Wang 2200 Instruction Set

Computer Concepts Corporation
8001 W. 63rd Street
Shawnee Mission, Ks. 66202

Revised 4/26/1983

1.0 Overall Description of the Wang 2200

The Wang 2200 series computers are fast and powerful minicomputers specializing in the Basic language. The structure of the machine is geared around the language, and as such, can outperform almost all mainframes on the market today.

A common Input/Output bus is utilized, transferring at rates of about 200K bytes per second. Two separate memories, one for Data, and the other for Control are implemented. This permits the separation of Control Memory (Where the Basic, Cobol or Diagnostics are stored) from the User memory.

The actual processing section utilizes a 24 bit wide control word and numerous internal registers to perform the operations required to execute Basic code.

1.1 Internal Register structure - General Description

The Wang 2200 system contains eight (8) general purpose registers labeled R0 thru R7. Each of the general registers are eight (8) bits wide. These registers are used to hold temporary data, statuses of searches, math operands, etc..

Two registers are normally combined to form a pointer for Data memory operations. These registers are the PH and PL registers. The PH and PL registers are each eight (8) bits wide, but can be accessed as a 16 bit register by some instructions. 65536 memory locations can be accessed by these registers.

Another set of registers is available for storing data read from Data memory. These registers are called CH and CL. Data read from Data memory is 16 bits in width. The upper byte is stored in CH while the lower byte goes to CL.

To write to Data memory, another register, called the DUM is employed. The DUM register is only eight bits wide. Therefore, only one byte can be written at a time. Control bits are available to write to either the High or Low bytes.

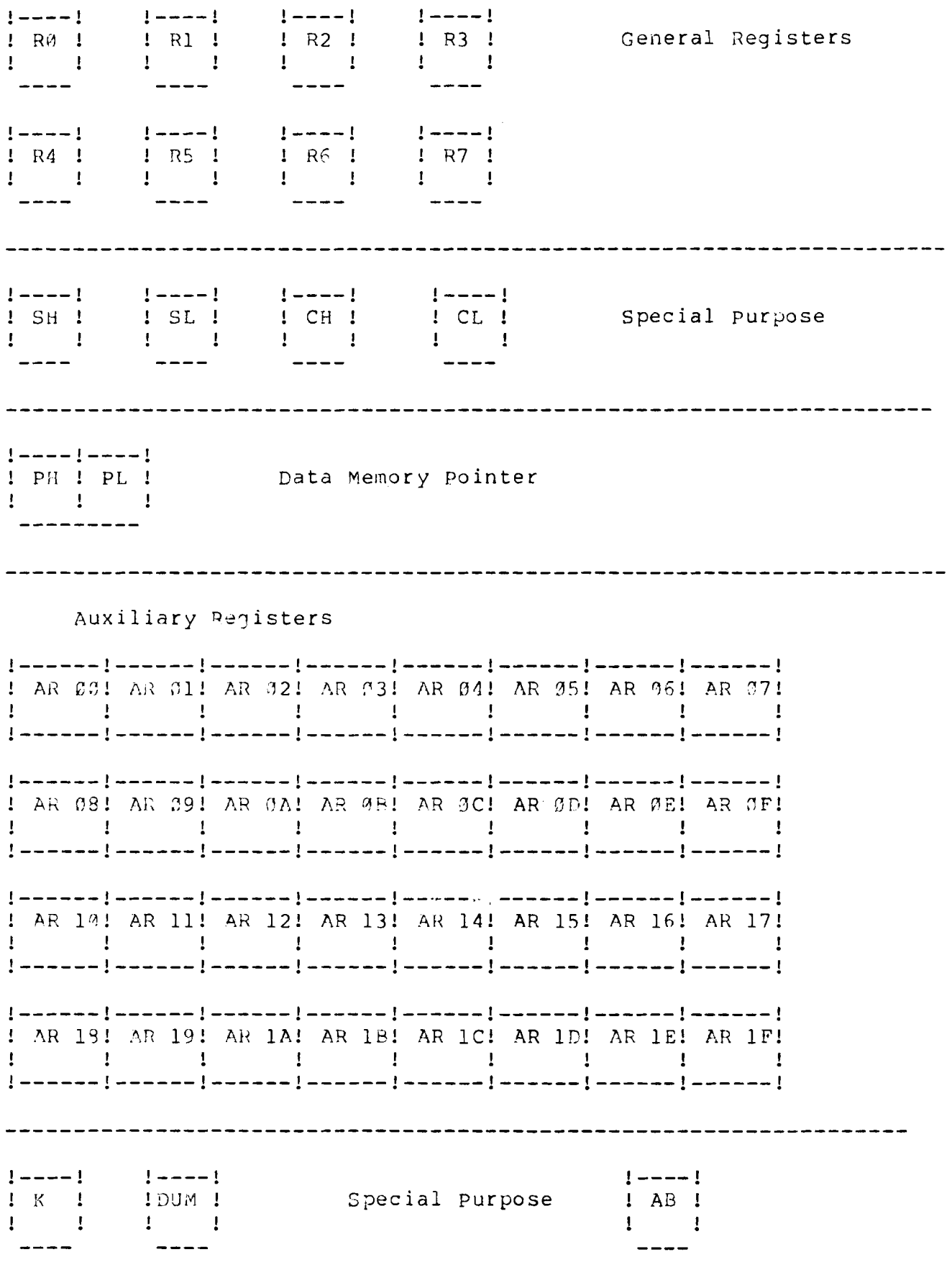
All input and output of Data to the IO bus is performed through the K register. This register is 8 bits in width. The Wang 2200 does not have a hardware interrupt structure, nor can data transfers occur without the direct participation of the host. The 2200 classifies as a Polled Interrupt machine.

The AB register, eight bits wide, forms the address of the peripheral that is to be accessed. The AB register is a special case register. It cannot be directly addressed by the Wang. To store data into the AB, data must pass through the K register.

Two status registers, each 8 bits wide are available. The SH register connotes to the Wang hardware status information. The SL register is used for software status as well as Data Memory bank selection. A more detailed breakdown of the functions of the SH and SL registers is available in section 1.2

32 Auxiliary registers (AR) of 16 bits each are available for general storage of pointers, counts and TS data. These 32 registers are what helps the Wang be so fast. Data can be transferred into and out of these AR registers only through the PH-PL register pair.

The system also contains a "stack" which is primarily used to hold the return address of routines using the Jump to Subroutine instructions. 192 eight bit words are available for the stack. This permits a nesting level of 96 items (16 bits wide) that can be placed into the stack.



Programming Model for Wang 2200 Series

1.2 Status register SL description

The SL register is an eight bit register used to select banks of Data memory, as well as provide software status to the program.

Register layout:

```
7 6 5 4  3 2 1 0

0 0 0 x  x x x x    Select First 64K Bank
0 1 0 x  x x x x    Select Second 64k Bank
1 0 0 x  x x x x    Select Third 64k Bank
1 1 0 x  x x x x    Select Fourth 64k Bank
0 0 1 x  x x x x    Select Fifth 64k Bank
0 1 1 x  x x x x    Select Sixth 64k Bank
1 0 1 x  x x x x    Select Seventh 64k Bank
1 1 1 x  x x x x    Select Eight 64k Bank

x x x x  x x 0 0    Atomize Pass
x x x x  x x 0 1    Assign Variables Pass
x x x x  x x 1 0    Execution Phase Pass
```

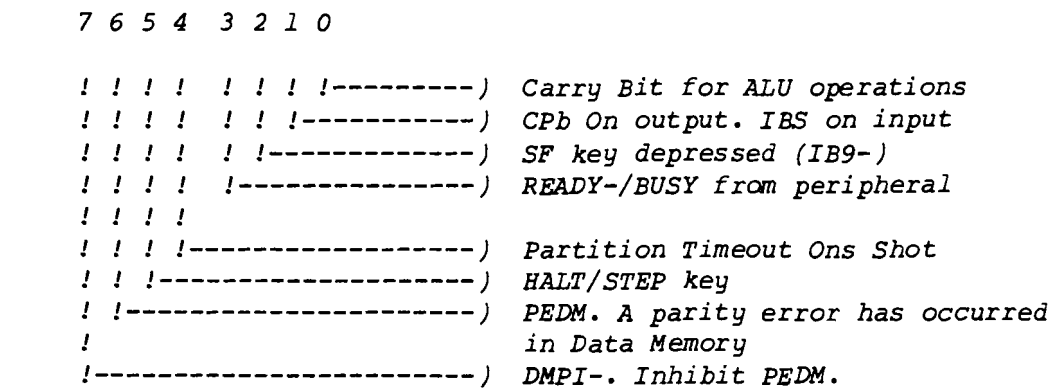
Partitions cannot overlap banks because there is no direct connection between the PHPL register pair and the SL register Bank selection bits. Software solutions to this problem become quite unwieldly. Therefore, unless Wang announces a new 2200 computer, the 65K limitation holds.

Global memory is done by accessing any location below $2000 to Bank 0. Therefore, no matter what bank you are in, accessing below $2000 switches you to Bank 0 for that cycle.

Because they actually remove that memory (Logically, not Physically) from every memory bank, larger Global memory would mean smaller partition memory!

1.3 Hardware Status register (SH)

The primary hardware status register is the SH register. Consisting of eight bits, it can be read or written to.



The computer makes the bus not available for input data from the IO devices by clearing bit 1, CPb-. When the computer wants data, it sets CPb- to a 1. The external device seeing this, places data on the Computers IB (Input Bus) and asserts IBS-, which in turn, resets the CPb- line, removing the request from the line.

Those of you familiar with the structure of the $GIO statements will remember that ENDI is a special termination for some commands. IB9- is the hardware derivative of ENDI- and sets bit 2 of the status register.

The READY-/BUSY line is the result of reading the RDY- or RB- line from the Wang IO bus. Remember that the 2200 is a polled IO machine, and has no hardware interrupt structure.

The DMPI- bit prevents the 2200 hardware from taking an automatic vector if an Parity Error Data Memory signal is received. In general, this bit is used by the diagnostics, for obvious reasons, and by initialization routines to size memory.

2.0 General Instruction breakdowns

Almost all minicomputers execute one complete instruction by fetching from memory and executing in timing cycles the functions necessary to perform the requested function.

In most minicomputers, the instruction word fetched is a multiple byte word, and is referred to as an instruction word. The size of the instruction word for the Wang 2200 VP/MVP system is 24 bits, or three bytes wide. However, not all of that word has functional meaning.

2.1 Parity Bit

The most significant bit of the instruction, 2!23, takes on the meaning of Parity. That is, any instruction that is fetched from Control Memory must contain the Parity bit set or reset such that the summation of all one bits results in ODD parity. If the instruction is fetched with EVEN parity, the system will vector to an hardware address as an error, and report that to the system console.

However, if Control memory is used as a data area, Parity is never checked when read. Therefore the whole three bytes of the instruction word may be used for data.

Parity must be formed by the user when the instruction is written initially into Control memory. The Wang 2200 does not generate the Parity when it writes, so it is possible for the user to write incorrect parity without noticing it. When that incorrect parity word is executed, an PECM error will occur.

When writing code using the Computer Concepts Corporation Assembler, the Assembler will automatically calculate and insert correct Parity into the instruction word. We cannot stress the importance of remembering that parity is generated by the user, not by Wang.

2.2 Classes of Instructions

In general, the following classes of instructions are available:

1: Branch
2: Masked Branch
3: Valued Branch
4: Register Comparison Branch
5: Register ALU
6: Immediate Data ALU
7: Peripheral Control
8: Load Data Memory Pointer
9: Stack Manipulation

The mnenomics that were assigned to the instruction was done only after research showed that the mnemonic would conform to Wang code. However, we must state that the mnemonics assigned are in part arbitrarily named due in part to the authors past experience, and in general, fit the role or function of other similar computers.

To a lesser extent, bits 2!22 and bits 2!21 of the instruction are used to define family classes. This is true, but we took the liberty of breaking down the instructions within even these limits to facilitate our understanding of them.

2.2.1 Branch Instructions

The unconditional branching of program flow is performed by this series of instruction words. A Jump, (JMP) and Jump to Subroutine (JSR) instruction allows the computer to ~~get to~~ any one of 65536 locations in Control memory. As a general background note, the system will execute all instruction words in cycles of 600 nanoseconds while in RAM control memory, while executing the same instructions in PROM control memory at 1.2 microseconds. This mode is called half-speed and is used only for the convenience of older Proms' with slower access times.

