Wang 2200 Instruction Set

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

Revised 4/26/1983
Wang 2200 Instruction Set - Index

1.0 Overall Description of the Wang 2200 1
1.1 Internal Register Structure - Description 1
1.2 Status Register SL description 4
1.3 Hardware Status Register (SH) description 5
2.0 General Instruction Breakdowns 6
2.1 Parity Bits 6
2.2 Classes of Instructions 7
2.2.1 Branch Instructions 7
2.2.2 Masked Branch Instructions 8
2.2.3 Valued Branch Instructions 9
2.2.4 Register Comparison Instructions 10
2.2.5 Register ALU Instructions 11
2.2.6 Immediate Data ALU instructions 14
2.2.7 Peripheral Control 16
2.2.8 Load Data Memory Pointer 16
2.2.9 Stack and Auxiliary register manipulations 17
2.2.9.1 Auxiliary Register Manipulation 17
2.2.9.2 Stack Manipulation Instructions 19
3.0 Data Memory Read/Write operations 20

Specific Instruction Mnemonics 24-79

Appendix 1 General Bit Control Tables
Table 1 A Register Bus gating 80
Table 2 B Register Bus gating 80
Table 3 C (Destination) Bus gating 80
Table 4 Data Memory Control Bits 80
Table 5 Control Memory Bits 80

Appendix ii Conditional Branch Instruction examples 81
Appendix iii Alphabetical listing of Mnemonics 82
Appendix iv Numerical Listing of Mnemonics 84

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed
written permission of Computer Concepts Corporation
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.
## Wang 2200 Machine Instruction Set

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>General Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>Special Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Auxiliary Registers

<table>
<thead>
<tr>
<th>AR 00</th>
<th>AR 01</th>
<th>AR 02</th>
<th>AR 03</th>
<th>AR 04</th>
<th>AR 05</th>
<th>AR 06</th>
<th>AR 07</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AR 08</th>
<th>AR 09</th>
<th>AR 0A</th>
<th>AR 0B</th>
<th>AR 0C</th>
<th>AR 0D</th>
<th>AR 0E</th>
<th>AR 0F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AR 10</th>
<th>AR 11</th>
<th>AR 12</th>
<th>AR 13</th>
<th>AR 14</th>
<th>AR 15</th>
<th>AR 16</th>
<th>AR 17</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AR 18</th>
<th>AR 19</th>
<th>AR 1A</th>
<th>AR 1B</th>
<th>AR 1C</th>
<th>AR 1D</th>
<th>AR 1E</th>
<th>AR 1F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Special Purpose

<table>
<thead>
<tr>
<th>K</th>
<th>DUM</th>
<th>Programming Model for Wang 2200 Series</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

\[
\begin{array}{cccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 0 & 0 & x & x & x & x & x \\
0 & 1 & 0 & x & x & x & x & x \\
1 & 0 & 0 & x & x & x & x & x \\
1 & 1 & 0 & x & x & x & x & x \\
0 & 0 & 1 & x & x & x & x & x \\
0 & 1 & 1 & x & x & x & x & x \\
1 & 0 & 1 & x & x & x & x & x \\
1 & 1 & 1 & x & x & x & x & x \\
0 & 0 & 0 & x & x & x & x & 0 \\
0 & 1 & 0 & x & x & x & x & 0 \\
1 & 0 & 0 & x & x & x & x & 0 \\
1 & 1 & 0 & x & x & x & x & 0 \\
0 & 0 & 1 & x & x & x & x & 0 \\
0 & 1 & 1 & x & x & x & x & 0 \\
1 & 0 & 1 & x & x & x & x & 0 \\
1 & 1 & 1 & x & x & x & x & 0 \\
0 & 0 & 0 & x & x & x & x & 1 \\
0 & 1 & 0 & x & x & x & x & 1 \\
1 & 0 & 0 & x & x & x & x & 1 \\
1 & 1 & 0 & x & x & x & x & 1 \\
0 & 0 & 1 & x & x & x & x & 1 \\
0 & 1 & 1 & x & x & x & x & 1 \\
1 & 0 & 1 & x & x & x & x & 1 \\
1 & 1 & 1 & x & x & x & x & 1 \\
\end{array}
\]

Select First 64K Bank
Select Second 64K Bank
Select Third 64K Bank
Select Fourth 64K Bank
Select Fifth 64K Bank
Select Sixth 64K Bank
Select Seventh 64K Bank
Select Eighth 64K Bank
Atomize Pass
Assign Variables Pass
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 unwieldy. 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.

```
7 6 5 4 3 2 1 0

7 6 5 4 3 2 1 0

Carry Bit for ALU operations
CPb On output. IBS on input
SF key depressed (IB9-)
READY/-BUSY from peripheral
Partition Timeout On Shot
HALT/STEP key
PEDM. A parity error has occurred
in Data Memory
DMPI-. Inhibit PEDM.
```

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 1. The external device seeing this, places data on the Computer's 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 mnemonics 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 jump 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 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 l o 1 v 1 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

O = Jump to Subroutine

l = 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:

```
2222 1111 1111 1100 0000 0000
3210 9876 5432 1098 7654 3210
```

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, RO 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
  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 i l l f f 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 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 l o f f f x x x x x x x x x x x x x A A A 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 \((A \neq B)\)
- **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 mathematical data to be added, subtracted, and/or processed must be handled 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.

\[
\begin{align*}
\text{2222211111111100000000000} \\
\text{321098765432109876543210} \\
\hline
\text{p00aav0xxxmmdddddAAAA BBBBB}
\end{align*}
\]

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 \( v = 1 \)
- \( xx \) = Carry flag controls

\[1514 \text{ Carry Control}\]

- 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:

<table>
<thead>
<tr>
<th>ALU Codes</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>OR</td>
</tr>
<tr>
<td>0 0 1</td>
<td>XOR</td>
</tr>
<tr>
<td>0 1 0</td>
<td>AND</td>
</tr>
<tr>
<td>0 1 1</td>
<td>SBC</td>
</tr>
<tr>
<td>1 0 0</td>
<td>DAC</td>
</tr>
<tr>
<td>1 0 1</td>
<td>DSC</td>
</tr>
<tr>
<td>1 1 0</td>
<td>ADC</td>
</tr>
<tr>
<td>1 1 1</td>
<td>MUL</td>
</tr>
</tbody>
</table>

OR the contents of B with A
Exclusive OR the contents of B with A
Logical AND of B with A
Binary Subtract with Carry, \( A - B \)
Decimal Add with Carry
Decimal Subtract with Carry, \( A - B \)
Binary Add with Carry
Multiply two 4 bit values

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

11
Multiply is similar to the Immediate register multiply. Bits 14 and 15 of the instruction determine which two nibbles will be multiplied:

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Multiply Lower B by Lower A ALBL</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Multiply Lower B by Upper A AHBL</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Multiply Upper B by Lower A ALBH</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Multiply Upper B by Upper A ANBH</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>18</th>
<th>19</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>B Lower 4 ORED A lower 4 (B nibble is always the MSB)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>B lower 4 ORED A upper 4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>B upper 4 ORED A lower 4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>B upper 4 ORED A upper 4</td>
</tr>
</tbody>
</table>
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
94C22F 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
0CB22F SBC R2 < R2,00 ,CC R2 = FF, Carry = 0 (See!)
8CC223 SBC R2 < R2,R3 ,CS R2 = CC, Carry = 1
0CB223 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 to. Different instructions utilize it in a different manner than one would suspect.

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
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
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 o i 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:

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

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. 2BC2FO   IAND R2 < $FO,RO

Loads the Register #2 with the contents of Register #0 logically *anded* by the constant $FO.

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.

<table>
<thead>
<tr>
<th>15</th>
<th>16</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>Multiply B lower 4 by Constant lower 4 ALBL</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>upper * * * lower * ALBH</td>
</tr>
</tbody>
</table>
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 sub-routine calls, as well as holding temporary data. The author believes that the AR registers are vestigial 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

```
2222 1111 1111 1100 0000 0000
3210 9876 5432 1098 7654 3210
---------------------------------------------------------------
p0000 fff1 1cm m Xcrrrrr r 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

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Effect</td>
</tr>
<tr>
<td>0.1</td>
<td>PHPL then store - PH-PL not affected</td>
</tr>
<tr>
<td>0.1</td>
<td>PHPL then store - PH-PL not affected</td>
</tr>
<tr>
<td>1</td>
<td>No Effect</td>
</tr>
<tr>
<td>1.0</td>
<td>PHPL then store - PH-PL not affected</td>
</tr>
<tr>
<td>1.0</td>
<td>PHPL then store - PH-PL not affected</td>
</tr>
<tr>
<td>1.1</td>
<td>PHPL then store - PH-PL not affected</td>
</tr>
</tbody>
</table>

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 allow 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 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
```

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 straightforward 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.

---

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

19
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:

<table>
<thead>
<tr>
<th>13</th>
<th>12</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>No Data Memory Operation is to be performed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>,RD Read Contents of Data Memory</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>,W1 Write byte at current PHPL</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>,W2 Write byte at current PHPL XOR 1</td>
</tr>
</tbody>
</table>

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 RO < 0 ,RD

CH would contain 12, CL would contain 34

**PHPL = 0543**  Data at 0542 = 1234

SET RO < 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.

