# 8XC196KC/KD Instruction Set Reference

# A

# APPENDIX A
# 8XC196KC/KD INSTRUCTION SET REFERENCE

This appendix provides reference information for the 8XC196KC/KD instruction set. It describes each instruction, shows the relationships between instructions and PSW flags, and shows hexadecimal opcodes, instruction lengths, and execution times.

Table A-1 defines the variables used in Table A-2 to represent instruction operands. Table A-2 lists the instructions alphabetically and describes each of them.

Tables A-3 and A-4 define the abbreviations and symbols used in Tables A-5 and A-6. Table A-5 shows the effect of each instruction on the Program Status Word flags, and Table A-6 shows the effect of the PSW flags or a specified register bit on conditional jump instructions.

Table A-7 lists the instruction opcodes, in hexadecimal order, along with the corresponding instruction mnemonics. Table A-8 is a map of the 8XC196KC/KD opcodes.

Table A-9 lists instruction lengths and opcodes for each applicable addressing mode. Table A-10 lists instruction execution times, expressed in state times. Table A-11 lists execution times, expressed in state times, for PTS cycles.

## Table A-1. Operand Variables

| Variable | Description |
|---|---|
| aa | A two-bit field within an opcode that selects the basic addressing mode used. This field is present only in those opcodes that allow address mode options. The field is encoded as follows:<br><br>0 0 Register direct<br>0 1 Immediate<br>1 0 Indirect<br>1 1 Indexed |
| baop | A byte operand that is addressed by any address mode. |
| bbb | A three-bit field within an opcode that selects a specific bit within a register. |
| bitno | A three-bit field within an opcode that selects one of the eight bits in a byte. |
| breg | A byte register in the internal Register File. When it could be unclear whether this variable refers to a source or a destination register, it is prefixed with an *S* or a *D*. |
| cadd | An address in the program code. |
| Dbreg * | A byte register in the internal Register File that serves as the destination of the instruction operation. |
| disp | Displacement. The distance between the end of an instruction and the target label. |
| Dwreg * | A word register in the internal Register File that serves as the destination of the instruction operation. Must be aligned on an address that is evenly divisible by 2. |
| lreg | A 32-bit register in the internal Register File. Must be aligned on an address that is evenly divisible by 4. |
| Sbreg * | A byte register in the internal Register File that serves as the source of the instruction operation. |
| Swreg * | A byte register in the internal Register File that serves as the source of the instruction operation. Must be aligned on an address that is evenly divisible by 2. |
| waop | A word operand that is addressed by any address mode. |
| wreg | A word register in the internal Register File. When it could be unclear whether this variable refers to a source or a destination register, it is prefixed with an *S* or a *D*. Must be aligned on an address that is evenly divisible by 2. |
| xxx | The three high-order bits of displacement. |

* The *D* or *S* prefix is used only when it could be unclear whether a variable refers to a destination or a source register.