A general format of the literal branch instructions is:

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 0 1  v 1 x x  x x x x  x x x x  y y y y  y y 0 0
```

Where P = Parity Bit (Odd Parity)
x = LSD of Address
y = MSD of Address
v = Type of Branch

0 = Jump to Subroutine 1 = Jump

2.2.2 Masked Branch Instructions

Branching based upon the condition of individual bit structures in the registers, these instructions can selectively branch or not branch. Since a good part of the instruction is used up to decide whether or not to branch, the range that they can branch is limited to 1024 word pages or "maps". The words "pages" or "maps" are interchangeable, and both have the same meaning in the minicomputer world.

A general format for the masked branch instructions are:

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 1 0  f f x x  x x x x  x x x x  c c c c  B B B B
```

Where: p = Parity Bit (Odd Parity)
x = Location in current Page to branch to
c = Bits to Mask for test
B = Register to test (See Table 2 in Appendix)
ff = Type of Function

Note that in general, the rightmost four bits of this instruction will specify a code that determines which register, called the B register selection, will be used for the test. If the value of the bits are below 8, they refer directly to the General registers, R0 thru R7. If the number is greater than 7, the charts at Appendix i, Table 2 must be used. The contents of these registers are always "gated" to the ALU.

The following instructions belong to this family:

00 BTL Branch if Masked Bits True, Low Nibble
01 BTH Branch if Masked Bits True, High Nibble
10 BFL Branch if Masked Bits False, Low Nibble
11 BFH Branch if Masked Bits False, High Nibble

2.2.3 Valued Branch Instructions

Conditional branching based upon the comparison of a constant against a selected registers are performed by these instructions. Since a good part of the instruction is used up to decide whether or not to branch, the range that they can branch is limited to 1024 word pages or "maps". The words "pages" or "maps" are interchangeable, and both have the same meaning in the minicomputer world.

A general format for the Valued Branch instructions are:

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
---------------------------------------------------
p 1 1 1  f f x x  x x x x  x x x x  c c c c  B B B B
```

Where: p = Parity Bit (Odd Parity)
x = Location in current Page to branch to
c = Constant to test against
B = Register to test (See Table 2 in Appendix)
ff = Type of Function

Note that in general, the right most four bits of this instruction will specify a code that determines which register, called the B register selection, will be used for the test. If the value of the bits are below 8, they refer directly to the General registers, R0 thru R7. If the number is greater than 7, the charts at Appendix i, Table 2 must be used. The contents of these registers are always "gated" to the ALU.

The following instructions belong to this family:

00 BEL Branch if low nibble equals Constant
01 BEH Branch if high nibble equals Constant
10 BNL Branch if low nibble not equal to Constant
11 BNH Branch if high nibble not equal to Constant

2.2.4 Register Comparison Branch Instructions

Conditional branching based upon the comparison of two General registers with each other is performed by these instructions. Since a good part of the instruction is used up to decide whether or not to branch, the range that they can branch is limited to 1024 word pages or "maps". The words "pages" or "maps" are interchangeable, and both have the same meaning in the minicomputer world.

A general format for the Register Comparison Branch instructions are:

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 0 f  f f x x  x x x x  x x x x  A A A A  B B B B
```

Where: p = Parity Bit (Odd Parity)
x = Location in current Page (map) to branch to
A = Register A Select (See Table 1 in Appendix)
B = Register B Select (See Table 2 in Appendix)
ff = Type of Function

Note that in general, the right most four bits of this instruction will specify a code that determines which register, called the B register selection, will be used for the test. If the value of the bits are below 8, they refer directly to the General registers, R0 thru R7. If the number is greater than 7, the charts at Appendix i, Table 2 must be used. The contents of these registers are always "gated" to the ALU.

The following instructions belong to this family:

000 BLR Branch if Register A less than Register B (A < B)
001 BLRX Branch if Register A+1,A less than Register B+1,B (16 bit comparison)
010 BLER Branch if Register A less than or equal to Register B (A (= B)
011 BLEX Branch if Register A+1,A less than or equal Register B+1,B (16 bit comparison)
100 BER Branch if Register A equals Register B (A = B)
110 BNR Branch if Register A not equal to Register B (A <> B)

100 BEZ Branch if Register A equals Zero (Modification of BER)
110 BNZ Branch if Register A not equal to Zero.

2.2.5 Register ALU

The Arithmetic Logic Unit of the computer is usually the "brains" of the system. All mathametical data to be added, subtracted, anded, inclusive ored or any of the 16 possible Boolean functions must be processed by this ALU.

The Register ALU series of instructions permits two selected registers to be manipulated, and the result of this manipulation to be sent to another, called the destination register.

In general, we can select a Source A register, acted upon by a Source B register, and the result sent to a Destination register. The operation to be performed can be Decimal ADDs, Subtracts, binary Multiplies, ANDS, Exclusive ORs and several other functions to be outlined later.

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-----------------------------------------------------
p 0 0 a  a a v 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
aaa = ALU Operation
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
v = Extended Math Flag (16 bit) if = 1
xx = Carry flag controls

15 14	Carry Control	
0 0	Normal No effect	
0 1	Shift Decimal Character (SDC - In place of first 4 ALU instructions)	
1 0	Clear Carry First (Not on ALU =7)	CC
1 1	Set Carry first (Not on ALU = 7)	CS

The following table outlines the functions available for the register instructions ALU:

ALU Codes	Function	
0 0 0	OR	OR the contents of B with A
0 0 1	XOR	Exclusive OR the contents of B with A
0 1 0	AND	Logical AND of B with A
0 1 1	SBC	Binary Subtract with Carry, A - B
1 0 0	DAC	Decimal Add with Carry
1 0 1	DSC	Decimal Subract with Carry, A - B
1 1 0	ADC	Binary Add with Carry
1 1 1	MUL	Multiply two 4 bit values

Multiply is similar to the Immediate register multiply. Bits 14 and 15 of the instruction determine which two nibbles will be multiplied:

15	14	Function	
0	0	Multiply Lower B by Lower A	ALBL
0	1	Multiply Lower B by Upper A	AHBL
1	0	Multiply Upper B by Lower A	ALBH
1	1	Multiply Upper B by Upper A	AHBH

The Shift Decimal Character (SDC) instruction permits the manipulation of nibbles between Register A and Register B, as well as permitting a nibble shift. Bits 18 and 19 determine the shift status:

18	19	Function
0	0	B Lower 4 ORED A lower 4 (B nibble is always the MSN)
0	1	B lower 4 ORED A upper 4
1	0	B upper 4 ORED A lower 4
1	1	B upper 4 ORED A upper 4

Most of the instructions are self explanatory, but the DSC and the SBC should be outlined in more detail. The below examples goes through enough iterations of the instructions to be understood by most:

Original R2 = 00, R3 = 33

```
94C22F  DSC R2 < R2,00  ,CS      R2 = 99, Carry = 1

94C22F  DSC R2 < R2,00  ,CS      R2 = 98, Carry = 0

94C223  DSC R2 < R2,R3  ,CS      R2 = 64, Carry = 0

148223  DSC R2 < R2,R3  ,CC      R2 = 31, Carry = 0

148223  DSC R2 < R2,R3  ,CC      R2 = 98, Carry = 1
```

Note that the Carry flag is actually a value to be subtracted from the registers. In the next examples, the Carry bits complemented value is used.

Original R2 = 00, R3 = 33

```
8CC22F  SBC R2 < R2,00  ,CS      R2 = 00, Carry = 1

0C822F  SBC R2 < R2,00  ,CC      R2 = FF, Carry = 0 (See!)

8CC223  SBC R2 < R2,R3  ,CS      R2 = CC, Carry = 1

0C8223  SBC R2 < R2,R3  ,CC      R2 = 98, Carry = 1

8CC223  SBC R2 < R2,R3  ,CS      R2 = 65, Carry = 1
```

The above examples show us that we must be careful of what the carry is set too. Different instructions utilize it in a different manner than one would suspect.

2.2.6 Immediate Data ALU

This is very similar to the Register ALU instructions. The Source A register however is not present, and immediate data is supplied instead. Since changing of Control Memory once program execution is started is frowned upon, the immediate data is referred to as a constant, and allows us to subtract, add or perform boolean arithmetic on the Source B register and send the result to the Destination register.

This grouping of instructions allows the system to perform eight (8) different mathematical operations using a constant and a selected register. The result of this operation may be then stored into the same or a different register.

Data memory may be either read or written to at the same time that ALU operations are taking place. That is, the result of the operation is made available to be written to memory immediately. If data is being read from memory, it is transparent to this instruction.

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-----------------------------------------------------
p 0 1 a  a a i i  i i m m  d d d d  I I I I  B B B B
```

Where P = Parity Bit (Odd Parity)
aaa = ALU operation to perform (See following Page)
iii = High Nibble of Constant
III = Low Nibble of Constant
m = Data Memory Control (See table 4)
B = Source Register B gating (See Table 2)
d = Destination Register C (See Table 3)

Data can be written to memory by enabling the m bits, Data memory control. The data that is to be written is the result of the mathematical operation. The register referred to as DUM is in effect a null register. Any output of the ALU's will be stored here. Only from this register may memory be written to.

The following table outlines the functions available for the Immediate instructions:

ALU Codes (aaa)			Function	
0	0	0	IOR SET	OR the Contents of B with a Constant If Constant = 00
0	0	1	IXOR	Exclusive OR the contents of B with a constant
0	1	0	IAND	Logical AND the contents of B
0	1	1	IADD	ADD without carry, Binary
1	0	0	IDAC	Decimal Add with Carry
1	0	1	IDSC	Decimal Subtract with Carry
1	1	0	IADC	Binary Add with Carry
1	1	1	IMUL	Multiply two 4 bit nibbles

Throughout the Wang assembly code, the IOR instruction with $00 immediate data is used to load a register. Because of this, the following may be viewed as two different mnemonics:

```
I.E. 214E2F     IOR  K < $52,$00
     ... or     SET  K < $52
```

The latter looks better, and requires no operation to understand Further note that all Immediate instructions are prefaced with an I code to identify themselves apart from the Register instructions

```
I.E.  2BC2F0    IAND  R2 < $F0,R0
```

Loads the Register #2 with the contents of Register #0 logically "anded" by the constant $F0.

A Multiply instruction deserves further clarification. Bit 15 of the instruction determine which nibble, high or low, of the byte are going to be multiplied.

15	Function
0	Multiply B lower 4 by Constant lower 4 ALBL
1	" " upper " " " lower " ALBH

2.2.7 Peripheral Control

The Wang 2200 has only one real Peripheral control instruction. This instruction is called the CIO and is used to send strobes to the peripheral IO bus. Refer to the CIO instruction sheet for more detailed operation.

Other than the diagnostics, we have found that this is probably the least used of all instructions.

2.2.8 Load Data Memory Pointer

Called the LPI instruction, the system allows the programmer to directly access the PH-PL registers as one 16 bit value. The PH-PL pair, as previously brought out, points to the address to be written to or read from Data memory. Facilities are enabled in the instruction to clear memory locations without using any of the general registers. Refer to data sheet for the LPI for more detailed information.

2.2.9 Stack and Auxiliary Register Manipulations

The Wang 2200 computer contains 256 bytes of fast random access memory that is used for storage of stack data and for the 32 Auxiliary registers (AR) of 16 bits each. The AR registers form handy pointers to contain temporary data for usage during instruction execution without taking up room in data or control memory. The stack area serves as both a nesting place for subroutine calls, as well as holding temporary data. The author believes that the AR registers are vestigal artifacts from the structure of the 'T' machine series.

Imbedded among these instructions is the subroutine return instruction. It is of special note that it is located among this group. The Return instruction has the special characteristic of being the only instruction that can read or write to Control Memory. Because of the amount of time required to execute this instruction, Control Memory is rarely used for storage of variable data during program execution.

2.2.9.1 Auxiliary Register Manipulation

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  f f f 1  1 c m m  X c c r  r r r r  B B B B
```

Where P = Parity Bit (Odd Parity)
c = P register Control (See Table this section)
f = Function
m = Data Memory Control (See Table 4)
r = AR register (00 to 1F)
B = B Register Selection
X = Don't Care

Where f is as follows:

0 0 0	TPA	Transfer PH-PL to selected AR
0 0 1	XPA	Exchange current PH-PL with selected AR
1 0 1	TAP	Transfer selected AR to PH-PL

The PH-PL register pair is used as an index register through data memory. By means of bits 14, 10 and 9 of the instruction, the PH-PL pair may be incremented or decremented prior to storage in a selected AR or pushed to the stack.