---

Copyright © 1982,1983 by Computer Concepts Corporation  
Shawnee Mission, KS  
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
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 \( W_1 \) option was chosen, the RESULT of the mathematical operation is sent to the memory at the current position of the PHPL pair.

If the \( W_2 \) 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.

\[
\text{PHPL} = 0542 \quad \text{Contents of 0542} = 1234
\]

- \( \text{SET RO < $53, } W_1 \)
  - The new contents of 0542 would be 5334
- \( \text{IAND R2 < $12, RO, } W_2 \)
  - The new contents of 0542 would be 5312

\[
\text{PHPL} = 0543 \quad \text{Contents of 0542} = 5312
\]

- \( \text{IADD RO < $20, RO, } W_1 \)
  - The new contents of 0542 would be 5373
- \( \text{IOR R2 < $F0, RO, } W_2 \)
  - 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:

\[
\text{TPA AR 00, } W_1, R5
\]

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

\[
\text{Copyright } \odot 1982,1983 \text{ by Computer Concepts Corporation}
\]

Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation.
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+,RO ,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

<table>
<thead>
<tr>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>!</th>
<th>!</th>
<th>Result CC</th>
<th>!</th>
<th>Result CS</th>
<th>!</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>RO</td>
<td>R1</td>
<td>R2</td>
<td>Carry</td>
<td>R2</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>0</td>
<td>02</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>0</td>
<td>02</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>AA</td>
<td>0</td>
<td>AB</td>
<td>0</td>
</tr>
<tr>
<td>AA</td>
<td>AA</td>
<td>54</td>
<td>1</td>
<td>55</td>
<td>1</td>
</tr>
<tr>
<td>80</td>
<td>01</td>
<td>81</td>
<td>0</td>
<td>82</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
<td>7F</td>
<td>1</td>
<td>80</td>
<td>F</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>46</td>
<td>0</td>
<td>47</td>
<td>0</td>
</tr>
<tr>
<td>56</td>
<td>78</td>
<td>CE</td>
<td>0</td>
<td>CF</td>
<td>0</td>
</tr>
<tr>
<td>9A</td>
<td>BC</td>
<td>56</td>
<td>1</td>
<td>57</td>
<td>1</td>
</tr>
<tr>
<td>F0</td>
<td>DE</td>
<td>CE</td>
<td>1</td>
<td>CF</td>
<td>1</td>
</tr>
<tr>
<td>02</td>
<td>FF</td>
<td>01</td>
<td>1</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>FB</td>
<td>1</td>
<td>FF</td>
<td>1</td>
</tr>
</tbody>
</table>

Copyright © 1982,1983 by Computer Concepts Corporation

Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
ADCX (Binary Add with carry 16 bit)

Family Type: Register ALU

<table>
<thead>
<tr>
<th>P</th>
<th>m</th>
<th>A</th>
<th>B</th>
<th>d</th>
<th>xx</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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
- 1 1 Set Carry first

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 |

<table>
<thead>
<tr>
<th>R1 R0</th>
<th>R3 R2</th>
<th>R5 R4</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>0</td>
</tr>
<tr>
<td>00 00</td>
<td>00 01</td>
<td>00 01</td>
<td>0</td>
</tr>
<tr>
<td>00 01</td>
<td>00 00</td>
<td>00 01</td>
<td>0</td>
</tr>
<tr>
<td>00 01</td>
<td>00 00</td>
<td>00 01</td>
<td>0</td>
</tr>
<tr>
<td>55 AA</td>
<td>AA 55</td>
<td>FF FF</td>
<td>0</td>
</tr>
<tr>
<td>11 22</td>
<td>33 44</td>
<td>44 66</td>
<td>0</td>
</tr>
<tr>
<td>44 55</td>
<td>66 77</td>
<td>AA CC</td>
<td>0</td>
</tr>
<tr>
<td>80 00</td>
<td>00 01</td>
<td>80 01</td>
<td>0</td>
</tr>
<tr>
<td>00 01</td>
<td>80 00</td>
<td>80 01</td>
<td>0</td>
</tr>
<tr>
<td>9A BC</td>
<td>CD EF</td>
<td>6B AB</td>
<td>1</td>
</tr>
<tr>
<td>AA AA</td>
<td>22 22</td>
<td>CC CC</td>
<td>0</td>
</tr>
<tr>
<td>55 19</td>
<td>55 91</td>
<td>AA AA</td>
<td>0</td>
</tr>
</tbody>
</table>

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
AND (Logical AND registers)

Family Type: Register ALU

\[
\begin{array}{cccccccccccccccc}
2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
\end{array}
\]

p 0 0 0 0 1 0 0 0 x x m m d d d d A A A A B B B B

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

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:

<table>
<thead>
<tr>
<th>Result</th>
<th>RO</th>
<th>R1</th>
<th>R2</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0001</td>
<td>0 0</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0100</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1111</td>
<td>0 0</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1000</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0110</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1010</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0000</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0001</td>
<td>0 0</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0100</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1111</td>
<td>0 0</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1000</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0110</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1010</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0000</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0001</td>
<td>0 0</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0100</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1111</td>
<td>0 0</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1000</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0110</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1010</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
</tbody>
</table>

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
ANDX (Logical AND registers 16 bit)

Family Type: Register ALU

```
2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 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
```

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:

<table>
<thead>
<tr>
<th>R1</th>
<th>R0</th>
<th>R3</th>
<th>R2</th>
<th>R5</th>
<th>R4</th>
<th>Carry</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0 0</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>5 5</td>
<td>A A</td>
<td>5 5</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>2 2</td>
<td>3 3</td>
<td>4 4</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>4 4</td>
<td>5 5</td>
<td>6 6</td>
<td>7 7</td>
<td>4 4</td>
<td>5 5</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>8 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0 0</td>
<td>0 1</td>
<td>8 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>9 A</td>
<td>B C</td>
<td>C D</td>
<td>E F</td>
<td>8 8</td>
<td>A C</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>A A</td>
<td>A A</td>
<td>2 2</td>
<td>2 2</td>
<td>2 2</td>
<td>2 2</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>5 5</td>
<td>1 9</td>
<td>5 5</td>
<td>9 1</td>
<td>5 5</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
BEH (Branch if Equal, High nibble)

Family Type: Valued Branch

```
2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 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
```

Where:
- \( P \) = Parity Bit (Odd Parity)
- \( x \) = Map location to Branch to
- \( c \) = 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 RO <$01 Not Taken
6002 DC0050 JMP $5000
6003 20000F SET RO <$00 Branch Taken
6004 DC0050 JMP $5000
```

Branch will be taken if nibble is equal to 3x
BEL (Branch if Equal, Low nibble)

Family Type: Valued Branch

```
2 2 2 1 1 1 1 1 1 1 1 1 1 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
```

Where:  
- P = Parity Bit (Odd Parity)
- x = Map location to Branch to
- 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 RO $01 Not Taken
6002 DC0050 JMP $5000 Not Taken
6003 20000F SET RO $00 Branch Taken
6004 DC0050 JMP $5000
```

Branch will be taken if nibble is equal to x5

Copyright © 1982, 1983 by Computer Concepts Corporation  
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
BER (Branch if A = B)

Family Type: Register Comparison Branch

```
  2 2 2 2 1 1 1 1 1 1 1 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 l o 1 0 0 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:

<table>
<thead>
<tr>
<th>R3</th>
<th>R6</th>
<th>Branch</th>
<th>D20D36</th>
<th>BER R3,R6</th>
<th>TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>05</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>8F</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>02</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>80</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>83</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>DE</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
BEZ (Branch if A = Zero (0))

Family Type: Register Comparison Branch

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 syntactically 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:

<table>
<thead>
<tr>
<th>R3</th>
<th>Branch</th>
<th>D20D3F</th>
<th>BEZ</th>
<th>R3</th>
<th>TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>!</td>
<td>!</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>!</td>
<td>x</td>
<td>!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>!</td>
<td>x</td>
<td>!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>!</td>
<td>x</td>
<td>!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>!</td>
<td>x</td>
<td>!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>!</td>
<td>x</td>
<td>!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>!</td>
<td>x</td>
<td>!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>!</td>
<td>x</td>
<td>!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>!</td>
<td>x</td>
<td>!</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
BFH  (Branch if Bits False, High nibble)

Family Type: Masked Branch

2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 3 2 1 0 9 8 7 6 5 4 3 2 1 0
-----------------------------------------------

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:

\[ \text{\$ORG \$6000} \]

\begin{verbatim}
6000 6C0352 BFH \$05,R2 \$6003
6001 A0001F SET RO < \$01  Not Taken
6002 DC0050 JMP \$5000
6003 20000F SET RO < \$00  Branch Taken
6004 DC0050 JMP \$5000
\end{verbatim}

Branch will be taken if nibbles are Ox,2x,8x or Ax.
BFL (Branch if Bits False, Low nibble)

Family Type: Masked Branch

```
2222 1111 1111 1100 0000 0000
3210 9876 5432 1098 7654 3210
```

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

Branch will be taken if nibbles are xo,x4,x8 or xc.
### BLER (Branch if \( A \leq B \))

**Family Type:** Register Comparison Branch

