified by RC and replaces the contents of the twin (in a pair) of register RB. Condition bits LT, EQ, and GT are set.

NSRI -- and,then shift right immediate



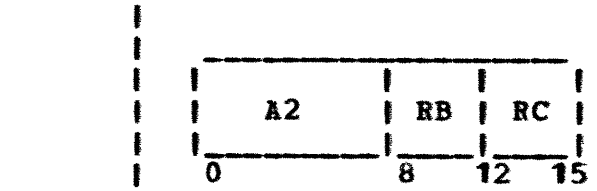
The "and" of field I, extended to the left by eight zeros, and the contents of register RB, shifted right the number of bits specified by RC, replaces the contents of register RB. Condition bits LT, EQ, and GT are set.

NSRPI -- and,then shift right paired immediate



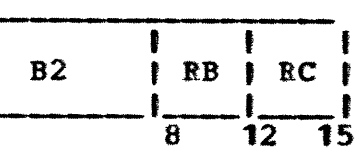
The "and" of field I, extended to the left by eight zeros, and the contents of register RB is shifted right the number of bits specified by RC and replaces the contents of the twin (in a pair) of register RB. Condition bits LT, EQ, and GT are set.

SAR -- shift algebraic right



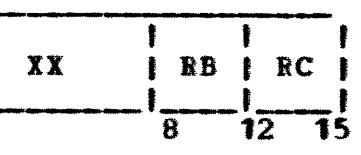
The contents of register RB (bits 0-23) is shifted right the number of bit positions specified by digit D3 of register RC. Bits equal to the original sign bit (bit 0) are supplied to the vacated high order positions. Condition bits LT, EQ, and GT are set.

SARI -- shift algebraic right immediate



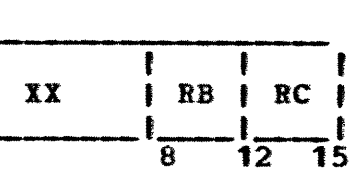
The contents of register RB (bits 0-23) is shifted right the number of positions specified by RC. Bits equal to the original sign bit (bit 0) are supplied to the vacated high order positions. Condition bits LT, EQ, and GT are set.

SHL -- shift half left



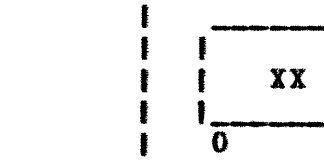
The content of the half of register RB (bits 8-23) is shifted left the number of bit positions specified by digit D3 of register RC. Zeros are supplied to the vacated low order positions. The prefix of the result is set to zero. Condition bits LT, EQ, and GT are set.

SHLI -- shift half left immediate



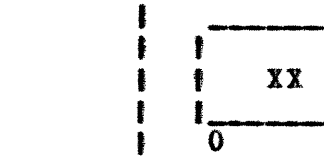
The content of the half of register RB (bits 8-23) is
* shifted left the number of bit positions specified by RC. Zeros are supplied to the vacated low order positions. The prefix of the result is set to zero. Condition bits LT, EQ, and GT are set.

SHLP -- shift half left paired



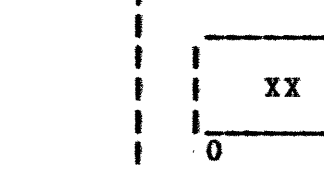
The content of the half of register RB (bits 8-23) is shifted left the number of bit positions specified by digit D3 of register RC. Zeros are supplied to the vacated low order positions. The prefix of the result is set to zero and the result is stored in the twin, in a register pair, of RB. Condition bits LT, EQ, and GT are set.

SHLPI -- shift half left paired immediate



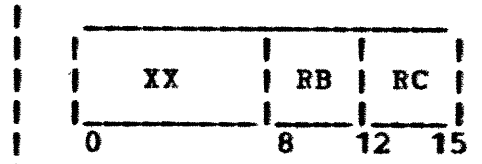
The content of the half of register RB (bits 8-23) is shifted left the number of bit positions specified by RC. Zeros are supplied to the vacated low order positions. The prefix of the result is set to zero and the result is stored in the twin, in a register pair, of RB. Condition bits LT, EQ, and GT are set.

SHR -- shift half right



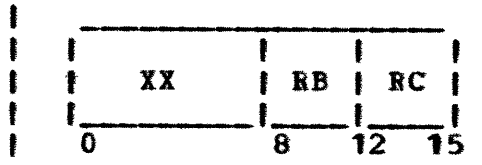
The content of the half of register RB (bits 8-23) is shifted right the number of bit positions specified by digit D3 of register RC. Zeros are supplied to the vacated high order positions. The prefix is set to zero. Condition bits LT, EQ, and GT are set.