Bit 14 10 9

0	0	0	No Effect
0	0	1	+1 to (PHPL) then store - PH-PL not affected
0	1	0	+2 " "
0	1	1	+3 " "
1	0	0	No Effect
1	0	1	-1 to (PHPL) then store - PH-PL not affected
1	1	0	-2 to (PHPL) then store - PH-PL not affected
1	1	1	-3 to (PHPL) then store - PH-PL not affected

In all the above cases, the PHPL pair are first transferred to an intermediate register where the actual increment or decrement takes place. The real contents of the PHPL pair are never affected except in the XPA instruction.

I.E. 03A00F XPA AR 00

Transfers the contents of the PHPL pair to register AR 00, and at the same time, transfers the contents of AR 00 to PHPL.

Data memory may be read or written to by this instruction as well. The B register gating is only effective for the write operation.

2.2.9.2 Stack Manipulation Instructions

The stack manipulation instructions allows the data to be placed (Pushed) onto the stack, and taken (Popped) from the stack. The address counters of the stack are transparent to both the user and the machine language. These counters are automatically incremented and decremented for every Push or Pop operation.

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  f f f 1  1 c m m  X c c 0  0 0 0 0  B B B B
  0 0 0  0 1 0 1  1 x m m  X x x 0  0 0 0 0  B B B B
```

Where P = Parity Bit (Odd Parity)
x = P register control (TPS only Refer to previous section
m = Data memory Control (See Table 4)
c = Control Memory Functions (Only during the RTS instruction - See Table 5)
B = B Register Selection (See Table 2)
X = Don't Care

Functions

0 1 0	TPS	Transfer PHPL registers to stack
1 1 0	TSP	Transfer contents of stack to PHPL registers
0 1 1	RTS	Return from Subroutine

The RTS instruction is the only instruction that is not as straight forward as it would appear to be. This instruction is the only one that may Read or Write to Control Memory. When an RCM (Read Control Memory) or WCM (Write Control Memory) operation is requested, the system executes what is called a LOP, or Long OPeration. A LOP causes the stack to be popped twice, and the resultant data sent to the Control Memory Address Register. The Read or Write operation is performed, and in the case of a Read operation the data goes to the K, PH and PL register. The MSB is in K, while the LSB is in PL. If the operation has been a Write operation, the K register, PH and PL registers would be sent to the CM module and written. Note that the design of the hardware requires that the K register must be 1's complemented prior to writing.

3.0 Data Memory Read and Write features

Remember that the Wang computer is a semi-pipelined machine. As such, some of the features may seem strange to those not accustomed to this type of computer. However, the advantages of the pipeline machine are such that data may be pre-fetched from memory for use by an instruction further down the flow.

Under no circumstance may the contents of Data memory be read for use by the current instruction. This would require "wait" states, that would be against the concept set by Wang. However, we can read data memory during the course of an instruction, and use the results later on.

Other than the Jump, Branch and Subroutine calls, all instructions have two bits that determine what Data memory functions are to be performed. These bits, bit 13 and 12, are decoded as follows:

13	12		
0	0		No Data Memory Operation is to be perfromed
0	1	,RD	Read Contents of Data Memory
1	0	,W1	Write byte at current PHPL
1	1	,W2	Write byte at current PHPL XOR 1

Except for the LPI instruction, whenever a Data memory operation is encountered, the CURRENT position of the PHPL pair is used to form the address to READ or to WRITE to.

Essentially, the contents of the PHPL pair is sent to the memory and latched there prior to the math operation. Therefore, if the contents of the PHPL pair were 0900, and the following instruction was issued:

```
SET     PL < $50  ,RD
```

The data at location 0900 would be read, not the data at 0950 as one may assume. As stated before, the only exception to this is the LPI instruction, which acts on the PHPL prior to the memory read. Therefore, if the PHPL pair was set to 0900, and the following instruction was issued:

```
LPI     $0950  ,RD
```

The data at location 0950 would be read.

Now where does this data go? When read, the data will be available on the next instruction cycle in the CH CL pair. These two registers form the 16 bits of the Data memory location accessed. If the PHPL pair was an even number, the read command would result in the "high" byte to be placed in the CH register, while the "low" byte would be placed in the CL register.

If the PHPL were odd, then the "High" byte gets placed into the CL register, while the "low" byte gets placed into the CH register. If not confused by now, let me further muddle your mind by stating simply that the BYTE pointed to by the PHPL pair will be placed into the CH register, while the opposite byte gets placed into the CL register. The reason for this is that the PHPL pair addresses 16 bit words, and cannot do two reads from Data memory.

Assume that the following data is in location 0542 and the following instructions are issued:

PHPL = 0542 Data at 0542 = 1234

```
SET   R0 < 0  ,RD
```

CH would contain 12, CL would contain 34

PHPL = 0543 Data at 0542 = 1234

```
SET   R0 < 0  ,RD
```

CH would contain 34, CL would contain 12

The ability to perform these types of reads may seem dubious at first, but it sure does simplify operations such as shifts of data.

Writing to Data memory follows the same basic rules. However, note that a READ operation is performed, even though a write operation will be requested. This type of operation is called a READ-MODIFY-WRITE operation.

If the ,W1 option was chosen, the RESULT of the mathametical operation is sent to the memory at the current position of the PHPL pair.

If the ,W2 option was chosen, the RESULT of the operation is sent to the memory location opposite to the PHPL pair. That is, if the PHPL pair was Even, the Odd byte gets written. If the PHPL pair was Odd, the Even byte location will be written to.

```
PHPL = 0542     Contents of 0542  =  1234

     SET  R0 < $53  ,W1

     The new contents of 0542 would be  5334

     IAND  R2 < $12,R0  ,W2

     The new contents of 0542 would be  5312

PHPL = 0543     Contents of 0542  =  5312

     IADD  R0 < $20,R0  ,W1

     The new contents of 0542 would be  5373

     IOR   R2 < $F0,R0  ,W2

     The new contents of 0542 would be  F373
```

The AR and stack registers may also perform Read/Write operations. However, since no math is performed by them, the contents of the select B register is used as Data to be sent during a write operation:

```
          TPA    AR 00  ,W1,R5
```

Would result in the R5 register to be written at the current location pointed to by the PHPL register.

All write operations take place through the DUM register. Actually, this register is used to hold the result of any math operation as well. Therefore, we can write to memory, without altering any registers, by specifying the DUM register as the destination:

```
SET    DUM  < $55   ,W1
XOR    DUM  < CH+,R0 ,W2
```

The only exception this rule is again, the LPI instruction. As specified in the data sheet for this instruction, the issuance of any Write command results in the clearing of that byte:

```
LPI    $0542 ,W1      Clears location 0542 to Zero
LPI    $0542 ,W2      Clears location 0543 to Zero
LPI    $0543 ,W1      Clears location 0543 to Zero
LPI    $0543 ,W2      Clears location 0542 to Zero
```

ADC (Binary Add with Carry)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 1  1 0 0 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control
0 0	Normal, Initial Carry State not affected
0 1	Normal, Initial Carry State not affected
1 0	Clear Carry First ,CC
1 1	Set Carry first ,CS

The following table outlines the results returned for various constants using the ADC instruction

		Result CC		Result CS	
R0	R1	R2	Carry	R2	Carry
00	00	00	0	01	0
00	01	01	0	02	0
01	00	01	0	02	0
55	55	AA	0	AB	0
AA	AA	54	1	55	1
80	01	81	0	82	0
80	FF	7F	1	80	1
12	34	46	0	47	0
56	78	CE	0	CF	0
9A	BC	56	1	57	1
F0	DE	CE	1	CF	1
02	FF	01	1	02	1
FF	FF	FE	1	FF	1

```
188201   ADC  R2 < R0,R1  ,CC
98C201   ADC  R2 < R0,R1  ,CS
```

ADCX (Binary Add with carry 16 bit)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-----------------------------------------------------
p 0 0 1  1 0 1 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control
0 0	Normal, Initial Carry State not affected
0 1	Normal, Initial Carry State not affected
1 0	Clear Carry First ,CC
1 1	Set Carry first ,CS

The following table outlines the results returned for various constants using the ADCX instruction

```
9A8402   ADCX  R4 < R0,R2  ,CC
1AC402   ADCX  R4 < R0,R2  ,CS
```

		Result	CC	Result	CS
R1 R0	R3 R2	R5 R4	Carry	R5 R4	Carry
00 00	00 00	00 00	0	00 01	0
00 00	00 01	00 01	0	00 02	0
00 01	00 00	00 01	0	00 02	0
55 AA	AA 55	FF FF	0	00 00	1
11 22	33 44	44 66	0	44 67	0
44 55	66 77	AA CC	0	AA CD	0
80 00	00 01	80 01	0	80 02	0
00 01	80 00	80 01	0	80 02	0
9A BC	CD EF	68 AB	1	68 AC	1
AA AA	22 22	CC CC	0	CC CD	0
55 19	55 91	AA AA	0	AA AB	0

AND (Logical AND registers)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  1 0 0 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control
0 0	Normal, Initial Carry State not affected
0 1	Not Permitted (See SDC, SDCX makeups)
1 0	Clear Carry First ,CC
1 1	Set Carry first ,CS

The following table outlines the results returned for various constants using the AND instruction

		Result	
R0	R1	R2	Carry
00	00	00	0
00	01	00	0
01	00	00	0
55	55	55	0
AA	AA	AA	0
80	01	00	0
80	FF	80	0
12	34	10	0
56	78	50	0
9A	BC	98	0
F0	DE	D0	0
02	FF	02	0
FF	FF	FF	0

888201 AND R2 < R0,R1 ,CC

ANDX (Logical AND registers 16 bit)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  1 0 1 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control
0 0	Normal, Initial Carry State not affected
0 1	Not Permitted (See SDC, SDCX makeups)
1 0	Clear Carry First ,CC
1 1	Set Carry first ,CS

The following table outlines the results returned for various constants using the ANDX instruction

R1	R0	R3	R2	R5	R4	Carry
00	00	00	00	00	00	0
00	00	00	01	00	00	0
00	01	00	00	00	00	0
55	AA	AA	55	00	00	0
11	22	33	44	11	00	0
44	55	66	77	44	55	0
80	00	00	01	00	00	0
00	01	80	00	00	00	0
9A	BC	CD	EF	88	AC	0
AA	AA	22	22	22	22	0
55	19	55	91	55	11	0

```
0A8402   ANDX  R4 < R0,R2  ,CC
```

BEH

BEH (Branch if Equal, High nibble)

Family Type: Valued Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 1 1  0 1 x x  x x x x  x x x x  c c c c  B B B B
```

Where: P = Parity Bit (Odd Parity)
x = Map location to Branch to
~~ff = Function to perform~~
c = Constant to compare Register against
B = B Register to test. (See Table 2 in the Appendix)

Direct mathematical comparisons of an immediate 4 bit nibble with any of the 16 registers is performed by this instruction. As with the previous Masked instructions, any location within the 1024 map can be branched to.

I.E. Test program in memory:

```
          $ORG    $6000

6000  740332   BEH  $03,R2   $6003
6001  A0001F   SET  R0 < $01          Not Taken
6002  DC0050   JMP  $5000
6003  20000F   SET  R0 < $00          Branch Taken
6004  DC0050   JMP  $5000
```

Branch will be taken if nibble is equal to 3x

BEL

BEL (Branch if Equal, Low nibble)

Family Type: Valued Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 1 1  0 0 x x  x x x x  x x x x  c c c c  B B B B
```

Where: *P = Parity Bit (Odd Parity)*
x = Map location to Branch to
~~*ff = Function to perform*~~
c = Constant to compare Register against
B = B Register to test. (See Table 2 in the Appendix)

Direct mathematical comparisons of an immediate 4 bit nibble with any of the 16 registers is performed by this instruction. As with the previous Masked instructions, any location within the 1024 map can be branched to.

I.E. Test program in memory:

```
          $ORG    $6000

6000  F00352   BEL  $05,R2   $6003
6001  A0001F   SET  R0 < $01          Not Taken
6002  DC0050   JMP  $5000
6003  20000F   SET  R0 < $00          Branch Taken
6004  DC0050   JMP  $5000
```

Branch will be taken if nibble is equal to x5

BER (Branch if A = B)

Family Type: Register Comparison Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 0 1  0 0 x x  x x x x  x x x x  A A A A  B B B B
```

Where: p = Parity Bit (Odd Parity)
x = Address in map to branch to
A = Register Selection A (See table 1 in the Appendix)
B = Register Selection B (See table 2 in the Appendix)

Facilities are included, within the Wang CPU, to test the value of all 8 bits, and sometimes 16 bits with another Register.