```
2 2 2 2 1 1 1 1 1 0 0 0 0 0 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 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:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Branch</th>
<th>CAOD36</th>
<th>BLER R3,R6 TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>X</td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>X</td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>05</td>
<td>!</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>80</td>
<td>8F</td>
<td>!</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>80</td>
<td>02</td>
<td>!</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>02</td>
<td>80</td>
<td>!</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>FF</td>
<td>83</td>
<td>!</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>DE</td>
<td>DE</td>
<td>X</td>
<td>!</td>
<td>!</td>
</tr>
</tbody>
</table>

---

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

34
BLEX (Branch if A ≤ B 16 bit test)

Family Type: Register Comparison Branch

<table>
<thead>
<tr>
<th>R3</th>
<th>R2</th>
<th>R6</th>
<th>R5</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>00</td>
<td>10</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>00</td>
<td>05</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>05</td>
<td>00</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>05</td>
<td>00</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>80</td>
<td>8F</td>
<td>80</td>
<td>8F</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>80</td>
<td>00</td>
<td>8F</td>
<td>00</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>80</td>
<td>00</td>
<td>00</td>
<td>02</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>00</td>
<td>08</td>
<td>02</td>
<td>00</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>02</td>
<td>00</td>
<td>80</td>
<td>00</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>83</td>
<td>00</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>DE</td>
<td>DE</td>
<td>DE</td>
<td>DE</td>
<td>X</td>
<td>!</td>
</tr>
</tbody>
</table>

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:

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

35
BLR (Branch if A < B)

Family Type: Register Comparison Branch

```
| 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 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 |
```

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:

<table>
<thead>
<tr>
<th>l R3 ! R6 ! Branch !</th>
<th>l 420D36 BLR R3,R6 TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>l 00 ! 00 ! X !</td>
<td>1 X 1</td>
</tr>
<tr>
<td>l 01 ! 10 ! X !</td>
<td>1 X 1</td>
</tr>
<tr>
<td>l 10 ! 05 ! X !</td>
<td>1 X 1</td>
</tr>
<tr>
<td>l 80 ! 8F ! X !</td>
<td>1 X 1</td>
</tr>
<tr>
<td>l 80 ! 02 ! X !</td>
<td>1 X 1</td>
</tr>
<tr>
<td>l 02 ! 80 ! X !</td>
<td>1 X 1</td>
</tr>
<tr>
<td>l FF ! 83 ! X !</td>
<td>1 X 1</td>
</tr>
<tr>
<td>l DE ! DE ! X !</td>
<td>1 X 1</td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
BLRX (Branch if A < B 16 bit test)

Family Type: Register Comparison Branch

<table>
<thead>
<tr>
<th>22221111111111000000000000</th>
<th>321098765432109876543210</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001xxxxxxxxAAXAXAXAXAXAXB</td>
<td>BBBBBBBBBBBBBBBBBBBBBBBBB</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>R3</th>
<th>R2</th>
<th>R6</th>
<th>R5</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>00</td>
<td>05</td>
<td>!</td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>05</td>
<td>00</td>
<td>!</td>
<td>X</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>05</td>
<td>00</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>80</td>
<td>8F</td>
<td>80</td>
<td>8F</td>
<td>!</td>
<td>X</td>
</tr>
<tr>
<td>80</td>
<td>00</td>
<td>8F</td>
<td>00</td>
<td>X</td>
<td>!</td>
</tr>
<tr>
<td>80</td>
<td>00</td>
<td>00</td>
<td>02</td>
<td>!</td>
<td>X</td>
</tr>
<tr>
<td>00</td>
<td>08</td>
<td>02</td>
<td>00</td>
<td>!</td>
<td>X</td>
</tr>
<tr>
<td>02</td>
<td>00</td>
<td>80</td>
<td>00</td>
<td>!</td>
<td>X</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>83</td>
<td>00</td>
<td>!</td>
<td>X</td>
</tr>
<tr>
<td>DE</td>
<td>DE</td>
<td>DE</td>
<td>DE</td>
<td>!</td>
<td>X</td>
</tr>
</tbody>
</table>

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
BNH (Branch if Not Equal, High nibble)

**Family Type:** Valued Branch

<table>
<thead>
<tr>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

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:**

```assembly
$ORG $6000

6000 FC03C2 BNH $OC,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 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

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

\[
\begin{array}{cccccccccccccccc}
2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

\[
\begin{array}{cccccccccccccccc}
3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4
\end{array}
\]

\[
\begin{array}{cccccccccccccccc}
3 & 2 & 1 & 0 & 1 & 0 & x & x & x & x & x & x & x & x & A & A & A & B & B & B
\end{array}
\]

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:

<table>
<thead>
<tr>
<th>R3</th>
<th>R6</th>
<th>Branch</th>
<th>5A0D36</th>
<th>BNR R3,R6 TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>!</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>!</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>05</td>
<td>!</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>02</td>
<td>!</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>82</td>
<td>!</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>83</td>
<td>!</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>DE</td>
<td>!</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
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 1 0 0 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 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:

```
<table>
<thead>
<tr>
<th>R3</th>
<th>Branch</th>
<th>5A0D3F</th>
<th>BNZ</th>
<th>R3</th>
<th>TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>! X !</td>
<td></td>
<td></td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>! X !</td>
<td>!</td>
<td></td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>! X !</td>
<td>!</td>
<td></td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>! X !</td>
<td>!</td>
<td></td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>! X !</td>
<td>!</td>
<td></td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>! X !</td>
<td>!</td>
<td></td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>! X !</td>
<td>!</td>
<td></td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>! X !</td>
<td>!</td>
<td></td>
<td>!</td>
<td></td>
</tr>
</tbody>
</table>
```

Copyright © 1982, 1983 by Computer Concepts Corporation  
Shawnee Mission, KS  
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
BTH (Branch if Bits True, High nibble)

Family Type: Masked Branch

```
2 2 2 2 1 1 1 1 1 1 1 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
```

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 RO < $01 Not Taken
6002 DC0050 JMP $5000
6003 20000F SET RO < $00 Branch Taken
6004 DC0050 JMP $5000
```

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

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
BTL (Branch if Bits True, Low Nibble)

Family Type: Masked Branch

```
2 2 2 2 1 1 1 1 1 1 1 1 1 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
```

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 RO < $01 Not Taken
6002 DC0050 JMP $5000
6003 20000F SET RO = $00 Branch Taken
6004 DC0050 JMP $5000
```

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

---

Copyright © 1982, 1983 by Computer Concepts Corporation  
Shawnee Mission, KS  
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
CIO (Input Output Instruction)

Family: Peripheral Control

```
2 2 2 2 1 1 1 1 1 1 1 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
```

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

- \( xxx x 1 \):
  - Fire internal IBS one shot (SRS). Sets CPb
  - Basically used for Status Requests from MUXD

- \( xxx l x \):
  - Fire CBS Strobe

- \( x l x x x \):
  - Fire OBS Strobe

- \( l x x x x \):
  - 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.
Selecting devices:

A005F 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 < $BD,SH Mask out and clear CPb
B10D2D 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 BPR $01,SH FIRE See if expired
3001 87800F RTS Still on
3002 17800C FIRE CIO TIM Fire Oneshot
3003 87800F RTS Return to caller

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
DAC (Decimal Add with Carry)

Family Type: Register ALU

\[
2 \begin{array}{cccccc}
2 & 2 & 2 & 2 & 1 & 1 \\
3 & 2 & 1 & 0 & 9 & 8 \\
7 & 6 & 5 & 4 & 3 & 2 \\
1 & 0 & 9 & 8 & 7 & 6 \\
5 & 4 & 3 & 2 & 1 & 0 \\
\end{array}
\]

\[
0 \begin{array}{cccccc}
0 & 0 & 1 & 0 & 0 & 0 \\
1 & 6 & 3 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

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.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>Carry</th>
<th>CC</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>01</td>
<td>0</td>
<td>02</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
<td>0</td>
<td>02</td>
<td>0</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>10</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>AA</td>
<td>AA</td>
<td>BA</td>
<td>1</td>
<td>BB</td>
<td>1</td>
</tr>
<tr>
<td>80</td>
<td>01</td>
<td>81</td>
<td>0</td>
<td>82</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
<td>85</td>
<td>1</td>
<td>86</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>46</td>
<td>0</td>
<td>47</td>
<td>0</td>
</tr>
<tr>
<td>56</td>
<td>78</td>
<td>34</td>
<td>1</td>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>9A</td>
<td>BC</td>
<td>BC</td>
<td>1</td>
<td>BD</td>
<td>1</td>
</tr>
<tr>
<td>FO</td>
<td>DE</td>
<td>34</td>
<td>1</td>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>02</td>
<td>FF</td>
<td>67</td>
<td>1</td>
<td>68</td>
<td>1</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>54</td>
<td>1</td>
<td>55</td>
<td>1</td>
</tr>
</tbody>
</table>

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

46
DACX (Decimal Add with carry 16 bit)

Family Type: Register ALU