* SHRI -- shift half right immediate



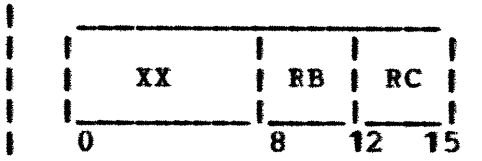
The content of the half of register RB (bits 8-23) is
* shifted right the number of bit positions specified by RC.
Zeros are supplied to the vacated high order positions. The
prefix is set to zero. Condition bits LT, EQ, and GT are
set.

SHRP -- shift half right paired



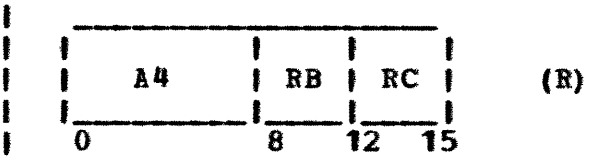
The content of the half of register RB (bits 8-23) is
shifted right the number of bit positions specified by digit
D3 of register RC. Zeros are supplied to the vacated high
order positions. The prefix of the result is set to zero
and the result is stored in the twin, in a register pair, of
RB. Condition bits LT, EQ, and GT are set.

SHRPI -- shift half right paired immediate



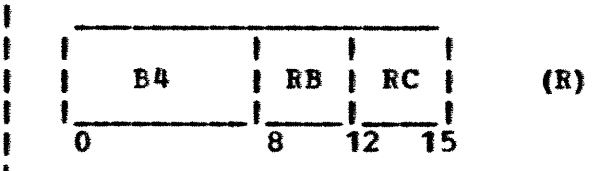
The content of the half of register RB (bits 8-23) is
* shifted right the number of bit positions specified by RC.
Zeros are supplied to the vacated high order positions. The
prefix of the result is set to zero and the result is stored
in the twin, in a register pair, of RB. Condition bits LT,
EQ, and GT are set.

SL -- shift left



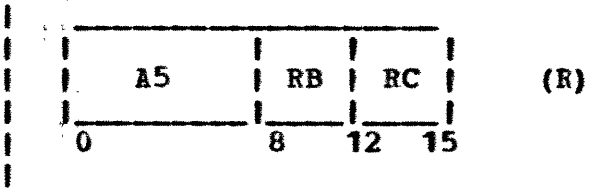
The content of register RB is shifted left the number of bit positions specified by digit D3 of register RC. Zeros are supplied to the vacated low order positions. Condition bits LT, EQ, and GT are set.

SLI -- shift left immediate



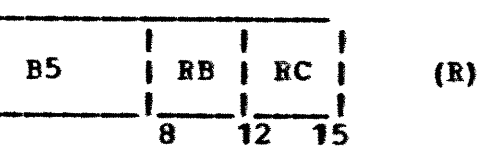
The content of register RB is shifted left the number of bit positions specified by RC. Zeros are supplied to the vacated low order positions. Condition bits LT, EQ, and GT are set.

SLP -- shift left paired



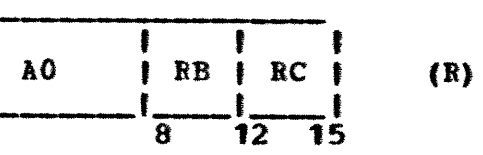
The content of register RB is placed in its twin (in a pair) and the content of this register is shifted left the number of bit positions specified by digit D3 of register RC. Zeros are supplied to the vacated low order positions. Condition bits LT, EQ, and GT are set.

SLPI -- shift left paired immediate



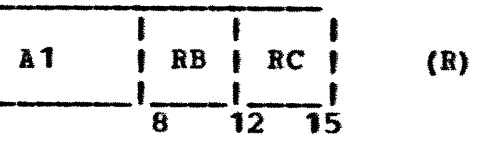
The content of register RB is placed in its twin (in a pair) and the content of this register is shifted left the number of bit positions specified by RC. Zeros are supplied to the vacated low order positions. Condition bits LT, EQ, and GT are set.