Examples of this instruction and whether or not the Branch will be taken follows below:

R3	R6	Branch Yes	Branch No
00	00	X	
01	10		X
10	05		X
80	8F		X
80	02		X
02	80		X
FF	83		X
DE	DE	X	

```
D20D36   BER R3,R6  TEST
```

BEZ (Branch if A = Zero (0))

Family Type: Register Comparison Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 0 1  0 0 x x  x x x x  x x x x  A A A A  1 1 1 1
```

Where: *p = Parity Bit (Odd Parity)*
x = Address in map to branch to
A = Register Selection A (See table 1 in the Appendix)

This Branch instruction is syntacally the same as the BER instruction, with the B register gating bits set to the constant zero. This format allows a clearer understanding of the flow of a program.

Examples of this instruction and whether or not the Branch will be taken follows below:

R3	Branch	
	Yes	No
00	X	
01		X
10		X
80		X
80		X
02		X
FF		X
DE		X

```
D20D3F   BEZ   R3  TEST
```

BFH (Branch if Bits False, High nibble)

Family Type: Masked Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 1 0  1 1 x x  x x x x  x x.x x  c c c c  B B B B
```

Where: *p* = Parity Bit (Odd Parity)
x = Location in current Page to branch to
c = Bits to Mask for test
B = Register to test (See Table 2 in Appendix)

This instruction implies that a "mask" of the contents of the selected B register is performed. The original contents of the register are not touched. Only those bits set to one in the 'c' field of the instruction are evaluated.

The bits masked in the high nibble of the selected register are evaluated. If all the masked bits are False, that is low, the branch is taken. If any of the masked bits are True, or high, the branch is not taken.

The branch, if taken, can only traverse a Range of 1024 decimal locations from the Base Page or Map. The computer may not cross a map or page boundary with a taken Branch.

I.E. Test program in memory:

```
          $ORG    $6000

6000  6C0352   BFH  $05,R2   $6003
6001  A0001F   SET  R0 < $01            Not Taken
6002  DC0050   JMP  $5000
6003  20000F   SET  R0 < $00            Branch Taken
6004  DC0050   JMP  $5000
```

Branch will be taken if nibbles are 0x,2x,8x or Ax.

BFL (Branch if Bits False, Low nibble)

Family Type: Masked Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 1 0  1 0 x x  x x x x  x x x x  c c c c  B B B B
```

Where: p = Parity Bit (Odd Parity)
x = Location in current Page to branch to
c = Bits to Mask for test
B = Register to test (See Table 2 in Appendix)

This instruction implies that a "mask" of the contents of the selected B register is performed. The original contents of the register are not touched. Only those bits set to one in the 'c' field of the instruction are evaluated.

The bits masked in the low nibble of the selected register are evaluated. If <u>all</u> the masked bits are False, that is low, the branch is taken. If <u>any</u> of the masked bits are True, or high, the branch is not taken.

The branch, if taken, can only traverse a Range of 1024 decimal locations from the Base Page or Map. The computer may not cross a map or page boundary with a taken Branch.

I.E. Test program in memory:

```
          $ORG    $6000

6000  E80332   BFL  $03,R2   $6003
6001  A0001F   SET  R0 < $01          Not Taken
6002  DC0050   JMP  $5000
6003  20000F   SET  R0 < $00          Branch Taken
6004  DC0050   JMP  $5000
```

Branch will be taken if nibbles are x0,x4,x8 or xC.

BLER

BLER (Branch if A <= B)

Family Type: Register Comparison Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 0 0  1 0 x x  x x x x  x x x x  A A A A  B B B B
```

Where: *p = Parity Bit (Odd Parity)*
x = Address in map to branch to
A = Register Selection A (See table 1 in the Appendix)
B = Register Selection B (See table 2 in the Appendix)

Facilities are included, within the Wang CPU, to test the value of all 8 bits, and sometimes 16 bits with another Register.

Examples of this instruction and whether or not the Branch will be taken follows below:

R3	R6	Branch Yes	Branch No
00	00	X	
01	10	X	
10	05		X
80	8F	X	
80	02		X
02	80	X	
FF	83		X
DE	DE	X	

```
CA0D36   BLER R3,R6  TEST
```

BLEX (Branch if A <= B 16 bit test)

Family Type: Register Comparison Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-----------------------------------------------------
p 1 0 0  1 1 x x  x x x x  x x x x  A A A A  B B B B
```

Where: p = Parity Bit (Odd Parity)
x = Address in map to branch to
A = Register Selection A (See table 1 in the Appendix)
B = Register Selection B (See table 2 in the Appendix)

Facilities are included, within the Wang CPU, to test the value of all 8 bits, and sometimes 16 bits with another Register.

Examples of this instruction are listed below with the branch conditions:

R3	R2	R6	R5	Branch Yes	Branch No
00	00	00	00	X	
00	01	00	10	X	
01	00	00	10		X
00	10	00	05		X
10	00	05	00		X
00	01	05	00	X	
80	8F	80	8F	X	
80	00	8F	00	X	
80	00	00	02		X
00	08	02	00	X	
02	00	80	00	X	
FF	FF	83	00		X
DE	DE	DE	DE	X	

```
CE1025  BLEX  R2,R5  TEST
```

BLR (Branch if A < B)

Family Type: Register Comparison Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 0 0  0 0 x x  x x x x  x x x x  A A A A  B B B B
```

Where: p = Parity Bit (Odd Parity)
x = Address in map to branch to
A = Register Selection A (See table 1 in the Appendix)
B = Register Selection B (See table 2 in the Appendix)

Facilities are included, within the Wang CPU, to test the value of all 8 bits, and sometimes 16 bits with another Register.

Examples of this instruction and whether or not the Branch will be taken follows below:

R3	R6	Branch Yes	Branch No
00	00		X
01	10	X	
10	05		X
80	8F	X	
80	02		X
02	80	X	
FF	83		X
DE	DE		X

```
420D36   BLR  R3,R6  TEST
```

BLRX (Branch if A < B 16 bit test)

Family Type: Register Comparison Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 0 0  0 1 x x  x x x x  x x x x  A A A A  B B B B
```

Where:
- p = Parity Bit (Odd Parity)
- x = Address in map to branch to
- A = Register Selection A (See table 1 in the Appendix)
- B = Register Selection B (See table 2 in the Appendix)

Facilities are included, within the Wang CPU, to test the value of all 8 bits, and sometimes 16 bits with another Register.

Examples of this instruction are listed below with the branch conditions:

R3	R2	R6	R5	Branch Yes	Branch No
00	00	00	00		X
00	01	00	10	X	
01	00	00	10		X
00	10	00	05		X
10	00	05	00		X
00	01	05	00	X	
80	8F	80	8F		X
80	00	8F	00	X	
80	00	00	02		X
00	08	02	00	X	
02	00	80	00	X	
FF	FF	83	00		X
DE	DE	DE	DE		X

```
461025  BLRX  R2,R5   TEST
```

BNH (Branch if Not Equal, High nibble)

Family Type: Valued Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 1 1  1 1 x x  x x x x  x x x x  c c c c  B B B B
```

Where: *P = Parity Bit (Odd Parity)*
x = Map location to Branch to
ff = Function to perform
c = Constant to compare Register against
B = B Register to test. (See Table 2 in the Appendix)

Direct mathematical comparisons of an immediate 4 bit nibble with any of the 16 registers is performed by this instruction. Any location within the 1024 map can be branched to.

I.E. Test program in memory:

```
          $ORG    $6000

6000  FC03C2   BNH  $0C,R2   $6003
6001  A0001F   SET  R0 < $01          Not Taken
6002  DC0050   JMP  $5000
6003  20000F   SET  R0 < $00          Branch Taken
6004  DC0050   JMP  $5000
```

Branch will be always be taken unless nibble is equal to Cx

BNL (Branch if Not Equal, Low nibble)

Family Type: Valued Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 1 1  1 0 x x  x x x x  x x x x  c c c c  B B B B
```

Where:
- P = Parity Bit (Odd Parity)
- x = Map location to Branch to
- ff = Function to perform
- c = Constant to compare Register against
- B = B Register to test. (See Table 2 in the Appendix)

Direct mathematical comparisons of an immediate 4 bit nibble with any of the 16 registers is performed by this instruction. As with the previous Masked instructions, any location within the 1024 map can be branched to.

I.E. Test program in memory:

```
          $ORG    $6000

6000  7803A2   BNL  $0A,R2   $6003
6001  A0001F   SET  R0 < $01          Not Taken
6002  DC0050   JMP  $5000
6003  20000F   SET  R0 < $00          Branch Taken
6004  DC0050   JMP  $5000
```

Branch will be always be taken unless nibble is equal to xA

BNR (Branch if A <> B)

Family Type: Register Comparison Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 0 1  1 0 x x  x x x x  x x x x  A A A A  B B B B
```

Where: p = Parity Bit (Odd Parity)
x = Address in map to branch to
A = Register Selection A (See table 1 in the Appendix)
B = Register Selection B (See table 2 in the Appendix)

Facilities are included, within the Wang CPU, to test the value of all 8 bits, and sometimes 16 bits with another Register.

Examples of this instruction and whether or not the Branch will be taken follows below:

R3	R6	Branch	
		Yes	No
00	00		X
01	10	X	
10	05	X	
80	8F	X	
80	02	X	
02	80	X	
FF	83	X	
DE	DE		X

```
5A0D36   BNR R3,R6  TEST
```

BNZ (Branch if A <> Zero (0))

Family Type: Register Comparison Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 0 1  1 0 x x  x x x x  x x x x  A A A A  1 1 1 1
```

Where: p = Parity Bit (Odd Parity)
x = Address in map to branch to
A = Register Selection A (See table 1 in the Appendix)

This instruction is an adaptation of the BNR instruction with the B register gating bits set to the constant zero. It allows us to better understand the flow of the program by having only to look at one operand. This instruction will branch whenever a non-zero operand is present in the A register gating bits.

Examples of this instruction and whether or not the Branch will be taken follows below:

R3	Branch	
	Yes	No
00		X
01	X	
10	X	
80	X	
80	X	
02	X	
FF	X	
DE	X	

```
5A0D3F   BNZ    R3  TEST
```

BTH (Branch if Bits True, High nibble)

Family Type: Masked Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-----------------------------------------------------
p 1 1 0  0 1 x x  x x x x  x x x x  c c c c  B B B B
```

Where: p = Parity Bit (Odd Parity)
x = Location in current Page to branch to
c = Bits to Mask for test
B = Register to test (See Table 2 in Appendix)

This instruction implies that a "mask" of the contents of the selected B register is performed. The original contents of the register are not touched. Only those bits set to one in the 'c' field of the instruction are evaluated.

The bits masked in the high nibble of the selected register are evaluated. If all the masked bits are True, that is high, the branch is taken. If any of the masked bits are false, or low, the branch is not taken.

The branch, if taken, can only traverse a Range of 1024 decimal locations from the Base Page or Map. The computer may not cross a map or page boundary with a taken Branch.

I.E. Test program in memory:

```
          $ORG    $6000

6000  E403A2   BTH  $0A,R2   $6003
6001  A0001F   SET  R0 < $01          Not Taken
6002  DC0050   JMP  $5000
6003  20000F   SET  R0 < $00          Branch Taken
6004  DC0050   JMP  $5000
```

Branch will be taken if nibbles are Ax,Bx,Ex or Fx.

BTL (Branch if Bits True, Low Nibble)

Family Type: Masked Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 1 0  0 0 x x  x x x x  x x x x  c c c c  B B B B
```

Where: p = Parity Bit (Odd Parity)
x = Location in current Page to branch to
c = Bits to Mask for test
B = Register to test (See Table 2 in Appendix)

This instruction implies that a "mask" of the contents of the selected B register is performed. The original contents of the register are not touched. Only those bits set to one in the 'c' field of the instruction are evaluated.

The bits masked in the low nibble of the selected register are evaluated. If all the masked bits are True, that is high, the branch is taken. If any of the masked bits are false, or low, the branch is not taken.

The branch, if taken, can only traverse a Range of 1024 decimal locations from the Base Page or Map. The computer may not cross a map or page boundary with a taken Branch.

I.E. Test program in memory:

```
          $ORG    $6000

6000  600362   BTL  $06,R2   $6003
6001  A0001F   SET  R0 < $01          Not Taken
6002  DC0050   JMP  $5000
6003  20000F   SET  R0 < $00          Branch Taken
6004  DC0050   JMP  $5000
```

If values of nibbles are x6,x7,xE or xF Branch will be taken.

CIO (Input Output Instruction)

Family: Peripheral Control

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 1  0 1 1 1  1 0 x x  y y y y  y d d d  z z d d
```

Where:
- p = Parity Bit (Odd Parity)
- xx = Data Memory Control (See table 4 of the Appendix)
- y = Strobe Control
- d = Immediate data field
- zz = if equal to 11, fire the partition timer (MVP only)
  ASM Syntax for timer is: CIO TIM

Strobe Control

x x x x 1	Fire internal IBS one shot (SRS). Sets CPb Basically used for Status Requests from MUXD
x x x 1 x	Fire CBS Strobe
x x 1 x x	Fire OBS Strobe
x 1 x x x	Fire ABS strobe
1 x x x x	Clock the AB data register from register K This is performed prior to any strobes.