```
2 2 2 2 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
```

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 \( CS \)
- 1 1 Set Carry first \( CS \)

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

<table>
<thead>
<tr>
<th></th>
<th>Result</th>
<th>CC</th>
<th>Result</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
<td>R0</td>
<td>R3</td>
<td>R2</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
<td>0</td>
<td>0 0 0 2</td>
</tr>
<tr>
<td>5 5 5 5</td>
<td>5 5 5 5</td>
<td>6 6 6 6</td>
<td>1</td>
<td>6 6 6 6</td>
</tr>
<tr>
<td>1 1 2 2</td>
<td>3 3 4 4</td>
<td>4 4 6 6</td>
<td>0</td>
<td>4 4 6 7</td>
</tr>
<tr>
<td>1 1 2 2</td>
<td>3 3 4 4</td>
<td>4 4 6 6</td>
<td>1</td>
<td>4 4 6 7</td>
</tr>
<tr>
<td>8 0 0 0</td>
<td>8 0 0 1</td>
<td>8 0 0 1</td>
<td>0</td>
<td>8 0 0 2</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
<td>0</td>
<td>0 0 0 2</td>
</tr>
<tr>
<td>9 A B C</td>
<td>C D E F</td>
<td>C E 0 1</td>
<td>1</td>
<td>C E 0 2</td>
</tr>
<tr>
<td>A A A A</td>
<td>2 2 2 2</td>
<td>3 3 3 2</td>
<td>1</td>
<td>3 3 3 3</td>
</tr>
<tr>
<td>5 5 1 9</td>
<td>5 5 9 1</td>
<td>1 1 0 1</td>
<td>1</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

---

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

47
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 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
```

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

### 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:

<table>
<thead>
<tr>
<th>RO</th>
<th>R1</th>
<th>R2</th>
<th>Carry</th>
<th>R2</th>
<th>Carry</th>
<th>Result CC</th>
<th>Result CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0 0</td>
<td>99</td>
<td>1</td>
<td>948201</td>
<td>DSC R2 &lt; R0,R1 ,CC</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>99</td>
<td>1</td>
<td>98</td>
<td>1</td>
<td>94C201</td>
<td>DSC R2 &lt; R0,R1 ,CS</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>00</td>
<td>0 0</td>
<td>99</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>AA</td>
<td>00</td>
<td>0 0</td>
<td>99</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>01</td>
<td>79</td>
<td>0 0</td>
<td>78</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
<td>2B</td>
<td>1 1</td>
<td>2A</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>78</td>
<td>1 1</td>
<td>77</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>78</td>
<td>78</td>
<td>1 1</td>
<td>77</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9A</td>
<td>BC</td>
<td>78</td>
<td>1 1</td>
<td>77</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>FF</td>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0 0</td>
<td>99</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
DSCX (Decimal Subtract with carry 16 bit)

Family Type: Register ALU

\[
\begin{array}{ccccccccccccccccccc}
2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
\end{array}
\]

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

<table>
<thead>
<tr>
<th>15 14 Carry Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 Normal, Initial Carry State not affected</td>
</tr>
<tr>
<td>0 1 Normal, Initial Carry State not affected</td>
</tr>
<tr>
<td>1 0 Clear Carry First, CC</td>
</tr>
<tr>
<td>1 1 Set Carry first, CS</td>
</tr>
</tbody>
</table>

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>R0</td>
<td>R3</td>
<td>R2</td>
</tr>
<tr>
<td>R5</td>
<td>R4</td>
<td>Carry</td>
<td>R5</td>
</tr>
<tr>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
</tr>
<tr>
<td>00 00</td>
<td>00 00</td>
<td>99 99</td>
<td>1</td>
</tr>
<tr>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
</tr>
<tr>
<td>55 AA</td>
<td>AA 55</td>
<td>45 55</td>
<td>1</td>
</tr>
<tr>
<td>11 22</td>
<td>33 44</td>
<td>1</td>
<td>77 77</td>
</tr>
<tr>
<td>44 55</td>
<td>66 77</td>
<td>77 77</td>
<td>1</td>
</tr>
<tr>
<td>80 00</td>
<td>00 00</td>
<td>79 99</td>
<td>1</td>
</tr>
<tr>
<td>00 01</td>
<td>80 00</td>
<td>20 01</td>
<td>1</td>
</tr>
<tr>
<td>9A BC</td>
<td>CD EF</td>
<td>66 67</td>
<td>1</td>
</tr>
<tr>
<td>AA AA</td>
<td>22 22</td>
<td>88 88</td>
<td>0</td>
</tr>
<tr>
<td>55 19</td>
<td>55 91</td>
<td>99 28</td>
<td>1</td>
</tr>
</tbody>
</table>

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
IADC (Immediate Binary Add with Carry registers)

Family Type: Immediate ALU

```
<table>
<thead>
<tr>
<th>2 2 2 2</th>
<th>1 1 1 1</th>
<th>1 1 1 1</th>
<th>1 1 0 0</th>
<th>0 0 0 0</th>
<th>0 0 0 0</th>
<th>0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 2 1 0</td>
<td>9 8 7 6</td>
<td>5 4 3 2</td>
<td>1 0 9 8</td>
<td>7 6 5 4</td>
<td>3 2 1 0</td>
<td></td>
</tr>
</tbody>
</table>
```

```
p o l l i o 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)
- iii = High order Nibble of Immediate Data Constant
- III = 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:

<table>
<thead>
<tr>
<th>! Imm!</th>
<th>! Result CC</th>
<th>! Result CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>! Data!</td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 01 ! 01 ! 0</td>
<td>! 02 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 01 ! 00 ! 01 ! 0</td>
<td>! 02 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 01 ! 00 ! 01 ! 0</td>
<td>! 02 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 00 ! 00 ! 00 ! 0</td>
<td>! 01 ! 0 !</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
**IADD (Immediate ADD registers)**

*Family Type: Immediate ALU*

```
  2 2 2 2 1 1 1 1 1 1 1 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
```

Where  
- \(P\) = Parity Bit (Odd Parity)
- \(m\) = Data Memory Control (See table 4)
- \(I\) = High order Nibble of Immediate Data Constant
- \(I I I I\) = 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:

<table>
<thead>
<tr>
<th>Imm!</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>! 00</td>
<td>00</td>
</tr>
<tr>
<td>! 00</td>
<td>01</td>
</tr>
<tr>
<td>! 01</td>
<td>00</td>
</tr>
<tr>
<td>! 01</td>
<td>01</td>
</tr>
<tr>
<td>! 55</td>
<td>55</td>
</tr>
<tr>
<td>! AA</td>
<td>AA</td>
</tr>
<tr>
<td>! 80</td>
<td>01</td>
</tr>
<tr>
<td>! 80</td>
<td>FF</td>
</tr>
<tr>
<td>! 12</td>
<td>14</td>
</tr>
<tr>
<td>! 56</td>
<td>78</td>
</tr>
<tr>
<td>! 9A</td>
<td>BC</td>
</tr>
<tr>
<td>! F0</td>
<td>DE</td>
</tr>
<tr>
<td>! 02</td>
<td>FF</td>
</tr>
<tr>
<td>! FF</td>
<td>FF</td>
</tr>
</tbody>
</table>

---

Copyright © 1982, 1983 by Computer Concepts Corporation  
Shawnee Mission, KS  
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation  
51
IAND (Immediate Logical AND registers)

Family Type: Immediate ALU

```
2 2 2 2 1 1 1 1 1 1 1 1 1 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 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)  
niiii = High order Nibble of Immediate Data Constant  
III = 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

<table>
<thead>
<tr>
<th>Imm</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>R1</td>
</tr>
<tr>
<td>!00</td>
<td>!00</td>
</tr>
<tr>
<td>!00</td>
<td>!01</td>
</tr>
<tr>
<td>!01</td>
<td>!00</td>
</tr>
<tr>
<td>!55</td>
<td>!55</td>
</tr>
<tr>
<td>!AA</td>
<td>!AA</td>
</tr>
<tr>
<td>!80</td>
<td>!01</td>
</tr>
<tr>
<td>!80</td>
<td>!FF</td>
</tr>
<tr>
<td>!12</td>
<td>!34</td>
</tr>
<tr>
<td>!56</td>
<td>!78</td>
</tr>
<tr>
<td>!9A</td>
<td>!BC</td>
</tr>
<tr>
<td>!F0</td>
<td>!DE</td>
</tr>
<tr>
<td>!02</td>
<td>!FF</td>
</tr>
<tr>
<td>!FF</td>
<td>!FF</td>
</tr>
</tbody>
</table>

IAND R2 < (Immediate Data),R1
IDAC (Immediate Decimal Add with Carry registers)

Family Type: Immediate ALU

\[
\begin{array}{ccccccccc}
2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 \\
3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0
\end{array}
\]

\[
p_{0000}110000_{d d d d I I I I B B B B}
\]