SR -- shift right



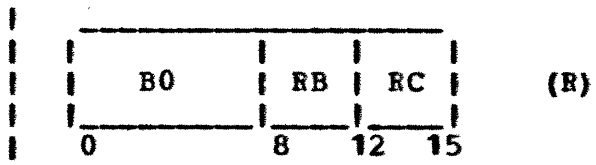
The content of register RB is shifted right the number of bit positions specified by digit D3 of register RC. Zeros are supplied to the vacated high order positions. Condition bits LT, EQ, and GT are set.

SRI -- shift right immediate



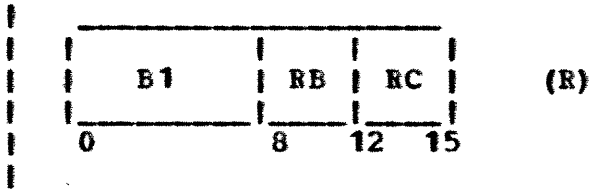
The content of register RB is shifted right the number of bit positions specified by RC. Zeros are supplied to the vacated high order positions. Condition bits LT, EQ, and GT are set.

SRP -- shift right paired



The content of register RB is placed in its twin (in a pair) and the content of this register is shifted right the number of bit positions specified by digit D3 of register RC. Zeros are supplied to the vacated high order positions. Condition bits LT, EQ, and GT are set.

SRPI -- shift right paired immediate



The content of register RB is placed in its twin (in a pair) and the content of this register is shifted right the number of bit positions specified by RC. Zeros are supplied to the vacated high order positions. Condition bits LT, EQ and GT are set.

10. System Control

10.1 Locking

Locking control is provided to support the use of a special disable state of the processor as a lock. In the uniprocessor version, there is exactly one such lock, which is seized by the Lock instruction and released automatically 16 machine instruction executions later. This system lock can be used to protect short critical paths or to protect the implementation of a more complex discipline (such as semaphores) for longer critical regions. The lock has the effect of defering external interrupts independtly of the current enable state of the machine. Thus, during a locked critical path, the machine can be enabled or disabled as necessary. However, external interrupts will be defered. When the lock is released, interrupts will again be controlled by the enable state. A defered interrupt will be taken only if the processor is enabled when the lock is released.

If this mechanism is extended to a multiprocessor system, the use of a single lock to protect all short critical regions may cause excessive blocking. If this is the case, the lock instruction can be provided with a parameter which is interpreted as a critical section or lock ID. Critical sections will then be mutually exclusive only if they are protected by the same lock ID. A given processor can only seize one lock ID at a time. An attempt to seize a second lock (with the same or a different ID) will cause the current lock to be released and the processor enabled for pending interrupts (unless it is in the disabled state independent of the lock mechanism). Only then will the new lock be seized and the processor disabled for a new 16 instruction period.

10.2 Cache Control Operations

The 801 processor is organized to allow independent memory access for data and instructions. Each access path may be served by an independent cache. The effects of these caches on program execution (other than to improve performance) occurrs only in special circumstances. Particularly, modifications to main memory by I/O paths must not be assumed to be reflected to the processor, since the areas affected may already be copied in either or both caches. Modifications to memory by the processor may not be reflected in subsequent instruction or I/O access to main memory, since the updates may be buffered in the data access cache for an indeterminate period of time. This buffereing can affect both read and write I/O accesses. Reads can be changed if a buffered modification to the target area of the read is accomplished after the read has completed. Writes

will transmit the wrong values if buffered modifications have not been accomplished. (It should be stressed that the 801 architecture allows for indeterminately long delays between store instructions and the actual modification of main memory.)

The cache control instructions are provided to allow program control of the relationship between main memory and the caches. These instructions deal with cache lines which are implementation defined. In the current implementation, lines of both data and instructions are 32 bytes long on 32 byte boundaries. The data cache does not attempt to update main memory until a line which has been changed must be removed to make room for a new line.

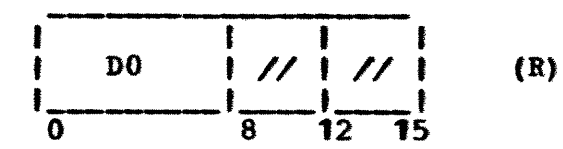
It is likely that details of the cache will be model dependent, and may even be changed in the prototype. Thus all cache control algorithms should be designed and packaged in a way which makes response to such changes reasonably easy. At the very least, the line sizes of the instruction and data caches should be reflected as <u>independent</u> symbolic constants in each routine which issues cache control operations or aligns data on cache line boundaries. (It is probably safe to assume that each line size is a power of two and that the lines are aligned in memory with respect to their own size.)