The purpose of this group is to provide a means of communicating with the peripherals, CRT, DISK or other devices on the IO bus.

The Input/Output instructions (I/O) allow the 2200 to fire one of four one-shot strobes and allow limited peripheral data transfers to take place.

All strobes above are about 5 microseconds in length. Data transmission to the peripheral is always done through the K register. Data present in K is always present on the OB bus. Transfer of data to the AB bus must also occur through K register, via an Clock AB strobe control. All oneshots can be fired together, or through combinations, but all fire at the same time.

The next page contains sample usages of the CIO series instructions.

CIO (Input Output Instruction) - Cont'd

Selecting devices:

```
A00E5F  SET  K < $05        Address of CRT out
178C00  CIO  CLK AB,ABS     Transfer K to AB. Address = 05
                               Issue ABS strobe.
```

If any device has the address $05, they will lock on.

Waiting for Input from Devices:

```
AB8DDD  IAND   SH < $ED,SH   Mask out and clear CPb
E10D2D  BTL    $02,SH $090D  Branch if IBS
5D2008  JMP    *-1           Loop Back
```

Waiting for Device Ready:

```
E36A8D  BTL    $08,SH   $0B6A   Test Ready bit
5D2008  JMP    *-1              Loop Back
```

Check if Timeout, Refire Timer

```
3000  6C021D     BFH  $01,SH     FIRE      See if expired
3001  87800F     RTS                       Still on
3002  17800C FIRE    CIO   TIM                 Fire Oneshot
3003  87800F     RTS                       Return to caller
```

DAC (Decimal Add with Carry)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 1  0 0 0 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control
0 0	Normal, Initial Carry State not affected
0 1	Normal, Initial Carry State not affected
1 0	Clear Carry First ,CC
1 1	Set Carry first ,CS

The following table outlines the results returned for various constants using the DAC instruction

		Result CC		Result CS	
R0	R1	R2	Carry	R2	Carry
00	00	00	0	01	0
00	01	01	0	02	0
01	00	01	0	02	0
55	55	10	1	11	1
AA	AA	BA	1	BB	1
80	01	81	0	82	0
80	FF	E5	1	E6	1
12	34	46	0	47	0
56	78	34	1	35	1
9A	BC	BC	1	BD	1
F0	DE	34	1	35	1
02	FF	67	1	68	1
FF	FF	54	1	55	1

```
908201   DAC  R2 < R0,R1  ,CC
10C201   DAC  R2 < R0,R1  ,CS
```

DACX (Decimal Add with carry 16 bit)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 1  0 0 1 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control
0 0	Normal, Initial Carry State not affected
0 1	Normal, Initial Carry State not affected
1 0	Clear Carry First ,CC
1 1	Set Carry first ,CS

The following table outlines the results returned for various constants using the DACX instruction

```
128402   DACX  R4 < R0,R2  ,CC
92C402   DACX  R4 < R0,R2  ,CS
```

				Result		CC	Result		CS
R1	R0	R3	R2	R5	R4	Carry	R5	R4	Carry
00	00	00	00	00	00	0	00	01	0
00	00	00	01	00	01	0	00	02	0
00	01	00	00	00	01	0	00	02	0
55	AA	AA	55	66	55	1	66	66	1
11	22	33	44	44	66	0	44	67	0
44	55	66	77	11	32	1	11	33	1
80	00	00	01	80	01	0	80	02	0
00	01	80	00	80	01	0	80	02	0
9A	BC	CD	EF	CE	01	1	CE	02	1
AA	AA	22	22	33	32	1	33	33	1
55	19	55	91	11	10	1	11	11	1

DSC (Decimal Subtract with Carry)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 1  0 1 0 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control	
0 0	Normal, Initial Carry State not affected	
0 1	Normal, Initial Carry State not affected	
1 0	Clear Carry First	,CC
1 1	Set Carry first	,CS

The following table outlines the results returned for various constants using the DSC instruction

		Result CC		Result CS	
R0	R1	R2	Carry	R2	Carry
00	00	00	0	99	1
00	01	99	1	98	1
01	00	01	0	00	0
55	55	00	0	99	1
AA	AA	00	0	99	1
80	01	79	0	78	0
80	FF	2B	1	2A	1
12	34	78	1	77	1
56	78	78	1	77	1
9A	BC	78	1	77	1
F0	DE	1C	0	1B	0
02	FF	AD	1	AC	1
FF	FF	00	0	99	1

```
148201   DSC  R2 < R0,R1  ,CC
94C201   DSC  R2 < R0,R1  ,CS
```

DSCX (Decimal Subtract with carry 16 bit)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 1  0 1 1 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control
0 0	Normal, Initial Carry State not affected
0 1	Normal, Initial Carry State not affected
1 0	Clear Carry First ,CC
1 1	Set Carry first ,CS

The following table outlines the results returned for various constants using the DSCX instruction

```
968402   DSCX  R4 < R0,R2  ,CC
16C402   DSCX  R4 < R0,R2  ,CS
```

				Result		CC	Result		CS
R1	R0	R3	R2	R5	R4	Carry	R5	R4	Carry
00	00	00	00	00	00	0	99	99	1
00	00	00	01	99	99	1	99	98	1
00	01	00	00	00	01	0	00	00	0
55	AA	AA	55	45	55	1	45	54	1
11	22	33	44	77	78	1	77	77	1
44	55	66	77	77	78	1	77	77	1
80	00	00	01	79	99	0	79	98	0
00	01	80	00	20	01	1	20	00	1
9A	BC	CD	EF	66	67	1	66	66	1
AA	AA	22	22	88	88	0	88	87	0
55	19	55	91	99	28	1	99	27	1

IADC (Immediate Binary Add with Carry registers)

Family Type: Immediate ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 1 1  1 0 i i  i i m m  d d d d  I I I I  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
iiii = High order Nibble of Immediate Data Constant
IIII = Low order Nibble of Immediate Data
B = Source Register B (See table 2)
d = Destination register (See table 3)

The following table outlines the results returned for various constants using the IADC Instruction

IADC R2 < (Immediate Data),R1

Imm Data	R1	Result CC R2	Result CC Carry	Result CS R2	Result CS Carry
00	00	00	0	01	0
00	01	01	0	02	0
01	00	01	0	02	0
55	55	AA	0	AB	0
AA	AA	54	1	55	1
80	01	81	0	82	0
80	FF	7F	1	80	1
12	34	46	0	47	0
56	78	CE	0	CF	0
9A	BC	56	1	57	1
F0	DE	CE	1	CF	1
02	FF	01	1	02	1
FF	FF	FE	1	FF	1

IADD (Immediate ADD registers)

Family Type: Immediate ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-----------------------------------------------------
p 0 1 0  1 1 i i  i i m m  d d d d  I I I I  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
iiii = High order Nibble of Immediate Data Constant
IIII = Low order Nibble of Immediate Data
B = Source Register B (See table 2)
d = Destination register (See table 3)

The following table outlines the results returned for various constants using the IADD instruction

Imm Data	R1	Result R2	Carry
00	00	00	0
00	01	01	0
01	00	01	0
55	55	AA	0
AA	AA	54	0
80	01	81	0
80	FF	7F	0
12	34	46	0
56	78	CE	0
9A	BC	56	0
F0	DE	CE	0
02	FF	01	0
FF	FF	FE	0

IADD R2 < (Immediate Data),R1

IAND (Immediate Logical AND registers)

Family Type: Immediate ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 1 0  1 0 i i  i i m m  d d d d  I I I I  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
iiii = High order Nibble of Immediate Data Constant
IIII = Low order Nibble of Immediate Data
B = Source Register B (See table 2)
d = Destination register (See table 3)

The following table outlines the results returned for various constants using the IAND instruction

Imm Data	R1	Result R2	Result Carry
00	00	00	0
00	01	00	0
01	00	00	0
55	55	55	0
AA	AA	AA	0
80	01	00	0
80	FF	80	0
12	34	10	0
56	78	50	0
9A	BC	98	0
F0	DE	D0	0
02	FF	02	0
FF	FF	FF	0

IAND R2 ← (Immediate Data),R1

IDAC (Immediate Decimal Add with Carry registers)

Family Type: Immediate ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 1 1  0 0 i i  i i m m  d d d d  I I I I  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
iiii = High order Nibble of Immediate Data Constant
IIII = Low order Nibble of Immediate Data
B = Source Register B (See table 2)
d = Destination register (See table 3)

The following table outlines the results returned for various constants using the IDAC Instruction

IDAC R2 ((Immediate Data),R1

Imm Data	R1	Result CC R2	Result CC Carry	Result CS R2	Result CS Carry
00	00	00	0	01	0
00	01	01	0	02	0
01	00	01	0	02	0
55	55	10	1	11	1
AA	AA	BA	1	BB	1
80	01	81	0	82	0
80	FF	E5	1	E6	1
12	34	46	0	47	0
56	78	34	1	35	1
9A	BC	BC	1	BD	1
F0	DE	34	1	35	1
02	FF	67	1	68	1
FF	FF	54	1	55	1

IDSC (Immediate Decimal Subtract with Carry registers)

Family Type: Immediate ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 1 1  0 1 i i  i i m m  d d d d  I I I I  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
iiii = High order Nibble of Immediate Data Constant
IIII = Low order Nibble of Immediate Data
B = Source Register B (See table 2)
d = Destination register (See table 3)

The following table outlines the results returned for various constants using the IDSC Instruction

IDSC R2 < (Immediate Data),R1

Imm Data	R1	Result CC R2	Result CC Carry	Result CS R2	Result CS Carry
00	00	00	0	99	1
00	01	99	1	98	1
01	00	01	0	00	0
55	55	00	0	99	1
AA	AA	00	0	99	1
80	01	79	0	78	0
80	FF	2B	1	2A	1
12	34	78	1	77	1
56	78	78	1	77	1
9A	BC	78	1	77	1
F0	DE	1C	0	1B	0
02	FF	AD	1	AC	1
FF	FF	00	0	99	1

IMUL (Immediate Multiply Register with Constant)

Family Type: Immediate ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 1 1  1 1 0 0  c 0 m m  d d d d  I I I I  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
c = High or Low B nibble select
IIII = Immediate data to multiply by (Range of 0 to F)
B = Source Register B (See table 2)
d = Destination register (See table 3)

15 Nibble Control

0 Multiply Lower B by Lower A (ALBL)
1 Multiply Upper B by Lower A (ALBH)

The following table outlines the results returned for various constants using the IMUL instruction.