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**

<table>
<thead>
<tr>
<th>Imm</th>
<th>Result CC</th>
<th>Result CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>10</td>
</tr>
<tr>
<td>AA</td>
<td>AA</td>
<td>BA</td>
</tr>
<tr>
<td>80</td>
<td>01</td>
<td>81</td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
<td>E5</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>46</td>
</tr>
<tr>
<td>56</td>
<td>78</td>
<td>34</td>
</tr>
<tr>
<td>9A</td>
<td>BC</td>
<td>BC</td>
</tr>
<tr>
<td>FD</td>
<td>DE</td>
<td>34</td>
</tr>
<tr>
<td>02</td>
<td>FF</td>
<td>67</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>54</td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
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 1 0 0 0 0 0 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
```

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

<table>
<thead>
<tr>
<th>( \text{Imm} )</th>
<th>( \text{Result CC} )</th>
<th>( \text{Result CS} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Data: R1} )</td>
<td>( \text{R2} )</td>
<td>( \text{Carry} )</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>99</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>00</td>
</tr>
<tr>
<td>AA</td>
<td>AA</td>
<td>00</td>
</tr>
<tr>
<td>80</td>
<td>01</td>
<td>79</td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
<td>2B</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>78</td>
</tr>
<tr>
<td>56</td>
<td>78</td>
<td>1</td>
</tr>
<tr>
<td>9A</td>
<td>BC</td>
<td>78</td>
</tr>
<tr>
<td>F0</td>
<td>DE</td>
<td>1C</td>
</tr>
<tr>
<td>02</td>
<td>FF</td>
<td>AD</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>00</td>
</tr>
</tbody>
</table>

---

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

54
IMUL (Immediate Multiply Register with Constant)

Family Type: Immediate ALU

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 222 | 222 | 111 | 111 | 111 | 110 | 000 | 000 | 000 | 000 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 3210| 9876| 5432| 1098| 7654| 3210|
+-----------------------------------------------+
p0 l1 l1 100 c0 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
III = 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.

<table>
<thead>
<tr>
<th>Imm</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>!Data! R1</td>
<td>R2</td>
</tr>
<tr>
<td>! O ! 00 ! 00 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! B ! 01 ! 0B ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 1 ! 00 ! 00 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 5 ! 55 ! 19 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! A ! AA ! 64 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 0 ! 01 ! 00 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 0 ! FF ! 00 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 2 ! 34 ! 08 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 6 ! 78 ! 30 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 9 ! BC ! 6C ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! F ! ED ! C3 ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! 2 ! FF ! 1E ! 0 !</td>
<td></td>
</tr>
<tr>
<td>! F ! FF ! R1 ! 0 ! (Largest possible answer)</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
IOR (Immediate OR registers)

Family Type: Immediate ALU

```
2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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
```

Where

- **P** = Parity Bit (Odd Parity)
- **m** = Data Memory Control (See table 4)
- **iii** = High order Nibble of Immediate Data Constant
- **III** = 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:

<table>
<thead>
<tr>
<th>Imm</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>R1</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>AA</td>
<td>AA</td>
</tr>
<tr>
<td>80</td>
<td>01</td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
</tr>
<tr>
<td>56</td>
<td>78</td>
</tr>
<tr>
<td>9A</td>
<td>BC</td>
</tr>
<tr>
<td>FO</td>
<td>DE</td>
</tr>
<tr>
<td>02</td>
<td>FF</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
IXOR (Immediate Exclusive OR registers)

Family Type: Immediate ALU

```
2 2 2 2 1 1 1 1 1 1 1 1 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
```

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.

<table>
<thead>
<tr>
<th>Imm!</th>
<th>Result</th>
<th>IXOR R2 &lt; (Immediate Data), R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>! 00</td>
<td>00</td>
<td>! 00 0! 00 0! 0! 0 !</td>
</tr>
<tr>
<td>! 00</td>
<td>01</td>
<td>! 00 0! 01 0! 0! 0 !</td>
</tr>
<tr>
<td>! 01</td>
<td>00</td>
<td>! 01 0! 00 0! 0! 0 !</td>
</tr>
<tr>
<td>! 55</td>
<td>55</td>
<td>! 55 5! 55 5! 5! 0 !</td>
</tr>
<tr>
<td>! AA</td>
<td>AA</td>
<td>! AA A! AA A! AA 0 !</td>
</tr>
<tr>
<td>! 80</td>
<td>01</td>
<td>! 80 1! 81 1! 81 0 !</td>
</tr>
<tr>
<td>! 80</td>
<td>FF</td>
<td>! 80 F! 7F F! 7F 0 !</td>
</tr>
<tr>
<td>! 12</td>
<td>34</td>
<td>! 12 ! 34 ! 26 ! 0 !</td>
</tr>
<tr>
<td>! 56</td>
<td>78</td>
<td>! 56 7! 78 7! 78 0 !</td>
</tr>
<tr>
<td>! 9A</td>
<td>BC</td>
<td>! 9A B! 9A B! 26 ! 0 !</td>
</tr>
<tr>
<td>! 00</td>
<td>DE</td>
<td>! 00 D! 00 D! 2E ! 0 !</td>
</tr>
<tr>
<td>! 02</td>
<td>FF</td>
<td>! 02 F! 02 F! 02 0 !</td>
</tr>
<tr>
<td>! FF</td>
<td>FF</td>
<td>! FF F! FF F! FF 0 !</td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation.
Family Type: Branch

JMP (Jump)

<table>
<thead>
<tr>
<th>2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

-----------------------------
p 1 0 1 1 1 x x x x x x x x 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. DB6A08 = JMP $0A6A

| p | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

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.
Wang 2200 Machine Instruction Set

JSR (Jump to SubRoutine)

Family Type: Branch

```
2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
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 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 ....
0900 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 9 8 7 6 5 4 3 2 1 0
```

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.
LPI (Load PH-PL Pair directly) - Cont'd

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
```
Wang 2200 Machine Instruction Set

### 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 |
+------------------------+------------------------+------------------------+------------------------+
```

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:

<table>
<thead>
<tr>
<th>( R0 )</th>
<th>( R1 )</th>
<th>( R2 )</th>
<th>( Carry )</th>
<th>( \text{Result} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>00</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>19</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AA</td>
<td>AA</td>
<td>64</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>01</td>
<td>00</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
<td>00</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>08</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>56</td>
<td>78</td>
<td>30</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9A</td>
<td>BC</td>
<td>78</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F0</td>
<td>DE</td>
<td>00</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>02</td>
<td>FF</td>
<td>1E</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>E1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

62
Wang 2200 Machine Instruction Set

MULX (Multiply Two four bit nibbles)

Family Type: Register ALU

```
2 2 2 2 1 1 1 1 1 1 1 1 1 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
```

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

<p>|</p>
<table>
<thead>
<tr>
<th>15 14</th>
<th>Multiply Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Multiply Lower B by Lower A (ALBL)</td>
</tr>
<tr>
<td>0 1</td>
<td>Multiply Lower B by Upper A (ABBL)</td>
</tr>
<tr>
<td>1 0</td>
<td>Multiply Upper B by Lower A (ALBH)</td>
</tr>
<tr>
<td>1 1</td>
<td>Multiply Upper B by Upper A (ABBR)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>R1</th>
<th>R0</th>
<th>R3</th>
<th>R2</th>
<th>R5</th>
<th>R4</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 00</td>
<td>00 01</td>
<td>00 00</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 01</td>
<td>00 00</td>
<td>00 00</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 01</td>
<td>00 00</td>
<td>00 00</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55 AA</td>
<td>AA 55</td>
<td>32 32</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 22</td>
<td>33 44</td>
<td>03 08</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44 55</td>
<td>66 77</td>
<td>18 23</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 00</td>
<td>00 01</td>
<td>00 00</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 01</td>
<td>00 00</td>
<td>00 00</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9A BC</td>
<td>CD EF</td>
<td>82 B4</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA AA</td>
<td>22 22</td>
<td>14 14</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55 19</td>
<td>55 91</td>
<td>19 09</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

63
Wang 2200 Machine Instruction Set

OR (Inclusive OR registers)

Family Type: Register ALU

```
2 2 2 2 1 1 1 1 1 1 1 1 1 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
```

\[ \begin{array}{cccccccccccc}
\& 0 & 0 & 0 & 0 & 0 & 0 & x & x & m & d & d & d & A & A & A & A & B & B & B & B
\end{array} \]

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:

<table>
<thead>
<tr>
<th>( R0 )</th>
<th>( R1 )</th>
<th>( R2 )</th>
<th>Carry</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>01</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>55</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>AA</td>
<td>AA</td>
<td>AA</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>80</td>
<td>01</td>
<td>81</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
<td>FF</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>36</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>56</td>
<td>78</td>
<td>78</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9A</td>
<td>BC</td>
<td>BE</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F0</td>
<td>DE</td>
<td>FE</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>02</td>
<td>FF</td>
<td>FF</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>FF</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
ORX (Inclusive OR registers 16 bit)

Family Type: Register ALU

\[
\begin{array}{cccccccccccc}
2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0
\end{array}
\]

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

<table>
<thead>
<tr>
<th>( R_1 )</th>
<th>( R_0 )</th>
<th>( R_3 )</th>
<th>( R_2 )</th>
<th>( R_5 )</th>
<th>( R_4 )</th>
<th>( Carry )</th>
<th>( )</th>
<th>( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>0</td>
<td></td>
<td></td>
<td>828402 ORX ( R_4 ) ( &lt; R_0,R_2 ) ( ,CC )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 00</td>
<td>00 01</td>
<td>00 01</td>
<td>0</td>
<td></td>
<td></td>
<td>|</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 01</td>
<td>00 00</td>
<td>00 01</td>
<td>0</td>
<td></td>
<td></td>
<td>|</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55 AA</td>
<td>AA 55</td>
<td>FF FF</td>
<td>0</td>
<td></td>
<td></td>
<td>|</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 22</td>
<td>33 44</td>
<td>33 66</td>
<td>0</td>
<td></td>
<td></td>
<td>|</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44 55</td>
<td>66 77</td>
<td>66 77</td>
<td>0</td>
<td></td>
<td></td>
<td>|</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80 00</td>
<td>00 01</td>
<td>80 01</td>
<td>0</td>
<td></td>
<td></td>
<td>|</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 01</td>
<td>80 00</td>
<td>80 01</td>
<td>0</td>
<td></td>
<td></td>
<td>|</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9A BC</td>
<td>CD EF</td>
<td>DF FF</td>
<td>0</td>
<td></td>
<td></td>
<td>|</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA AA</td>
<td>22 22</td>
<td>AA AA</td>
<td>0</td>
<td></td>
<td></td>
<td>|</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55 19</td>
<td>55 91</td>
<td>55 99</td>
<td>0</td>
<td></td>
<td></td>
<td>|</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
Wang 2200 Machine Instruction Set

RTS (Return from Sub-Routine)

Family: Stack Manipulation

2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 3 2 1 0 9 8 7 6 5 4 3 2 1 0

Where

\[ P = \text{Parity Bit (Odd Parity)} \]
\[ m = \text{Data Memory Control (See Table 4)} \]
\[ cc = \text{Control Memory Control (See table 5)} \]
\[ B = \text{B Register Selection (Used only during DM writes)} \]
\[ X = \text{Don't Care} \]

The RTS instruction is one of the few Wang instructions that are not as straightforward 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 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.

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
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
07B400 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