10.3 Instructions

DI — disable

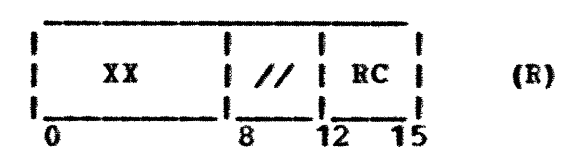
The cpu is disabled for any interruptions due to an external interruption condition.

EI -- enable



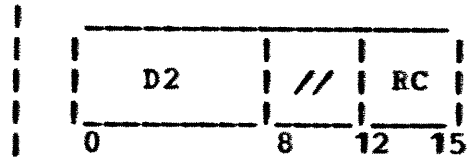
The cpu is enabled for interruptions due to an external interruption condition. Unless a lock is in force, a pending external interrupt will cause an immediate processor interrupt after ENABLE. (Engineering note: Care must be taken that an enable followed immediately by a DISABLE, LOCK, IOR, or IOW instruction works correctly.)

INDCL -- invalidate data cache line



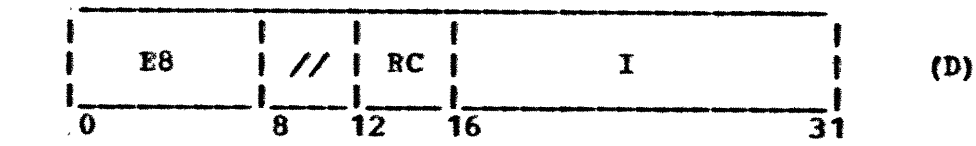
If a previously fetched copy of the line containing the byte addressed by the contents of register RC exists in the data cache, that copy is (logically) replaced by the current value of that line in main storage (no actual fetch from main storage is implied). This instruction can be used after the write into main storage of data from a direct memory attachment, to guarantee that the values received will be the ones seen by subsequent accesses. This instruction can also be used to suppress storing dead temporary values back into main storage from the data cache.

INICL -- invalidate instruction cache line



If the instruction cache has a copy of the line containing the word addressed by the contents of register RC, that copy is abandoned. Thus previous updates to that line which are reflected in main storage (see the Store Data Cache Line instruction) will be reflected in subsequent execution. If the line addressed by this instruction is the line currently addressed by the Instruction Address Register or the line following, the results are unreliable. This instruction can be used in conjunction with the STDCL instruction to synchronize the creation or modification of program text with the subsequent execution of this new program text.

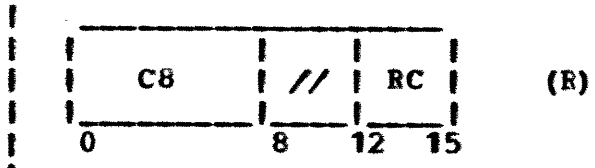
LOCK -- establish lock



If the processor holds a lock it is released. (If the processor is enabled, pending interrupts are taken. The old instruction address (IA) for such interrupts will be the lock instruction). In a uniprocessor, a new locked state is then entered. This lasts for 16 instructions or until another lock instruction is executed within these 16 instruction executions.

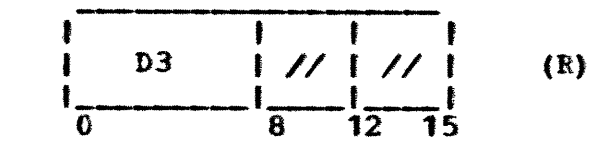
In a multiprocessor, 0/RC + I is interpreted as a 24 bit lock ID. If this lock is available, it is seized and the lock proceeds as in a uniprocessor. If the lock ID is unavailable, the lock instruction is retried. This retry includes accepting pending interrupts if the processor is enabled. The program may assume that the multiprocessor lock manager is fair. (A processor waiting for a lock will eventually get it no matter what other processors are doing.)

STDCL -- store data cache line



If a data cache or other mechanism that buffers or delays stores is installed, it is searched to see if a pending update to the line containing the byte addressed by the contents of register RC exists. If a pending update is found, it is performed. The line may be retained in the cache for later use.

ZNOP -- zero-time no-op

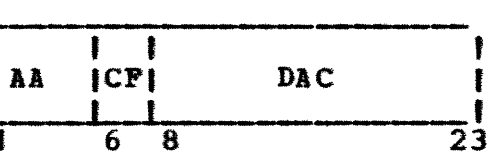


No operation is performed. If the address of this instruction is an odd half-word, and if the instruction is encountered in normal sequential instruction execution (as opposed to being the target of a branch), it appears to be executed in zero time. More precisely, the next following instruction is executed during the CPU cycle that would normally be taken for the execution of this instruction; hence the appearance of zero-time execution.