IMUL (ALBL) R2 ← (Immediate Data),R1

Imm Data	R1	Result R2	Carry	
0	00	00	0	
E	01	0E	0	
1	00	00	0	
5	55	19	0	
A	AA	64	0	
0	01	00	0	
0	FF	00	0	
2	34	08	0	
6	78	30	0	
9	BC	6C	0	
F	ED	C3	0	
2	FF	1E	0	
F	FF	E1	0	(Largest possible answer)

IOR (Immediate OR registers)

Family Type: Immediate ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 1 0  0 0 i i  i i m m  d d d d  I I I I  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
iiii = High order Nibble of Immediate Data Constant
IIII = Low order Nibble of Immediate Data
B = Source Register B (See table 2)
d = Destination register (See table 3)

The following table outlines the results returned for various constants using the IOR instruction

Imm Data	R1	Result R2	Carry
00	00	00	0
00	01	01	0
01	00	01	0
55	55	55	0
AA	AA	AA	0
80	01	81	0
80	FF	FF	0
12	34	36	0
56	78	7E	0
9A	BC	BE	0
F0	DE	FE	0
02	FF	FF	0
FF	FF	FF	0

IOR R2 ← (Immediate Data),R1

IXOR (Immediate Exclusive OR registers)

Family Type: Immediate ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 1 0  0 1 i i  i i m m  d d d d  I I I I  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
iiii = High order Nibble of Immediate Data Constant
IIII = Low order Nibble of Immediate Data
B = Source Register B (See table 2)
d = Destination register (See table 3)

The following table outlines the results returned for various constants using the IXOR instruction

Imm Data	R1	Result R2	Result Carry
00	00	00	0
00	01	01	0
01	00	01	0
55	55	00	0
AA	AA	00	0
80	01	81	0
80	FF	7F	0
12	34	26	0
56	78	2E	0
9A	BC	26	0
F0	DE	2E	0
02	FF	FD	0
FF	FF	00	0

IXOR R2 < (Immediate Data),R1

JMP (Jump)

Family Type: Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-----------------------------------------------------
p 1 0 1  1 1 x x  x x x x  x x x x  y y y y  y y 0 0
```

Where P = Parity Bit (Odd Parity)
x = LSD of Address
y = MSD of Address

This family of instructions allows the computer to alter the course of instruction execution by changing the address of micro-program store. Bit 22 of the Control word indicates that the instruction belongs to this family.

The JMP instruction allows the Wang to "jump" to any one of the 65,536 locations in Control Memory. Addresses greater than hex 8000 are located in PROM memory. The exception to this is on the new "C" chassis, where Control memory extends past 83FF. PROM Control Memory is executed at half-speed.

Address = yyyy yyxx xxxx xxxx

E.G. DE6A08 = JMP $0A6A

```
p 1 0 1  1 1 1 0  0 1 1 0  1 0 1 0  0 0 0 0  1 0 0 0
```

x = 10 0110 1010 = $26A
y = 00 0010 = $02

Combined = 0000 1010 0110 1010 ($0A6A)

Control is transferred to the new location, and no other CPU registers are altered in any way.

JSR (Jump to SubRoutine)

Family Type: Branch

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 1 0 1  0 1 x x  x x x x  x x x x  y y y y  y y 0 0
```

Where P = Parity Bit (Odd Parity)
x = LSD of Address
y = MSD of Address

Execution of a subroutine, similar to a GOSUB in BASIC, is performed by this instruction. The contents of the current memory address, plus 1, is stored on the internal system stack. A jump is executed to the desired location, and program execution commences. The format of the instruction is similar to the JMP instruction. Returning from a Subroutine call is accomplished by an RTS instruction.

```
E.G.  0400  D50108    JSR   $0901
      0401  ....


      0901  214E2F    SET   K < $52
      0902  87800F    RTS
```

Location 0401 would be placed on the stack, and the next instruction to be executed would be at $0901 in Control Memory.

The address of the stack is incremented by one. The level of nesting that is permitted for JSR instructions cannot exceed the depth of the stack, which is 96 locations. However, the stack is also used by BASIC to store temporary data, and in an MVP situation, the depth should never exceed ten items.

LPI (Load PH-PL Pair directly)

Family Type: Load

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-----------------------------------------------------
p 0 0 1  1 f f 1  f f m m  x x x x  x x x x  x x x x
```

Where P = Parity Bit (Odd Parity)
m = Memory Control bits (See table 4)
x = Lower twelve (12) bits of the address
f = Upper four (4) bits of the address

Fetching information from Data Memory or writing data to Data memory requires a pointer register in the Wang architecture. The pointer register is the PH - PL register pair, and can be directly loaded by this instruction.

The LPI instruction is an immediate load type and allows direct addressing of up to 65536 locations. Note that because of this limitation, bank boundaries must be observed, and no partition can overlap banks.

I.E. 192103 LPI $0103 ,W1

The executed LPI instruction will load the PH-PL pair with the hex value $103 and write the low order byte to Data memory.

It is of special note here that any LPI instruction containing the W1 or W2 bits will result in the destination location to be cleared out. This is a function of the LPI only.

Memory when read consists of two bytes, a high and low byte. The high byte is referred to as CH, while the low byte is referred to as the CL byte. When writing to data memory, W1 refers to the high byte, while W2 refers to the low byte. For clarity, remember that reading occurs in 16 bit, (two byte) segments, while writing occurs in 8 bit only segments.

The relationship of what is actually read or written when an LPI instruction is executed was extremely confusing to me at first. It becomes rather simple by showing examples. The following page will try to do that.

The PH PL register combination always points to a pair of memory locations. Remember that memory is read in 16 bit words. When a read command is executed, data at the actual location of PH-PL is loaded into the CH register, while data in the opposite number is read into the CL register.

As an example, if the following were in memory:

```
0102  22
0103  33

LPI $0102  ,RD    were executed, then CH = 22, CL = 33
```

However, let's now execute ...

```
LPI $0103  ,RD     ... now CH = 33 and CL =22.
```

Remember, even pair of bytes.

Now a similar function occurs with the write command, except we can only write one byte at a time. The ,W1 option will always write to the location pointed to by PHPL, while the ,W2 option writes to the opposite number.

```
LPI $0103  ,W1      Clears location $0103

LPI $0103  ,W2      Clears location $0102

LPI $0102  ,W2      Clears location $0103

LPI $0102  ,W1      Clears location $0102
```

MUL (Multiply Nibbles)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 1  1 1 0 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Multiply controls

15 14	Multiply Control
0 0	Multiply Lower B by Lower A (ALBL)
0 1	Multiply Lower B by Upper A (AHBL)
1 0	Multiply Upper B by Lower A (ALBH)
1 1	Multiply Upper B by Upper A (AHBH)

The following table outlines the results returned for various constants using the MUL instruction

		Result	
R0	R1	R2	Carry
00	00	00	0
00	01	00	0
01	00	00	0
55	55	19	0
AA	AA	64	0
80	01	00	0
80	FF	00	0
12	34	08	0
56	78	30	0
9A	BC	78	0
F0	DE	00	0
02	FF	1E	0
FF	FF	E1	0

1C0201 MUL (ALBL) R2 < R0,R1

MULX (Multiply Two four bit nibbles)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-----------------------------------------------------
p 0 0 1  1 1 1 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Multiply Control
0 0	Multiply Lower B by Lower A (ALBL)
0 1	Multiply Lower B by Upper A (AHBL)
1 0	Multiply Upper B by Lower A (ALBH)
1 1	Multiply Upper B by Upper A (AHBH)

The following table outlines the results returned for various constants using the MULX instruction

9E0402 MULX (ALBL) R4 < R0,R2

				Result		CC
R1	R0	R3	R2	R5	R4	Carry
00	00	00	00	00	00	0
00	00	00	01	00	00	0
00	01	00	00	00	00	0
55	AA	AA	55	32	32	0
11	22	33	44	03	08	0
44	55	66	77	18	23	0
80	00	00	01	00	00	0
00	01	80	00	00	00	0
9A	BC	CD	EF	82	B4	0
AA	AA	22	22	14	14	0
55	19	55	91	19	09	0

OR (Inclusive OR registers)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  0 0 0 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15	14	Carry Control
0	0	Normal, Initial Carry State not affected
0	1	Not Permitted (See SDC, SDCX makeups)
1	0	Clear Carry First ,CC
1	1	Set Carry first ,CS

The following table outlines the results returned for various constants using the OR instruction

		Result	
R0	R1	R2	Carry
00	00	00	0
00	01	01	0
01	00	01	0
55	55	55	0
AA	AA	AA	0
80	01	81	0
80	FF	FF	0
12	34	36	0
56	78	7E	0
9A	BC	BE	0
F0	DE	FE	0
02	FF	FF	0
FF	FF	FF	0

008201 OR R2 < R0,R1 ,CC

ORX (Inclusive OR registers 16 bit)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  0 0 1 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control
0 0	Normal, Initial Carry State not affected
0 1	Not Permitted (See SDC, SDCX makeups)
1 0	Clear Carry First ,CC
1 1	Set Carry first ,CS

The following table outlines the results returned for various constants using the ORX instruction

				Result		
R1	R0	R3	R2	R5	R4	Carry
00	00	00	00	00	00	0
00	00	00	01	00	01	0
00	01	00	00	00	01	0
55	AA	AA	55	FF	FF	0
11	22	33	44	33	66	0
44	55	66	77	66	77	0
80	00	00	01	80	01	0
00	01	80	00	80	01	0
9A	BC	CD	EF	DF	FF	0
AA	AA	22	22	AA	AA	0
55	19	55	91	55	99	0

```
828402   ORX  R4 < R0,R2  ,CC
```

RTS (Return from Sub-Routine)

Family: Stack Manipulation

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  0 1 1 1  1 0 m m  X c c 0  0 0 0 0  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See Table 4)
cc = Control Memory Control (See table 5)
B = B Register Selection (Used only during DM writes)
Normally set to hex F, (1111) if not writing
X = Don't Care

The RTS instruction is of the few Wang instructions that are not as straight forward as it would appear to be. This instruction is the only instruction that may Read or Write to Control Memory. When an RCM (Read Control Memory) or WCM (Write Control Memory) operation is requested, the system executes what is called a LOP, or Long OPeration.

A LOP causes the stack to be popped twice, and the resultant data sent to the Control Memory Address Register. The Read or Write operation is performed, and in the case of a Read operation the data goes to the K, PH and PL register. Data is read or written to Control Memory in 24 bit (3 byte) segments. The MSB is in K, while the LSB is in PL. If the operation has been a Write operation, the K register, PH and PL registers would be sent to the CM module and written.

```
------
! Note !
------
```

Note that the design of the hardware requires that the K register must be 1's complemented prior to writing.

Parity must be formed by the user. The Wang 2200 system checks the parity of every byte Read from the Control Memory that is a control instruction. That is, actually being executed. If a data word was read from Control Memory, the system does not check for parity.

RTS (Return from Sub-Routine) - Cont'd

Normal Usage of RTS instruction

```
0400  D50108    JSR   $0901
0401   ....

0901  214E2F   SET  K < $52
0902  87800F   RTS
```

Examples of Write to Control Memory:

A: Write 544F82 to Control Memory location 0001.

```
          214E4F    IOR  K < $54,00        Load K with $54
          9B0F82    LPI  $4F82            Load PH-PL with $4F82
          81800F    TPA  AR 00            Transfer PH-PL to AR 00
          990001    LPI  $0001            Load PH-PL with Address
          D7310C    JSR  WRITECM          Must be as a Sub-Routine
            ...      .....                Normal Return

WRITECM   05800F    TPS                   Push Address to Stack
          8B800F    TAP  AR 00            Bring Data Back to PHPL
          A7CEFE    IXOR K < $FF,K        1's Complement K
          078400    RTS  ,WC              Return from Subroutine and
                                          write K,PHPL to address 1.
```

Examples of Read from Control Memory

B: Read location $0FFE of Control Memory

```
0900  990FFE    LPI  $0FFE          Load PHPL with address
0901  D7313C    JSR  READCM         Goto Subroutine
0902  ...       .....               Data Now present in K PH PL


READCM    05800F    TPS             Transfer Address to stack
          878600    RTS   ,RC        Return and read data
```

SBC (Subtract Binary with Carry)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  1 1 0 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control
0 0	Normal, Initial Carry State not affected
0 1	Not Permitted (See SDC, SDCX makeups)
1 0	Clear Carry First ,CC
1 1	Set Carry first ,CS

The following table outlines the results returned for various constants using the SBC instruction

		Result CC		Result CS	
R0	R1	R2	Carry	R2	Carry
00	00	FF	0	00	1
00	01	FE	0	FF	0
01	00	00	1	01	1
55	55	FF	0	00	1
AA	AA	FF	0	00	1
80	01	7E	1	7F	1
80	FF	80	0	81	0
12	34	DD	0	DE	0
56	78	DD	0	DE	0
9A	BC	DD	0	DE	0
F0	DE	11	1	12	1
02	FF	02	0	03	0
FF	FF	FF	0	00	1

```
0C8201   SBC  R2 < R0,R1  ,CC
8CC201   SBC  R2 < R0,R1  ,CS
```

SBCX

SBCX (Subtract Binary with carry 16 bit)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  1 1 1 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control
0 0	Normal, Initial Carry State not affected
0 1	Not Permitted (See SDC, SDCX makeups)
1 0	Clear Carry First ,CC
1 1	Set Carry first ,CS

The following table outlines the results returned for various constants using the SBCX instruction