```
090FFE LPI $0FFE Load PHPL with address
D7110C JSR READCM Goto Subroutine
..... ..... Data Now present in K PH PL
```

```
READCM 05800F TPS Transfer Address to stack
87B600 RTS ,RC Return and read data
```

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
SBC (Subtract Binary with Carry)

Family Type: Register ALU

\[
\begin{align*}
2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1
\end{align*}
\]

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:

<table>
<thead>
<tr>
<th>R2 Carry</th>
<th>Result CC</th>
<th>Result CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>OC8201</td>
<td>SBC R2 &lt; R0,R1, CC</td>
<td></td>
</tr>
<tr>
<td>00 0 0 FF 0</td>
<td>00 1 1</td>
<td>80C201 SBC R2 &lt; R0,R1, CS</td>
</tr>
<tr>
<td>00 1 0 FF 0</td>
<td>1 FF 0 1</td>
<td></td>
</tr>
<tr>
<td>01 0 0 00 1</td>
<td>01 1 1</td>
<td></td>
</tr>
<tr>
<td>1 1 5 5 1 FF 0</td>
<td>0 00 1 1</td>
<td></td>
</tr>
<tr>
<td>1 AA 1 AA FF 0</td>
<td>0 00 1 1</td>
<td></td>
</tr>
<tr>
<td>1 80 1 0 1 7E 1</td>
<td>1 7F 1 1</td>
<td></td>
</tr>
<tr>
<td>1 80 FF BB 0</td>
<td>1 81 0 1</td>
<td></td>
</tr>
<tr>
<td>1 1 2 3 4 DD 0</td>
<td>1 DE 0 1</td>
<td></td>
</tr>
<tr>
<td>1 56 78 DD 0</td>
<td>1 DE 0 1</td>
<td></td>
</tr>
<tr>
<td>1 9A 1 BC DD 0</td>
<td>1 DE 0 1</td>
<td></td>
</tr>
<tr>
<td>1 F0 DE 1 1</td>
<td>1 1 12 1 1</td>
<td></td>
</tr>
<tr>
<td>1 02 FF BB 0</td>
<td>0 03 0 0</td>
<td></td>
</tr>
<tr>
<td>1 FF FF FF 0</td>
<td>0 00 1 1</td>
<td></td>
</tr>
</tbody>
</table>

---

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
Wang 2200 Machine Instruction Set

**SBCX (Subtract Binary with carry 16 bit)**

**Family Type:** Register ALU

```
2 2 2 2 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```

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 \( \text{CC} \)
- 1 1 Set Carry first \( \text{CS} \)

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

<table>
<thead>
<tr>
<th></th>
<th>Result</th>
<th>CC</th>
<th>Result</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 R0</td>
<td>R3 R2</td>
<td>R5 R4</td>
<td>Carry</td>
<td>R5 R4</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>00 00</td>
<td>FF FF</td>
<td>0 0 0 0</td>
<td>1 1</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>00 01</td>
<td>FF FE</td>
<td>0 0 0 0</td>
<td>1 1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>00 00</td>
<td>00 00</td>
<td>1 0 0 1</td>
<td>1 1</td>
</tr>
<tr>
<td>55 AA</td>
<td>AA 55</td>
<td>AB 54</td>
<td>0 0 AB</td>
<td>55 0</td>
</tr>
<tr>
<td>11 22</td>
<td>33 44</td>
<td>DD DD</td>
<td>0 0 DD</td>
<td>DB 0</td>
</tr>
<tr>
<td>44 55</td>
<td>66 77</td>
<td>DD DD</td>
<td>0 0 DD</td>
<td>DB 0</td>
</tr>
<tr>
<td>80 00</td>
<td>00 01</td>
<td>7F FE</td>
<td>1 7F FF</td>
<td>1 1</td>
</tr>
<tr>
<td>00 01</td>
<td>80 00</td>
<td>80 00</td>
<td>0 80 00</td>
<td>1 1</td>
</tr>
<tr>
<td>9A BC</td>
<td>CD EF</td>
<td>CC CC</td>
<td>0 0 CC</td>
<td>CD 0</td>
</tr>
<tr>
<td>AA AA</td>
<td>22 22</td>
<td>88 87</td>
<td>1 88 88</td>
<td>1 1</td>
</tr>
<tr>
<td>55 19</td>
<td>55 91</td>
<td>FF 87</td>
<td>0 FF 88</td>
<td>0 0</td>
</tr>
</tbody>
</table>

---

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
## SDC (Shift Decimal Characters)

**Family Type:** Register ALU

```
2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```

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

<table>
<thead>
<tr>
<th>19 18</th>
<th>Shift Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Shift Lower B and Lower A (ALBL)</td>
</tr>
<tr>
<td>0 1</td>
<td>Shift Lower B and Upper A (AHBL)</td>
</tr>
<tr>
<td>1 0</td>
<td>Shift Upper B and Lower A (ALBH)</td>
</tr>
<tr>
<td>1 1</td>
<td>Shift Upper B and Upper A (ABBH)</td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>AA</td>
<td>AA</td>
<td>AA</td>
</tr>
<tr>
<td>80</td>
<td>01</td>
<td>18</td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
<td>F8</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>41</td>
</tr>
<tr>
<td>56</td>
<td>78</td>
<td>85</td>
</tr>
<tr>
<td>9A</td>
<td>BC</td>
<td>C9</td>
</tr>
<tr>
<td>F0</td>
<td>DB</td>
<td>EF</td>
</tr>
<tr>
<td>02</td>
<td>FF</td>
<td>F0</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>FF</td>
</tr>
</tbody>
</table>

---

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
SDCX (Shift Decimal Characters - Extended)

Family Type: Register ALU

```
2 2 2 2 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
```

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 (AMBL) |
| 1  | 0 | Shift Upper B and Lower A (ALBH) |
| 1  | 1 | Shift Upper B and Upper A (AMBH) |

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 ABBL R4 < R0,R2
```

<table>
<thead>
<tr>
<th>R1</th>
<th>R0</th>
<th>R3</th>
<th>R2</th>
<th>R5</th>
<th>R4</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>00</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>55</td>
<td>AA</td>
<td>AA</td>
<td>55</td>
<td>AA</td>
<td>5A</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>22</td>
<td>33</td>
<td>44</td>
<td>31</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>55</td>
<td>66</td>
<td>77</td>
<td>64</td>
<td>75</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>08</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>80</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>09</td>
<td>BC</td>
<td>CD</td>
<td>EF</td>
<td>D9</td>
<td>F8</td>
<td>0</td>
</tr>
<tr>
<td>1A</td>
<td>AA</td>
<td>22</td>
<td>22</td>
<td>2A</td>
<td>2A</td>
<td>0</td>
</tr>
<tr>
<td>55</td>
<td>19</td>
<td>55</td>
<td>91</td>
<td>55</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

71
Wang 2200 Machine Instruction Set

SET (Load Register with Constant)

Family Type: Immediate ALU

<table>
<thead>
<tr>
<th>Imm</th>
<th>Result</th>
<th>SET R2 &lt; (Immediate Data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>AA</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>9A</td>
<td>9A</td>
<td></td>
</tr>
<tr>
<td>F0</td>
<td>F0</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td></td>
</tr>
</tbody>
</table>

Where

P = Parity Bit (Odd Parity)
m = Data Memory Control (See table 4)
iii = High order Nibble of Immediate Data Constant
III = 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
TAP (Transfer Auxiliary Register to PH-PL)

Family: Stack Manipulation

```
2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
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 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 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
```