11. Input/Output Control

* The input/output instructions form a 24 bit
* address/command field by replacing the high order bit of the
* 24 bit sum of I and 0/(RC) with 0 for an IOR or 1 for an
* IOW, and then transfer this command/address field to the
* adapter selected by this field. This field has the following
* format:

*

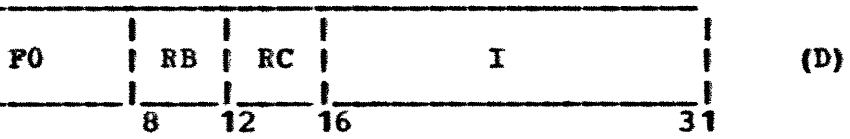


* where bit X represents a read or write, AA is a 5 bit field
* selecting an adapter, and the 2 bit CF field and 16 bit DAC
* field contain control information of a form specific to the
* selected adapter, control unit, and device.

* A complete description of the I/O structure, program
* architecture, and functional characteristics, appears in
* th$e "801 I/O Subsystem Definition" document.

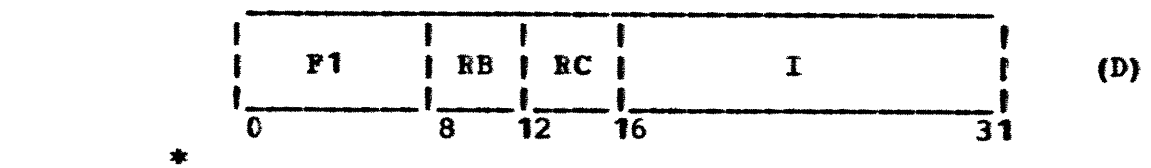
11.1 Instructions

IOR -- input-output read



*
* An attempt is made to transmit 16 bits from the I/O adapter
* selected in the command/address field formed from 0/(RC) + I
* into the half of register RB, setting the prefix of register
* RB to zeros. If the selected adapter can accept the
* command, the I/O Busy bit of the condition register is set
* to zero. If the adapter cannot accept the command, the I/O
* Busy bit is set to one.

IOW -- input-output write



*
* An attempt is made to transmit the half of register RB to
* the I/O adapter selected in the command/address field formed
* from 0/(RC) + I. If the selected adapter can accept the
* command, the I/O Busy bit of the condition register is set
* to zero. If the adapter cannot accept the command, the I/O
* Busy bit is set to one.

CODE	MNE	TYPE	SECTION	INSTRUCTION
0				
1				
2				
3				
4				
5				
6				
7				
80	MFC0	(R)	6:move	move from character indexed by SX0
81	MFC1	(R)	6:move	move from character indexed by SX1
82	MFC2	(R)	6:move	move from character indexed by SX2
83	MFC3	(R)	6:move	move from character indexed by SX3
84	MTC0	(R)	6:move	move to character indexed by SX0
85	MTC1	(R)	6:move	move to character indexed by SX1
86	MTC2	(R)	6:move	move to character indexed by SX2
87	MTC3	(R)	6:move	move to character indexed by SX3
88	MC00	(R)	6:move	move character zero from zero
89	MC01	(R)	6:move	move character zero from one
8A	MC10	(R)	6:move	move character one from zero
8B	MC11	(R)	6:move	move character one from one
8C	MFD	(R)	6:move	move from digit
8D	MFDP	(R)	6:move	move from digit paired
8E	MTD	(R)	6:move	move to digit
8F	MTDP	(R)	6:move	move to digit paired
90				
91				
92	MFP	(R)	6:move	move from prefix
93	MTP	(R)	6:move	move to prefix
94	BALR	(R)	4:brnch	branch and link,R-form
95	BALRX	(R)	4:brnch	branch and link with execute,R-form
96				
97				
98				
99				
9A				
9B				
9C				
9D				
9E				
9F	CLZ	(R)	8:logic	count leading zeros
A0	SR	(R)	9:shift	shift right
A1	SRI	(R)	9:shift	shift right immediate
A2	SAR	(R)	9:shift	shift algebraic right
A3				
A4	SL	(R)	9:shift	shift left
A5	SLP	(R)	9:shift	shift left paired
A6				
A7				
A8				
A9	AD	(R)	7:arith	add decimal
AA	A	(R)	7:arith	add