```
8E8402   SBCX  R4 < R0,R2  ,CC
0EC402   SBCX  R4 < R0,R2  ,CS
```

				Result		CC	Result		CS
R1	R0	R3	R2	R5	R4	Carry	R5	R4	Carry
00	00	00	00	FF	FF	0	00	00	1
00	00	00	01	FF	FE	0	FF	FF	0
00	01	00	00	00	00	1	00	01	1
55	AA	AA	55	AB	54	0	AB	55	0
11	22	33	44	DD	DD	0	DD	DE	0
44	55	66	77	DD	DD	0	DD	DE	0
80	00	00	01	7F	FE	1	7F	FF	1
00	01	80	00	80	00	0	80	00	1
9A	BC	CD	EF	CC	CC	0	CC	CD	0
AA	AA	22	22	88	87	1	88	88	1
55	19	55	91	FF	87	0	FF	88	0

SDC (Shift Decimal Characters)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  x x 0 0  0 1 m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Shift flag controls

19 18	Shift Control
0 0	Shift Lower B and Lower A (ALBL)
0 1	Shift Lower B and Upper A (AHBL)
1 0	Shift Upper B and Lower A (ALBH)
1 1	Shift Upper B and Upper A (AHBH)

When a SDC instruction occurs, the nibble pointed to by the B register and shift control combination will be transferred to the high order nibble of the destination register. The A register nibble selected by the Shift Control will be transferred to the low order of the destination register.

An example of the SDC instruction follows:

```
844402    SDC    AHBL    R2 < R0,R1
```

		Result	
R0	R1	R2	Carry
00	00	00	0
00	01	10	0
01	00	00	0
55	55	55	0
AA	AA	AA	0
80	01	18	0
80	FF	F8	0
12	34	41	0
56	78	85	0
9A	BC	C9	0
F0	DE	EF	0
02	FF	F0	0
FF	FF	FF	0

SDCX (Shift Decimal Characters - Extended)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  x x 1 0  0 1 m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Shift flag controls

19	18	Shift Control
0	0	Shift Lower B and Lower A (ALBL)
0	1	Shift Lower B and Upper A (AHBL)
1	0	Shift Upper B and Lower A (ALBH)
1	1	Shift Upper B and Upper A (AHBH)

When a SDCX instruction occurs, the nibble pointed to by the B register and Shift Control combination will be transferred to the high order nibble of the destination register. The A register nibble selected by the Shift Control will be transferred to the low order of the destination register.

An example of the extended SDCX instruction follows:

```
064402    SDCX   AHBL    R4 < R0,R2
```

				Result		
R1	R0	R3	R2	R5	R4	Carry
00	00	00	00	00	00	0
00	00	00	01	00	10	0
00	01	00	00	00	00	0
55	AA	AA	55	A5	5A	0
11	22	33	44	31	42	0
44	55	66	77	64	75	0
80	00	00	01	08	10	0
00	01	80	00	00	00	0
9A	BC	CD	EF	D9	FB	0
AA	AA	22	22	2A	2A	0
55	19	55	91	55	11	0

SET (Load Register with Constant)

Family Type: Immediate ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 1 0  0 0 i i  i i m m  d d d d  I I I I  1 1 1 1
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
iiii = High order Nibble of Immediate Data Constant
IIII = Low order Nibble of Immediate Data
d = Destination register (See table 3)

The following table outlines the results returned for various constants using the SET instruction

The SET instruction is nothing more than an IOR instruction with the B register gating set to all 0's. In this fashion, we are gating 00 with the Immediate Constant and setting the destination. Using this instruction is easier than to understand than its equivalent: IOR DD < $VV,00

Imm Data	Result R2	Carry
00	00	0
33	33	0
01	01	0
55	55	0
AA	AA	0
80	80	0
80	FF	0
12	12	0
56	56	0
9A	9A	0
F0	F0	0
02	02	0
FF	FF	0

SET R2 < (Immediate Data)

TAP (Transfer Auxiliary Register to PH-PL)

Family: Stack Manipulation

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  1 0 1 1  1 0 m m  X 0 0 r  r r r r  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See Table 4)
r = Source AR register (00 to 1F)
B = B Register Selection (Used only during DM writes)
Normally set to hex F, (1111) if not writing
X = Don't Care

The current contents of the selected AR register are transferred to the PH-PL pair. Note that the memory control bits function first. That is, the original PH-PL pair will be the address of Data Memory for any read or writes, not the new contents being read from the selected AR.

TPA (Transfer PH-PL to Auxiliary Register)

Family: Stack Manipulation

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  0 0 0 1  1 c m m  X c c r  r r r r  B B B B
```

Where P = Parity Bit (Odd Parity)
c = P register Control (See Table this section)
m = Data Memory Control (See Table 4)
r = Destination AR register (00 to 1F)
B = B Register Selection (Used only during DM writes)
Normally set to hex F, (1111) if not writing
X = Don't Care

P register control bits

When the PH-PL pair is transferred to the associated AR register, the system programmer may elect to increment or decrement the contents of PH-PL. The P register control bits gives the programmer a range of -3 to +3 for decrement or increment. The PH-PL pair is adjusted after transfer from the PH-PL register but prior to the entry into the Ar register. Thus, the original contents remains intact, and any Data memory Reads or Writes will always occur at the location originally pointed to by PH-PL.

Bit 14 10 9

0	0	0	No Effect
0	0	1	+1 to (PHPL) then store - PH-PL not affected
0	1	0	+2 " "
0	1	1	+3 " "
1	0	0	No Effect
1	0	1	-1 to (PHPL) then store - PH-PL not affected
1	1	0	-2 to (PHPL) then store - PH-PL not affected
1	1	1	-3 to (PHPL) then store - PH-PL not affected

TPS (Transfer PH-PL to System Stack)

Family: Stack Manipulation

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  0 1 0 1  1 c m m  X c c 0  0 0 0 0  B B B B
```

Where P = Parity Bit (Odd Parity)
c = P register Control (See Table this section)
m = Data Memory Control (See Table 4)
B = B Register Selection (Used only during DM writes)
Normally set to hex F, (1111) if not writing
X = Don't Care

P register control bits

When the PH-PL pair is transferred to the system stack, the system programmer may elect to increment or decrement the contents of PH-PL. The P register control bits gives the programmer a range of -3 to +3 for decrement or increment. The PH-PL pair is adjusted after transfer from the PH-PL register but prior to the entry into the system stack. Thus, the original contents remains intact, and any Data memory Reads or Writes will always occur at the location originally pointed to by PH-PL.

Bit 14 10 9

0	0	0	No Effect
0	0	1	+1 to (PHPL) then store - PH-PL not affected
0	1	0	+2 " "
0	1	1	+3 " "
1	0	0	No Effect
1	0	1	-1 to (PHPL) then store - PH-PL not affected
1	1	0	-2 to (PHPL) then store - PH-PL not affected
1	1	1	-3 to (PHPL) then store - PH-PL not affected

The user has no control over the stack address register. The user must be warned to remove data placed on the stack during execution of the routine. The stack will have been incremented twice (Two bytes) upon execution of this instruction.

TSP (Transfer System Stack to PH-PL)

Family: Stack Manipulation

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  1 1 0 1  1 0 m m  X 0 0 0  0 0 0 0  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See Table 4)
B = B Register Selection (Used only during DM writes)
Normally set to hex F, (1111) if not writing
X = Don't Care

The user has no control over the stack address register. The user must be warned to place data onto the stack prior to returning from a subroutine. The stack will have been decremented twice (Two bytes) upon execution of this instruction.

XOR (Exclusive OR registers)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  0 1 0 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15 14	Carry Control
0 0	Normal, Initial Carry State not affected
0 1	Not Permitted (See SDC, SDCX makeups)
1 0	Clear Carry First ,CC
1 1	Set Carry first ,CS

The following table outlines the results returned for various constants using the XOR instruction

		Result	
R0	R1	R2	Carry
00	00	00	0
00	01	01	0
01	00	01	0
55	55	00	0
AA	AA	00	0
80	01	81	0
80	FF	7F	0
12	34	26	0
56	78	2E	0
9A	BC	26	0
F0	DE	2E	0
02	FF	FD	0
FF	FF	00	0

```
848201   XOR  R2 < R0,R1  ,CC
```

XORX

XORX (Exclusive OR registers 16 bit)

Family Type: Register ALU

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  0 1 1 0  x x m m  d d d d  A A A A  B B B B
```

Where P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
A = Source Register A (See table 1)
B = Source Register B (See table 2)
d = Destination register (See table 3)
xx = Carry flag controls

15	14	Carry Control
0	0	Normal, Initial Carry State not affected
0	1	Not Permitted (See SDC, SDCX makeups)
1	0	Clear Carry First ,CC
1	1	Set Carry first ,CS

The following table outlines the results returned for various constants using the XORX instruction

R1	R0	R3	R2	Result R5	Result R4	Carry
00	00	00	00	00	00	0
00	00	00	01	00	01	0
00	01	00	00	00	01	0
55	AA	AA	55	FF	FF	0
11	22	33	44	22	66	0
44	55	66	77	22	22	0
80	00	00	01	80	01	0
00	01	80	00	80	01	0
9A	BC	CD	EF	57	53	0
AA	AA	22	22	88	88	0
55	19	55	91	00	88	0

```
068402   XORX  R4 < R0,R2  ,CC
```

XPA (Exchange PH-PL with Auxiliary Register)

Family: Stack Manipulation

```
2 2 2 2  1 1 1 1  1 1 1 1  1 1 0 0  0 0 0 0  0 0 0 0
3 2 1 0  9 8 7 6  5 4 3 2  1 0 9 8  7 6 5 4  3 2 1 0
-------------------------------------------------------
p 0 0 0  0 0 1 1  1 c m m  X c c r  r r r r  B B B B
```

Where P = Parity Bit (Odd Parity)
c = P register Control (See Table this section)
m = Data Memory Control (See Table 4)
r = Destination AR register (00 to 1F)
B = B Register Selection (Used only during DM writes)
Normally set to hex F, (1111) if not writing
X = Don't Care

P register control bits

When the PH-PL pair is transferred to the associated AR register, the system programmer may elect to increment or decrement the contents of PH-PL. The P register control bits gives the programmer a range of -3 to +3 for decrement or increment. The PH-PL pair is adjusted after transfer from the PH-PL register but prior to the entry into the Ar register. Thus, the original contents remains intact, and any Data memory Reads or Writes will always occur at the location originally pointed to by PH-PL.

Bit 14 10 9

0	0	0	No Effect
0	0	1	+1 to (PHPL) then store - PH-PL not affected
0	1	0	+2 " "
0	1	1	+3 " "
1	0	0	No Effect
1	0	1	-1 to (PHPL) then store - PH-PL not affected
1	1	0	-2 to (PHPL) then store - PH-PL not affected
1	1	1	-3 to (PHPL) then store - PH-PL not affected

Appendix i - General Bit Control Tables

Table 1 A Bus

7 6 5 4	
0 0 0 0	R0
0 0 0 1	R1
0 0 1 0	R2
0 0 1 1	R3
0 1 0 0	R4
0 1 0 1	R5
0 1 1 0	R6
0 1 1 1	R7
1 0 0 0	CL-
1 0 0 1	CH-
1 0 1 0	CL
1 0 1 1	CH
1 1 0 0	CL+
1 1 0 1	CH+
1 1 1 0	00+
1 1 1 1	00-

Table 2 B Bus

3 2 1 0	
0 0 0 0	R0
0 0 0 1	R1
0 0 1 0	R2
0 0 1 1	R3
0 1 0 0	R4
0 1 0 1	R5
0 1 1 0	R6
0 1 1 1	R7
1 0 0 0	PL
1 0 0 1	PH
1 0 1 0	CL
1 0 1 1	CH
1 1 0 0	SL
1 1 0 1	SH
1 1 1 0	K
1 1 1 1	0

Table 3 C Bus (Destination)

11 10 09 08	
0 0 0 0	R0
0 0 0 1	R1
0 0 1 0	R2
0 0 1 1	R3
0 1 0 0	R4
0 1 0 1	R5
0 1 1 0	R6
0 1 1 1	R7
1 0 0 0	PL
1 0 0 1	PH
1 0 1 0	ILLEGAL
1 0 1 1	ILLEGAL
1 1 0 0	SL
1 1 0 1	SH
1 1 1 0	K
1 1 1 1	DUM

All A register sources with a + after the mnemonics cause the PH-PL pair to be incremented by +1 after the operation. All A register sources with a - after the mnemonics causes the PH-PL pair to be decremented by 1 after the operation.

Table 4 Memory control Data

13 12	
0 0	No Op
0 1	,RD Read
1 0	,W1 Write Byte at current PHPL
1 1	,W2 Write Byte at opposite PHPL pointed to pair

Table 5 Control memory

10 09		
0 0	No Op	
0 1	No Op	
1 0	,WC	Write CM
1 1	,RC	Read CM

Appendix ii - Conditional Branch Examples

A reg		B reg		Instruction									
R3	R2	R5	R4	BNR		BER		BLER		BLR		BLRX	BLEX
00	00	00	00	n		y		y		n		n	y
80	00	00	00	y		n		n		n		n	n
00	00	80	00	y		n		y		y		y	y
FF	00	00	00	y		n		n		n		n	n
00	00	FF	00	y		n		y		y		y	y
33	00	55	00	y		n		y		y		y	y
55	00	33	00	y		n		n		n		n	n
80	00	85	00	y		n		y		y		y	y
85	00	80	00	y		n		n		n		n	n

In all the example above, the eight bit comparisons are defined as being the form:

```
Bxxx  R3,R5
```

Whereas the 16 bit comparisons are:

```
Bxxx  R2,R4
```

Note that the above examples are not signed numbers. The magnitude is treated as an unsigned positive number in the range of 0 to 255 for eight bit comparisons, and 0 to 65535 for 16 bit comparisons.