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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>14 10 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>No Effect</td>
</tr>
<tr>
<td>0 0 1</td>
<td>+1 to (PHPL) then store - PH-PL not affected</td>
</tr>
<tr>
<td>0 1 0</td>
<td>+2</td>
</tr>
<tr>
<td>0 1 1</td>
<td>+3</td>
</tr>
<tr>
<td>1 0 0</td>
<td>No Effect</td>
</tr>
<tr>
<td>1 0 1</td>
<td>-1 to (PHPL) then store - PH-PL not affected</td>
</tr>
<tr>
<td>1 1 0</td>
<td>-2</td>
</tr>
<tr>
<td>1 1 1</td>
<td>-3 to (PHPL) then store - PH-PL not affected</td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
TPS (Transfer PH-PL to System Stack)

Family: Stack Manipulation

\[
\begin{array}{cccccccccccccccc}
2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 9 & 8 \\
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 \\
6 & 5 & 4 & 3 & 2 & 1 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
\end{array}
\]

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 P, (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

<table>
<thead>
<tr>
<th>Bit</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>No Effect</td>
</tr>
<tr>
<td>0 0 1</td>
<td>+1 to (PHPL) then store - PH-PL not affected</td>
</tr>
<tr>
<td>0 1 0</td>
<td>+2</td>
</tr>
<tr>
<td>0 1 1</td>
<td>+3</td>
</tr>
<tr>
<td>1 0 0</td>
<td>No Effect</td>
</tr>
<tr>
<td>1 0 1</td>
<td>-1 to (PHPL) then store - PH-PL not affected</td>
</tr>
<tr>
<td>1 1 0</td>
<td>-2 to (PHPL) then store - PH-PL not affected</td>
</tr>
<tr>
<td>1 1 1</td>
<td>-3 to (PHPL) then store - PH-PL not affected</td>
</tr>
</tbody>
</table>

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.

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

75
TSP (Transfer System Stack to PH-PL)

Family: Stack Manipulation

2 2 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 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 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 P, (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

| P | m | A | B | d | x | x | m | d | d | d | d | A | A | A | A | B | B | B | B |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

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

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>Carry</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>01</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>00</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>AA</td>
<td>AA</td>
<td>00</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>80</td>
<td>01</td>
<td>81</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
<td>7F</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>26</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>56</td>
<td>78</td>
<td>2E</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>9A</td>
<td>BC</td>
<td>26</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>FO</td>
<td>DE</td>
<td>2E</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>02</td>
<td>FF</td>
<td>FD</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>FF</td>
<td>FF</td>
<td>00</td>
<td>0</td>
<td>00</td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
XORX (Exclusive OR registers 16 bit)