AB	AE	(R)	7:arith	add extended
AC				
AD	SD	(R)	7:arith	subtract decimal
AE	S	(R)	7:arith	subtract
AF	SE	(R)	7:arith	subtract extended
B0	SRP	(R)	9:shift	shift right paired
B1	SRPI	(R)	9:shift	shift right paired immediate
B2	SARI	(R)	9:shift	shift algebraic right immediate
B3				
B4	SLI	(R)	9:shift	shift left immediate
B5	SLPI	(R)	9:shift	shift left paired immediate
B6				
B7				
B8	MS	(R)	7:arith	multiply step
B9	DS	(R)	7:arith	divide step
BA				
BB				
BC	N	(R)	8:logic	and
BD	O	(R)	8:logic	or
BE	X	(R)	8:logic	exclusive or
BF				
C0				
C1				
C2	BEX	(R)	4:brnch	branch,execute and enable
C3				
C4	MTCR	(R)	6:move	move to condition
C5	MTMQ	(R)	6:move	move to MQ
C6				
C7				
C8	STDCL	(R)	10:sys	store data cache line
C9				
CA	CL	(R)	8:logic	compare logical
CB	C	(R)	7:arith	compare
CC				
CD				
CE				
CF				
D0	EI	(R)	10:sys	enable
D1	DI	(R)	10:sys	disable
D2	INICL	(R)	10:sys	invalidate instruction cache line
D3	ZNOP	(R)	10:sys	zero-time no-op
D4	MFCR	(R)	6:move	move from condition
D5	MFMQ	(R)	6:move	move from MQ
D6				
D7	MFTB	(R)	6:move	move from test bit
D8	MTTB	(R)	6:move	move to test bit
D9	MFTBV	(R)	6:move	move from test bit value
DA	MTTBV	(R)	6:move	move to test bit value
DB				
DC	ICBI	(R)	6:move	insert condition bit immediate
DD				
DE				
DF				
E0	BB	(BI)	4:brnch	branch on bit

E1	BBX	(BI)	4:brnch	branch on bit and execute
E2	BNB	(BI)	4:brnch	branch on not bit
E3	BNBX	(BI)	4:brnch	branch on not bit and execute
E4	BALI	(BI)	4:brnch	branch and link,I-form
E5	BALIX	(BI)	4:brnch	branch and link with execute,I-form
E6	BALA	(BA)	4:brnch	branch and link absolute
E7	BALAX	(BA)	4:brnch	branch and link absolute with execut
E8	LOCK	(D)	10:sys	establish lock
E9				
EA	CLI	(RI)	8:logic	compare logical immediate
EB	CI	(RI)	7:arith	compare immediate
EC	TGTI	(RI)	5:trap	trap if greater than immed
ED	TLTI	(RI)	5:trap	trap if less than immediate
EE	TNEI	(RI)	5:trap	trap if not equal immediate
EF				
F0	IOR	(D)	11:i/o	input-output read
F1	IOW	(D)	11:i/o	input-output write
F2	IPI	(RI)	6:move	insert prefix immediate
F3				
F4	NSLI	(D)	9:shift	and,then shift left immediate
F5	NSLPI	(D)	9:shift	and,then shift left paired immediate
F6	NSRI	(D)	9:shift	and,then shift right immediate
F7	NSRPI	(D)	9:shift	and,then shift right paired immediate
F8				
F9				
FA	AI	(D)	7:arith	add immediate
FB	AEI	(D)	7:arith	add extended immediate
FC	NI	(D)	8:logic	and immediate
FD	OI	(D)	8:logic	or immediate
FE	XI	(D)	8:logic	exclusive or immediate
FF				
X	CAX	(X)	3:adres	compute address,X-form
X	LHAX	(X)	2:strge	load half algebraic,X-form
X	LHZX	(X)	2:strge	load half zero,X-form
X	LX	(X)	2:strge	load, X-form
X	STCX	(X)	2:strge	store char,X-form
X	STHX	(X)	2:strge	store half,X-form
X	STX	(X)	2:strge	store,X-form
XX	ABS	(R)	7:arith	absolute value
XX	BBR	(R)	4:brnch	branch on bit, R-form
XX	BNBR	(R)	4:brnch	branch on not-bit, R-form
XX	BBRX	(R)	4:brnch	branch on bit and execute, R-form
XX	BNBRX	(R)	4:brnch	branch on not-bit and execute, r-for
XX	CAD	(D)	3:adres	compute address,D-form
XX	EXTS	(R)	7:arith	extend sign
XX	INDCL	(R)	10:sys	invalidate data cache line
XX	IPIZ	(RI)	6:move	insert prefix immediate and zero
XX	LD	(D)	2:strge	load,D-form
XX	LHAD	(D)	2:strge	load half algebraic,D-form
XX	LHZD	(D)	2:strge	load half zero,D-form
XX	MFIA	(R)	6:move	move from instruction address
XX	SFI	(D)	7:arith	subtract from immediate
XX	SHL	(R)	9:shift	shift half left
XX	SHLI	(R)	9:shift	shift half left immediate