## Table A-2. Instruction Set

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| ADD<br>(2 operands) | ADD WORDS. Adds the source and destination word operands and stores the sum into the destination operand.<br>(DEST) ← (DEST) + (SRC) | DEST, SRC<br>ADD wreg, waop<br>(011001aa) (waop) (wreg) |
| ADD<br>(3 operands) | ADD WORDS. Adds the two source word operands and stores the sum into the destination operand.<br>(DEST) ← (SRC1) + (SRC2) | DEST, SRC1, SRC2<br>ADD Dwreg, Swreg, waop<br>(010001aa) (waop) (Swreg) (Dwreg) |
| ADDB<br>(2 operands) | ADD BYTES. Adds the source and destination byte operands and stores the sum into the destination (leftmost) operand.<br>(DEST) ← (DEST) + (SRC) | DEST, SRC<br>ADDB breg, baop<br>(011101aa) (baop) (breg) |
| ADDB<br>(3 operands) | ADD BYTES. Adds the two source byte operands and stores the sum into the destination operand.<br>(DEST) ← (SRC1) + (SRC2) | DEST, SRC1, SRC2<br>ADDB Dbreg, Sbreg, baop<br>(010101aa) (baop) (breg) |
| ADDC | ADD WORDS WITH CARRY. Adds the source and destination word operands and stores the sum and the carry flag (0 or 1) into the destination operand.<br>(DEST) ← (DEST) + (SRC) + C | DEST, SRC<br>ADDC wreg, waop<br>(101001aa) (waop) (wreg) |
| ADDCB | ADD BYTES WITH CARRY. Adds the source and destination byte operands and stores the sum and the carry flag (0 or 1) into the destination operand.<br>(DEST) ← (DEST) + (SRC) + C | DEST, SRC<br>ADDCB breg, baop<br>(101101aa) (baop) (breg) |
| AND<br>(2 operands) | LOGICAL AND WORDS. ANDs the source and destination word operands and stores the result into the destination operand. The result has ones in the bit positions in which both operands had a "1" and zeros in all other bit positions.<br>(DEST) ← (DEST) AND (SRC) | DEST, SRC<br>AND wreg, waop<br>(011000aa) (waop) (wreg) |
| AND<br>(3 operands) | LOGICAL AND WORDS. ANDs the two source word operands and stores the result into the destination operand. The result has ones in only the bit positions in which both operands had a "1" and zeros in all other bit positions.<br>(DEST) ← (SRC1) AND (SRC2) | DEST, SRC1, SRC2<br>AND Dwreg, Swreg, waop<br>(010000aa) (waop) (Swreg) (Dwreg) |
| ANDB<br>(2 operands) | LOGICAL AND BYTES. ANDs the source and destination byte operands and stores the result into the destination operand. The result has ones in only the bit positions in which both operands had a "1" and zeros in all other bit positions.<br>(DEST) ← (DEST) AND (SRC) | DEST, SRC<br>ANDB breg, baop<br>(011100aa) (baop) (breg) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| ANDB<br>(3 operands) | LOGICAL AND BYTES. ANDs the two source byte operands and stores the result into the destination operand. The result has ones in only the bit positions in which both operands had a "1" and zeros in all other bit positions.<br><br>(DEST) ← (SRC1) AND (SRC2) | DEST, SRC1, SRC2<br><br>ANDB Dbreg, Sbreg, baop<br><br>(010100aa) (baop) (Sbreg) (Dbreg) |
| BMOV | BLOCK MOVE. Moves a block of word data from one location in memory to another. The source and destination addresses are calculated using the indirect with auto-increment addressing mode. A long register addresses the source and destination pointers, which are stored in adjacent word registers. A word register (CNTREG) specifies the number of transfers. The blocks of data can reside anywhere in memory, but should not overlap. | DEST, SRC<br><br>BMOV lreg, wreg<br><br>(11000001) (wreg) (lreg)<br><br>**NOTE:** CNTREG is not decremented during this instruction. It is easy to unintentionally create a long, uninterruptable operation with the BMOV instruction. Use the BMOVI instruction for an interruptable operation. |