Family Type: Register ALU

```
2 2 2 2 1 1 1 1 1 1 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 p 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

### Carry Control

| 0 0 | Normal, Initial Carry State not affected |
| 0 1 | Not Permitted (See SDC, SDCX makeup) |
| 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:

<table>
<thead>
<tr>
<th>( R1 )</th>
<th>( R0 )</th>
<th>( R3 )</th>
<th>( R2 )</th>
<th>( R5 )</th>
<th>( R4 )</th>
<th>( R6 )</th>
<th>( R7 )</th>
<th>( R8 )</th>
<th>( R9 )</th>
<th>( R10 )</th>
<th>( R11 )</th>
<th>( R12 )</th>
<th>( R13 )</th>
<th>( R14 )</th>
<th>( R15 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td></td>
</tr>
<tr>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td></td>
</tr>
<tr>
<td>11 22</td>
<td>33 44</td>
<td>55 66</td>
<td>77 88</td>
<td>99 AA</td>
<td>A0 BB</td>
<td>C0 D1</td>
<td>E0 F1</td>
<td>G0 H1</td>
<td>I0 J1</td>
<td>K0 L1</td>
<td>M0 N1</td>
<td>O0 P1</td>
<td>Q0 R1</td>
<td>S0 T1</td>
<td></td>
</tr>
<tr>
<td>44 55</td>
<td>66 77</td>
<td>88 99</td>
<td>00 AA</td>
<td>11 AB</td>
<td>22 AC</td>
<td>33 AD</td>
<td>44 AE</td>
<td>55 AF</td>
<td>66 AG</td>
<td>77 AH</td>
<td>88 AI</td>
<td>99 AJ</td>
<td>AA AK</td>
<td>BB AL</td>
<td></td>
</tr>
<tr>
<td>11 22</td>
<td>33 44</td>
<td>55 66</td>
<td>77 88</td>
<td>99 AA</td>
<td>A0 BB</td>
<td>C0 D1</td>
<td>E0 F1</td>
<td>G0 H1</td>
<td>I0 J1</td>
<td>K0 L1</td>
<td>M0 N1</td>
<td>O0 P1</td>
<td>Q0 R1</td>
<td>S0 T1</td>
<td></td>
</tr>
<tr>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td>80 00</td>
<td></td>
</tr>
<tr>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td>00 01</td>
<td></td>
</tr>
<tr>
<td>9A BC</td>
<td>CE DF</td>
<td>57 53</td>
<td>00 88</td>
<td>11 22</td>
<td>33 44</td>
<td>55 66</td>
<td>77 88</td>
<td>99 AA</td>
<td>A0 BB</td>
<td>C0 D1</td>
<td>E0 F1</td>
<td>G0 H1</td>
<td>I0 J1</td>
<td>K0 L1</td>
<td></td>
</tr>
<tr>
<td>AA AA</td>
<td>22 22</td>
<td>88 88</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td>00 00</td>
<td></td>
</tr>
</tbody>
</table>

---

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

78
XPA (Exchange PH-PL with Auxiliary Register)

Family: Stack Manipulation

2 2 2 2 1 1 1 1 1 0 0 0 0 0 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 0 No Effect
0 0 1 0 +1 to (PHPL) then store - PH-PL not affected
0 1 0 0 +2
0 1 1 0 +3
1 0 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  
<table>
<thead>
<tr>
<th>7 6 5 4</th>
<th>3 2 1 0</th>
<th>11 10 09 08</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>R0</td>
<td>0 0 0 0 R0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>R1</td>
<td>0 0 0 1 R1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>R2</td>
<td>0 0 1 0 R2</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>R3</td>
<td>0 0 1 1 R3</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>R4</td>
<td>0 1 0 0 R4</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>R5</td>
<td>0 1 0 1 R5</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>R6</td>
<td>0 1 1 0 R6</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>R7</td>
<td>0 1 1 1 R7</td>
</tr>
</tbody>
</table>

### Table 2 B Bus

<table>
<thead>
<tr>
<th>7 6 5 4</th>
<th>3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>R0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>R1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>R2</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>R3</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>R4</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>R5</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>R6</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>R7</td>
</tr>
</tbody>
</table>

### Table 3 C Bus (Destination)

<table>
<thead>
<tr>
<th>7 6 5 4</th>
<th>3 2 1 0</th>
<th>11 10 09 08</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>CL-</td>
<td>1 0 0 0 PL</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>CH-</td>
<td>1 0 0 1 PH</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>CL</td>
<td>1 0 1 0 CL</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>CH</td>
<td>1 0 1 1 CH</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>CL+</td>
<td>1 1 0 0 SL</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>CH+</td>
<td>1 1 0 1 SH</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>00+</td>
<td>1 1 1 0 K</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>00-</td>
<td>1 1 1 1 0</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>13 12</th>
<th>10 09</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 No Op</td>
<td>0 0 No Op</td>
</tr>
<tr>
<td>0 1 ,RD Read</td>
<td>0 1 No Op</td>
</tr>
<tr>
<td>1 0 ,W1 Write Byte at current PHPL</td>
<td>1 0 ,WC Write CM</td>
</tr>
<tr>
<td>1 1 ,W2 Write Byte at opposite PHPL pointed to pair</td>
<td>1 1 ,RC Read CM</td>
</tr>
</tbody>
</table>

---

Copyright © 1982,1983 by Computer Concepts Corporation  
Shawnee Mission, KS  
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
Appendix ii - Conditional Branch Examples

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

\[
\text{Bxxx } R3, R5
\]

Whereas the 16 bit comparisons are:

\[
\text{Bxxx } R2, R4
\]

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 8 bit comparisons, and 0 to 65535 for 16 bit comparisons.
Appendix iii Alphabetical Listing of Mnemonics

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>p 0 0 1 1 0 0 0 x x x m m d d d d A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADCX</td>
<td>p 0 0 1 1 0 1 0 x x x m m d d d d A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>p 0 0 0 1 0 0 0 x x x m m d d d d A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANDX</td>
<td>p 0 0 0 1 0 1 0 x x x m m d d d d A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEH</td>
<td>p 1 1 1 0 1 x x x x x x x x x x x x x x c c c c B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEL</td>
<td>p 1 1 1 0 0 x x x x x x x x x x x x x x c c c c B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BER</td>
<td>p 1 0 1 0 0 x x x x x x x x x x x x x x A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEST</td>
<td>p 1 1 1 0 1 x x x x x x x x x x x x x x c c c c B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFL</td>
<td>p 1 1 0 1 0 x x x x x x x x x x x x x x c c c c B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLER</td>
<td>p 1 0 0 1 0 x x x x x x x x x x x x x x A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLRX</td>
<td>p 1 0 0 0 1 x x x x x x x x x x x x x x A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNL</td>
<td>p 1 1 1 1 0 x x x x x x x x x x x x x x c c c c B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNR</td>
<td>p 1 0 1 1 0 x x x x x x x x x x x x x x A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNZ</td>
<td>p 1 0 1 1 0 x x x x x x x x x x x x x x A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTH</td>
<td>p 1 1 0 0 1 x x x x x x x x x x x x x x c c c c B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTL</td>
<td>p 1 1 0 0 0 x x x x x x x x x x x x x x c c c c B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIO</td>
<td>p 0 0 1 0 1 1 1 1 0 m m y y y y d d d d z z d d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAC</td>
<td>p 0 0 1 0 0 0 0 x x x m m d d d d A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DACX</td>
<td>p 0 0 1 0 0 1 0 x x x m m d d d d A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSC</td>
<td>p 0 0 1 0 1 0 0 x x x m m d d d d A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSCX</td>
<td>p 0 0 1 0 1 1 0 x x x m m d d d d A A A A B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDAC</td>
<td>p 0 1 1 1 0 i i i i i m m d d d d I I I I B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IADD</td>
<td>p 0 1 0 1 i i i i i m m d d d d I I I I B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IAND</td>
<td>p 0 1 0 1 i i i i i m m d d d d I I I I B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDAC</td>
<td>p 0 1 1 0 0 i i i i i m m d d d d I I I I B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDSC</td>
<td>p 0 1 1 0 1 i i i i i m m d d d d I I I I B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMUL</td>
<td>p 0 1 1 1 0 0 c 0 m m d d d d I I I I B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOR</td>
<td>p 0 1 0 0 i i i i i m m d d d d I I I I B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IXOR</td>
<td>p 0 1 0 0 i i i i i m m d d d d I I I I B B B B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 1982, 1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

82
### Appendix iii Alphabetical Listing of Mnemonics - Cont'd

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>p1o1l11x x x x x x x x x y y y y y y 0 0</td>
<td>Jump to Label P</td>
</tr>
<tr>
<td>JSR</td>
<td>p1o1l110 x x x x x x x x y y y y y y 0 0</td>
<td>Jump and Return to Label P</td>
</tr>
<tr>
<td>LPI</td>
<td>p001l1ffl ff m m x x x x x x x x x x</td>
<td>Link Program Instruction</td>
</tr>
<tr>
<td>MUL</td>
<td>p001l1100 x x m m d d d d A A A A B B B B</td>
<td>Multiply</td>
</tr>
<tr>
<td>MULX</td>
<td>p001l1110 x x m m d d d d A A A A B B B B</td>
<td>Multiply and Link</td>
</tr>
<tr>
<td>OR</td>
<td>p00000000 x x m m d d d d A A A A B B B B</td>
<td>OR (Logical OR)</td>
</tr>
<tr>
<td>ORX</td>
<td>p00000010 x x m m d d d d A A A A B B B B</td>
<td>OR with X</td>
</tr>
<tr>
<td>RTS</td>
<td>p0000011110 m m x c c 0 0 0 0 B B B B</td>
<td>Return from Subroutine</td>
</tr>
<tr>
<td>SBC</td>
<td>p00011100 x x m m d d d d A A A A B B B B</td>
<td>Subtract</td>
</tr>
<tr>
<td>SBCX</td>
<td>p00011110 x x m m d d d d A A A A B B B B</td>
<td>Subtract with Carry</td>
</tr>
<tr>
<td>SDC</td>
<td>p0000x x 0 0 0 1 m m d d d d A A A A B B B B</td>
<td>Subtract with Borrow</td>
</tr>
<tr>
<td>SD CX</td>
<td>p0000x x 1 0 0 1 m m d d d d A A A A B B B B</td>
<td>Subtract with Borrow and Link</td>
</tr>
<tr>
<td>SET</td>
<td>p0100001iiii m m d d d d IIII 1 1 11</td>
<td>Set Condition Code</td>
</tr>
<tr>
<td>TAP</td>
<td>p0001011110 m m x 0 0 r r r r r r B B B B</td>
<td>Test and Propagate</td>
</tr>
<tr>
<td>TPA</td>
<td>p00000011 m m x c c r r r r B B B B</td>
<td>Test and Propagate with Carry</td>
</tr>
<tr>
<td>TPS</td>
<td>p000010111 m m x c c 0 0 0 0 B B B B</td>
<td>Test and Propagate with Borrow</td>
</tr>
<tr>
<td>TSP</td>
<td>p0001110110 m m x 0 0 0 0 0 0 B B B B</td>
<td>Test and Propagate with Borrow and Link</td>
</tr>
<tr>
<td>XOR</td>
<td>p00000100 x x m m d d d d A A A A B B B B</td>
<td>Exclusive OR</td>
</tr>
<tr>
<td>XORX</td>
<td>p00000110 x x m m d d d d A A A A B B B B</td>
<td>Exclusive OR with X</td>
</tr>
<tr>
<td>XPA</td>
<td>p000000111 m m x c c r r r r B B B B</td>
<td>Exclusive OR and Propagate</td>
</tr>
</tbody>
</table>

---

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS

No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation
### Appendix iv - Numerical Listing of Mnemonics

<table>
<thead>
<tr>
<th>OR</th>
<th>TPA</th>
<th>CRX</th>
<th>SDCX</th>
<th>XPA</th>
<th>XOR</th>
<th>TPS</th>
<th>AND</th>
<th>ANDX</th>
<th>TAP</th>
<th>SBC</th>
<th>TSP</th>
<th>SBCX</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SACX</th>
<th>DAC</th>
<th>DSC</th>
<th>DSCX</th>
<th>CIO</th>
<th>ADC</th>
<th>AD CX</th>
<th>MUL</th>
<th>MULX</th>
<th>LPI</th>
<th>IOR</th>
<th>SET</th>
<th>IT XOR</th>
<th>IAND</th>
<th>IADD</th>
<th>IDAC</th>
<th>IDEC</th>
<th>IAD C</th>
<th>I MUL</th>
</tr>
</thead>
</table>

---

Copyright @ 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

84
### Appendix iv - Numerical Listing of Mnemonics - Cont'd

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Machine Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLR</td>
<td>100 0 0 x x x x x x x x x A A A A B B B B</td>
<td>Branch Link Register</td>
</tr>
<tr>
<td>BLRX</td>
<td>100 0 1 x x x x x x x x x A A A A B B B B</td>
<td>Branch Link Register X</td>
</tr>
<tr>
<td>BLER</td>
<td>100 1 0 x x x x x x x x x A A A A B B B B</td>
<td>Branch Link Extended Register</td>
</tr>
<tr>
<td>BLEX</td>
<td>100 1 1 x x x x x x x x x A A A A B B B B</td>
<td>Branch Link Extended Register X</td>
</tr>
<tr>
<td>BBR</td>
<td>101 0 0 x x x x x x x x x A A A A B B B B</td>
<td>Branch Branch Register</td>
</tr>
<tr>
<td>BRZ</td>
<td>101 0 1 0 x x x x x x x x y y y y y y 0 0</td>
<td>Branch Branch Register Zero</td>
</tr>
<tr>
<td>JSR</td>
<td>101 1 0 x x x x x x x x x A A A A B B B B</td>
<td>Jump to Subroutine</td>
</tr>
<tr>
<td>BNZ</td>
<td>101 1 1 x x x x x x x x x A A A A 1 1 1 1</td>
<td>Branch Not Zero</td>
</tr>
<tr>
<td>JMP</td>
<td>101 1 1 x x x x x x x x x y y y y y y 0 0</td>
<td>Jump to Memory</td>
</tr>
<tr>
<td>BTL</td>
<td>110 0 0 x x x x x x x x x c c c c B B B B</td>
<td>Branch To Label</td>
</tr>
<tr>
<td>BTH</td>
<td>110 1 0 x x x x x x x x x c c c c B B B B</td>
<td>Branch To High</td>
</tr>
<tr>
<td>BFL</td>
<td>110 1 1 x x x x x x x x x c c c c B B B B</td>
<td>Branch To Low</td>
</tr>
<tr>
<td>BFH</td>
<td>111 0 0 x x x x x x x x x c c c c B B B B</td>
<td>Branch To Far High</td>
</tr>
<tr>
<td>BNL</td>
<td>111 0 1 x x x x x x x x x c c c c B B B B</td>
<td>Branch Near Low</td>
</tr>
<tr>
<td>BNH</td>
<td>111 1 0 x x x x x x x x x c c c c B B B B</td>
<td>Branch Near High</td>
</tr>
<tr>
<td>BNZ</td>
<td>111 1 1 x x x x x x x x x c c c c 1 1 1 1</td>
<td>Branch Not Zero</td>
</tr>
</tbody>
</table>

\[ | = \]
\[ | <> | \]
\[ | <= | \]

\[ | = | \]
\[ | <> | \]
\[ | <= | \]

\[ \text{ABS}(-F) = \text{ABSOLUTE} \]
\[ \text{ABS}(-F) = \text{ABSOLUTE} \]

\[ | = | \]
\[ | <> | \]
\[ | <= | \]

\[ | = | \]
\[ | <> | \]
\[ | <= | \]

---

Copyright © 1982,1983 by Computer Concepts Corporation
Shawnee Mission, KS
No part of this document may be reproduced without the expressed written permission of Computer Concepts Corporation

85