XX	SHLP	(R)	9:shift	shift half left paired
XX	SHLPI	(R)	9:shift	shift half left paired immediate
XX	SHR	(R)	9:shift	shift half right
XX	SHRI	(R)	9:shift	shift half right immediate
XX	SHRP	(R)	9:shift	shift half right paired
XX	SHRPI	(R)	9:shift	shift half right paired immediate
XX	SEI	(D)	7:arith	subtract extended immediate
XX	SI	(D)	7:arith	subtract immediate
XX	STCD	(D)	2:strge	store char,D-form
XX	STD	(D)	2:strge	store,D-form
XX	STHD	(D)	2:strge	store half,D-form
XX	TLT	(R)	5:trap	trap if less than
XX	TNE	(R)	5:trap	trap if not equal
XX	TPO	(R)	7:arith	test prefix for overflow

MNE	CODE	TYPE	SECTION	INSTRUCTION
A	AA	(R)	7:arith	add
ABS	XX	(R)	7:arith	absolute value
AD	A9	(R)	7:arith	add decimal
AE	AB	(R)	7:arith	add extended
AEI	FB	(D)	7:arith	add extended immediate
AI	FA	(D)	7:arith	add immediate
BALA	E6	(BA)	4:brnch	branch and link absolute
BALAX	E7	(BA)	4:brnch	branch and link absolute with execut
BALI	E4	(BI)	4:brnch	branch and link,I-form
BALIX	E5	(BI)	4:brnch	branch and link with execute,I-form
BALR	94	(R)	4:brnch	branch and link,R-form
BALRX	95	(R)	4:brnch	branch and link with execute,R-form
BB	E0	(BI)	4:brnch	branch on bit
BBR	XX	(R)	4:brnch	branch on bit, R-form
BBRX	XX	(R)	4:brnch	branch on bit and execute, R-form
BBX	E1	(BI)	4:brnch	branch on bit and execute
BEX	C2	(R)	4:brnch	branch,execute and enable
BNB	E2	(BI)	4:brnch	branch on not bit
BNBR	XX	(R)	4:brnch	branch on not-bit, R-form
BNBRX	XX	(R)	4:brnch	branch on not-bit and execute, r-for
BNBX	E3	(BI)	4:brnch	branch on not bit and execute
C	CB	(R)	7:arith	compare
CAD	XX	(D)	3:adres	compute address,D-form
CAX	X	(X)	3:adres	compute address,X-form
CI	EB	(RI)	7:arith	compare immediate
CL	CA	(R)	8:logic	compare logical
CLI	EA	(RI)	8:logic	compare logical immediate
CLZ	9F	(R)	8:logic	count leading zeros
DI	D1	(R)	10:sys	disable
DS	B9	(R)	7:arith	divide step
EI	D0	(R)	10:sys	enable
EXTS	XX	(R)	7:arith	extend sign
ICBI	DC	(R)	6:move	insert condition bit immediate
INDCL	XX	(R)	10:sys	invalidate data cache line
INICL	D2	(R)	10:sys	invalidate instruction cache line
IOR	F0	(D)	11:i/o	input-output read
IOW	F1	(D)	11:i/o	input-output write
IPI	F2	(RI)	6:move	insert prefix immediate
IPIZ	XX	(RI)	6:move	insert prefix immediate and zero
LD	XX	(D)	2:strge	load,D-form
LHAD	XX	(D)	2:strge	load half algebraic,D-form
LHAX	X	(X)	2:strge	load half algebraic,X-form
LHZD	XX	(D)	2:strge	load half zero,D-form
LHZX	X	(X)	2:strge	load half zero,X-form
LOCK	E8	(D)	10:sys	establish lock
LX	X	(X)	2:strge	load,X-form
MC00	88	(R)	6:move	move character zero from zero
MC01	89	(R)	6:move	move character zero from one
MC10	8A	(R)	6:move	move character one from zero
MC11	8B	(R)	6:move	move character one from one
MFCR	D4	(R)	6:move	move from condition