Appendix iii Alphabetical Listing of Mnemonics

	2 2 2 2	1 1 1 1	1 1 1 1	1 1 0 0	0 0 0 0	0 0 0 0
	3 2 1 0	9 8 7 6	5 4 3 2	1 0 9 8	7 6 5 4	3 2 1 0
ADC	p 0 0 1	1 0 0 0	x x m m	d d d d	A A A A	B B B B
ADCX	p 0 0 1	1 0 1 0	x x m m	d d d d	A A A A	B B B B
AND	p 0 0 0	1 0 0 0	x x m m	d d d d	A A A A	B B B B
ANDX	p 0 0 0	1 0 1 0	x x m m	d d d d	A A A A	B B B B
BEH	p 1 1 1	0 1 x x	x x x x	x x x x	c c c c	B B B B
BEL	p 1 1 1	0 0 x x	x x x x	x x x x	c c c c	B B B B
BER	p 1 0 1	0 0 x x	x x x x	x x x x	A A A A	B B B B
BEZ	p 1 0 1	0 0 x x	x x x x	x x x x	A A A A	1 1 1 1
BFH	p 1 1 0	1 1 x x	x x x x	x x x x	c c c c	B B B B
BFL	p 1 1 0	1 0 x x	x x x x	x x x x	c c c c	B B B B
BLER	p 1 0 0	1 0 x x	x x x x	x x x x	A A A A	B B B B
BLEX	p 1 0 0	1 1 x x	x x x x	x x x x	A A A A	B B B B
BLR	p 1 0 0	0 0 x x	x x x x	x x x x	A A A A	B B B B
BLRX	p 1 0 0	0 1 x x	x x x x	x x x x	A A A A	B B B B
BNH	p 1 1 1	1 1 x x	x x x x	x x x x	c c c c	B B B B
BNL	p 1 1 1	1 0 x x	x x x x	x x x x	c c c c	B B B B
BNR	p 1 0 1	1 0 x x	x x x x	x x x x	A A A A	B B B B
BNZ	p 1 0 1	1 0 x x	x x x x	x x x x	A A A A	1 1 1 1
BTH	p 1 1 0	0 1 x x	x x x x	x x x x	c c c c	B B B B
BTL	p 1 1 0	0 0 x x	x x x x	x x x x	c c c c	B B B B
CIO	p 0 0 1	0 1 1 1	1 0 m m	y y y y	y d d d	z z d d
DAC	p 0 0 1	0 0 0 0	x x m m	d d d d	A A A A	B B B B
DACX	p 0 0 1	0 0 1 0	x x m m	d d d d	A A A A	B B B B
DSC	p 0 0 1	0 1 0 0	x x m m	d d d d	A A A A	B B B B
DSCX	p 0 0 1	0 1 1 0	x x m m	d d d d	A A A A	B B B B
IADC	p 0 1 1	1 0 i i	i i m m	d d d d	I I I I	B B B B
IADD	p 0 1 0	1 1 i i	i i m m	d d d d	I I I I	B B B B
IAND	p 0 1 0	1 0 i i	i i m m	d d d d	I I I I	B B B B
IDAC	p 0 1 1	0 0 i i	i i m m	d d d d	I I I I	B B B B
IDSC	p 0 1 1	0 1 i i	i i m m	d d d d	I I I I	B B B B
IMUL	p 0 1 1	1 1 0 0	c 0 m m	d d d d	I I I I	B B B B
IOR	p 0 1 0	0 0 i i	i i m m	d d d d	I I I I	B B B B
IXOR	p 0 1 0	0 1 i i	i i m m	d d d d	I I I I	B B B B

Appendix iii Alphabetical Listing of Mnemonics - Cont'd

	2 2 2 2	1 1 1 1	1 1 1 1	1 1 0 0	0 0 0 0	0 0 0 0
	3 2 1 0	9 8 7 6	5 4 3 2	1 0 9 8	7 6 5 4	3 2 1 0
JMP	p 1 0 1	1 1 x x	x x x x	x x x x	y y y y	y y 0 0
JSR	p 1 0 1	0 1 x x	x x x x	x x x x	y y y y	y y 0 0
LPI	p 0 0 1	1 f f 1	f f m m	x x x x	x x x x	x x x x
MUL	p 0 0 1	1 1 0 0	x x m m	d d d d	A A A A	B B B B
MULX	p 0 0 1	1 1 1 0	x x m m	d d d d	A A A A	B B B B
OR	p 0 0 0	0 0 0 0	x x m m	d d d d	A A A A	B B B B
ORX	p 0 0 0	0 0 1 0	x x m m	d d d d	A A A A	B B B B
RTS	p 0 0 0	0 1 1 1	1 0 m m	X c c 0	0 0 0 0	B B B B
SBC	p 0 0 0	1 1 0 0	x x m m	d d d d	A A A A	B B B B
SBCX	p 0 0 0	1 1 1 0	x x m m	d d d d	A A A A	B B B B
SDC	p 0 0 0	x x 0 0	0 1 m m	d d d d	A A A A	B B B B
SDCX	p 0 0 0	x x 1 0	0 1 m m	d d d d	A A A A	B B B B
SET	p 0 1 0	0 0 i i	i i m m	d d d d	I I I I	1 1 1 1
TAP	p 0 0 0	1 0 1 1	1 0 m m	X 0 0 r	r r r r	B B B B
TPA	p 0 0 0	0 0 0 1	1 c m m	X c c r	r r r r	B B B B
TPS	p 0 0 0	0 1 0 1	1 c m m	X c c 0	0 0 0 0	B B B B
TSP	p 0 0 0	1 1 0 1	1 0 m m	X 0 0 0	0 0 0 0	B B B B
XOR	p 0 0 0	0 1 0 0	x x m m	d d d d	A A A A	B B B B
XORX	p 0 0 0	0 1 1 0	x x m m	d d d d	A A A A	B B B B
XPA	p 0 0 0	0 0 1 1	1 c m m	X c c r	r r r r	B B B B

Appendix iv - Numerical Listing of Mnemonics

	2 2 2 2	1 1 1 1	1 1 1 1	1 1 0 0	0 0 0 0	0 0 0 0
	3 2 1 0	9 8 7 6	5 4 3 2	1 0 9 8	7 6 5 4	3 2 1 0
OR	p 0 0 0	0 0 0 0	x x m m	d d d d	A A A A	B B B B
SDC	p 0 0 0	x x 0 0	0 1 m m	d d d d	A A A A	B B B B
TPA	p 0 0 0	0 0 0 1	1 c m m	X c c r	r r r r	B B B B
ORX	p 0 0 0	0 0 1 0	x x m m	d d d d	A A A A	B B B B
SDCX	p 0 0 0	x x 1 0	0 1 m m	d d d d	A A A A	B B B B
XPA	p 0 0 0	0 0 1 1	1 c m m	X c c r	r r r r	B B B B
XOR	p 0 0 0	0 1 0 0	x x m m	d d d d	A A A A	B B B B
TPS	p 0 0 0	0 1 0 1	1 c m m	X c c 0	0 0 0 0	B B B B
XORX	p 0 0 0	0 1 1 0	x x m m	d d d d	A A A A	B B B B
RTS	p 0 0 0	0 1 1 1	1 0 m m	X c c 0	0 0 0 0	B B B B
AND	p 0 0 0	1 0 0 0	x x m m	d d d d	A A A A	B B B B
ANDX	p 0 0 0	1 0 1 0	x x m m	d d d d	A A A A	B B B B
TAP	p 0 0 0	1 0 1 1	1 0 m m	X 0 0 r	r r r r	B B B B
SBC	p 0 0 0	1 1 0 0	x x m m	d d d d	A A A A	B B B B
TSP	p 0 0 0	1 1 0 1	1 0 m m	X 0 0 0	0 0 0 0	B B B B
SBCX	p 0 0 0	1 1 1 0	x x m m	d d d d	A A A A	B B B B
DAC	p 0 0 1	0 0 0 0	x x m m	d d d d	A A A A	B B B B
DACX	p 0 0 1	0 0 1 0	x x m m	d d d d	A A A A	B B B B
DSC	p 0 0 1	0 1 0 0	x x m m	d d d d	A A A A	B B B B
DSCX	p 0 0 1	0 1 1 0	x x m m	d d d d	A A A A	B B B B
CIO	p 0 0 1	0 1 1 1	1 0 m m	y y y y	y d d d	z z d d
ADC	p 0 0 1	1 0 0 0	x x m m	d d d d	A A A A	B B B B
ADCX	p 0 0 1	1 0 1 0	x x m m	d d d d	A A A A	B B B B
MUL	p 0 0 1	1 1 0 0	x x m m	d d d d	A A A A	B B B B
MULX	p 0 0 1	1 1 1 0	x x m m	d d d d	A A A A	B B B B
LPI	p 0 0 1	1 f f 1	f f m m	x x x x	x x x x	x x x x
IOR	p 0 1 0	0 0 i i	i i m m	d d d d	I I I I	B B B B
SET	p 0 1 0	0 0 i i	i i m m	d d d d	I I I I	1 1 1 1
IXOR	p 0 1 0	0 1 i i	i i m m	d d d d	I I I I	B B B B
IAND	p 0 1 0	1 0 i i	i i m m	d d d d	I I I I	B B B B
IADD	p 0 1 0	1 1 i i	i i m m	d d d d	I I I I	B B B B
IDAC	p 0 1 1	0 0 i i	i i m m	d d d d	I I I I	B B B B
IDSC	p 0 1 1	0 1 i i	i i m m	d d d d	I I I I	B B B B
IADC	p 0 1 1	1 0 i i	i i m m	d d d d	I I I I	B B B B
IMUL	p 0 1 1	1 1 0 0	c 0 m m	d d d d	I I I I	B B B B

Appendix iv - Numerical Listing of Mnemonics - Cont'd

	2 2 2 2	1 1 1 1	1 1 1 1	1 1 0 0	0 0 0 0	0 0 0 0
	3 2 1 0	9 8 7 6	5 4 3 2	1 0 9 8	7 6 5 4	3 2 1 0
BLR	p 1 0 0	0 0 x x	x x x x	x x x x	A A A A	B B B B
BLRX	p 1 0 0	0 1 x x	x x x x	x x x x	A A A A	B B B B
BLER	p 1 0 0	1 0 x x	x x x x	x x x x	A A A A	B B B B
BLEX	p 1 0 0	1 1 x x	x x x x	x x x x	A A A A	B B B B
BER	p 1 0 1	0 0 x x	x x x x	x x x x	A A A A	B B B B
BEZ	p 1 0 1	0 0 x x	x x x x	x x x x	A A A A	1 1 1 1
JSR	p 1 0 1	0 1 x x	x x x x	x x x x	y y y y	y y 0 0
BNR	p 1 0 1	1 0 x x	x x x x	x x x x	A A A A	B B B B
BNZ	p 1 0 1	1 0 x x	x x x x	x x x x	A A A A	1 1 1 1
JMP	p 1 0 1	1 1 x x	x x x x	x x x x	y y y y	y y 0 0
BTL	p 1 1 0	0 0 x x	x x x x	x x x x	c c c c	B B B B
BTH	p 1 1 0	0 1 x x	x x x x	x x x x	c c c c	B B B B
BFL	p 1 1 0	1 0 x x	x x x x	x x x x	c c c c	B B B B
BFH	p 1 1 0	1 1 x x	x x x x	x x x x	c c c c	B B B B
BEL	p 1 1 1	0 0 x x	x x x x	x x x x	c c c c	B B B B
BEH	p 1 1 1	0 1 x x	x x x x	x x x x	c c c c	B B B B
BNL	p 1 1 1	1 0 x x	x x x x	x x x x	c c c c	B B B B
BNH	p 1 1 1	1 1 x x	x x x x	x x x x	c c c c	B B B B

1 =
2 <>
3 <=
4 <

[6] = ABSOLUTE or Relative
ABSOFF [1] =
ABSOFF [2] =
[3] = which MNEMONIC Set
6,59
8,59