MFC0	80	(R)	6:move	move from character indexed by SX0
MFC1	81	(R)	6:move	move from character indexed by SX1
MFC2	82	(R)	6:move	move from character indexed by SX2
MFC3	83	(R)	6:move	move from character indexed by SX3
MFD	8C	(R)	6:move	move from digit
MFDP	8D	(R)	6:move	move from digit paired
MFIA	XX	(R)	6:move	move from instruction address
MFMQ	D5	(R)	6:move	move from MQ
MFP	92	(R)	6:move	move from prefix
MFTBV	D9	(R)	6:move	move from test bit value
MFTB	D7	(R)	6:move	move from test bit
MS	B8	(R)	7:arith	multiply step
MTCR	C4	(R)	6:move	move to condition
MTC0	84	(R)	6:move	move to character indexed by SX0
MTC1	85	(R)	6:move	move to character indexed by SX1
MTC2	86	(R)	6:move	move to character indexed by SX2
MTC3	87	(R)	6:move	move to character indexed by SX3
MTD	8E	(R)	6:move	move to digit
MTDP	8F	(R)	6:move	move to digit paired
MTMQ	C5	(R)	6:move	move to MQ
MTP	93	(R)	6:move	move to prefix
MTTBV	DA	(R)	6:move	move to test bit value
MTTB	D8	(R)	6:move	move to test bit
N	BC	(R)	8:logic	and
NI	FC	(D)	8:logic	and immediate
NSLI	F4	(D)	9:shift	and,then shift left immediate
NSLPI	F5	(D)	9:shift	and,then shift left paired immediate
NSRI	F6	(D)	9:shift	and,then shift right immediate
NSRPI	F7	(D)	9:shift	and,then shift right paired immediate
O	BD	(R)	8:logic	or
OI	FD	(D)	8:logic	or immediate
S	AE	(R)	7:arith	subtract
SAR	A2	(R)	9:shift	shift algebraic right
SARI	B2	(R)	9:shift	shift algebraic algebraic immediate
SD	AD	(R)	7:arith	subtract decimal
SE	AF	(R)	7:arith	subtract extended
SEI	XX	(D)	7:arith	subtract extended immediate
SFI	XX	(D)	7:arith	subtract from immediate
SHL	XX	(R)	9:shift	shift half left
SHLI	XX	(R)	9:shift	shift half left immediate
SHLP	XX	(R)	9:shift	shift half left paired
SHLPI	XX	(R)	9:shift	shift half left paired immediate
SHR	XX	(R)	9:shift	shift half right
SHRI	XX	(R)	9:shift	shift half right immediate
SHRP	XX	(R)	9:shift	shift half right paired
SHRPI	XX	(R)	9:shift	shift half right paired immediate
SI	XX	(D)	7:arith	subtract immediate
SL	A4	(R)	9:shift	shift left
SLI	B4	(R)	9:shift	shift left immediate
SLP	A5	(R)	9:shift	shift left paired
SLPI	B5	(R)	9:shift	shift left paired immediate
SR	A0	(R)	9:shift	shift right
SRI	A1	(R)	9:shift	shift right immediate
SRP	B0	(R)	9:shift	shift right paired

SRPI	B1	(R)	9:shift	shift right paired immediate
STCD	XX	(D)	2:strge	store char,D-form
STCX	X	(X)	2:strge	store char,X-form
STD	XX	(D)	2:strge	store,D-form
STDCL	C8	(R)	10:sys	store data cache line
STHD	XX	(D)	2:strge	store half,D-form
STHX	X	(X)	2:strge	store half,X-form
STX	X	(X)	2:strge	store,X-form
TGTI	EC	(RI)	5:trap	trap if greater than immed
TLT	XX	(R)	5:trap	trap if less than
TLTI	ED	(RI)	5:trap	trap if less than immediate
TNE	XX	(R)	5:trap	trap if not equal
TNEI	EE	(RI)	5:trap	trap if not equal immediate
TPO	XX	(R)	7:arith	test prefix for overflow
XI	FE	(D)	8:logic	exclusive or immediate
X	BE	(R)	8:logic	exclusive or
ZNOP	D3	(R)	10:sys	zero-time no-op