| BMOVI | INTERRUPTABLE BLOCK MOVE. Moves a block of word data from one location in memory to another. The instruction is identical to BMOV, except that BMOVI is interruptable. The source and destination addresses are calculated using the indirect with auto-increment addressing mode. A long register addresses the source and destination pointers, which are stored in adjacent word registers. A word register (CNTREG) specifies the number of transfers. The blocks of data can reside anywhere in memory, but should not overlap.<br><br>COUNT ← (CNTREG)<br>LOOP: SRCPTR ← (PTRS)<br>DSTPTR ← (PTRS + 2)<br>(DSTPTR) ← (SRCPTR)<br>(PTRS) ← SRCPTR + 2<br>(PTRS + 2) ← DSTPTR + 2<br>COUNT ← COUNT 1<br>if COUNT ≠ 0 then<br>go to LOOP | DEST, SRC<br><br>BMOVI lreg, wreg<br><br>(11001101) (wreg) (lreg)<br><br>**NOTE:** CNTREG is not decremented unless the instruction is interrupted. When BMOVI is interrupted, CNTREG is updated to store the interim word count at the time of the interrupt. For this reason, you should always reload CNTREG before starting a BMOVI. |
| BR | BRANCH INDIRECT. Continues execution at the address specified in the operand word register.<br><br>PC ← (DEST) | DEST<br><br>BR [wreg]<br><br>(11100011) (wreg) |
| CLR | CLEAR WORD. Clears the value of the operand.<br><br>(DEST) ← 0 | DEST<br><br>CLR wreg<br><br>(00000001) (wreg) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| CLRB | CLEAR BYTE. Clears the value of the operand.<br>(DEST) ← 0 | DEST<br>CLRB breg<br>(00010001) (breg) |
| CLRC | CLEAR CARRY FLAG. Clears the carry flag.<br>C ← 0 | CLRC<br>(11111000) |
| CLRVT | CLEAR OVERFLOW-TRAP FLAG. Clears the overflow-trap flag.<br>VT ← 0 | CLRVT<br>(11111100) |
| CMP | COMPARE WORDS. Subtracts the source word operand from the destination word operand. The flags are altered, but the operands remain unaffected. If a borrow occurs, the carry flag is cleared; otherwise it is set.<br>(DEST) – (SRC) | DEST, SRC<br>CMP wreg, waop<br>(100010aa) (waop) (wreg) |
| CMPB | COMPARE BYTES. Subtracts the source byte operand from the destination byte operand. The flags are altered, but the operands remain unaffected. If a borrow occurs, the carry flag is cleared; otherwise it is set.<br>(DEST) – (SRC) | DEST, SRC<br>CMPB breg, baop<br>(100110aa) (baop) (breg) |
| CMPL | COMPARE LONG. Compares the magnitudes of two double-word (long) operands. The operands are specified using the direct addressing mode. The flags are altered, but the operands remain unaffected. If a borrow occurs, the carry flag is cleared; otherwise, it is set.<br>(DEST) – (SRC) | DEST, SRC<br>CMPL lreg, lreg<br>(11000101) (src lreg) (dest#lreg) |
| DEC | DECREMENT WORD. Decrements the value of the operand by one.<br>(DEST) ← (DEST)–1 | DEST<br>DEC wreg<br>(00000101) (wreg) |
| DECB | DECREMENT BYTE. Decrements the value of the operand by one.<br>(DEST) ← (DEST)–1 | DEST<br>DECB breg<br>(00010101) (breg) |
| DI | DISABLE INTERRUPTS. Disables interrupts. Interrupt-calls cannot occur after this instruction.<br>Interrupt Enable (PSW.1) ← 0 | DI<br>(11111010) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| DIV | DIVIDE INTEGERS. Divides the contents of the destination **long-integer** operand by the contents of the source **integer** word operand, using signed arithmetic. It stores the quotient into the low-order word of the destination (i.e., the word with the lower address) and the remainder into the high-order word.<br><br>(low word DEST) ← (DEST) / (SRC)<br>(high word DEST) ← (DEST) MOD(SRC) | DEST, SRC<br><br>DIV lreg, waop<br><br>(11111110) (100011aa) (waop) (lreg) |
| DIVB | DIVIDE SHORT-INTEGERS. Divides the contents of the destination **integer** operand by the contents of the source **short-integer** operand, using signed arithmetic. It stores the quotient into the low-order word of the destination (i.e., the word with the lower address) and the remainder into the high-order word.<br><br>(low byte DEST) ← (DEST) / (SRC)<br>(high byte DEST) ← (DEST) MOD (SRC) | DEST, SRC<br><br>DIVB wreg, baop<br><br>(11111110) (100111aa) (baop) (wreg) |
| DIVU | DIVIDE WORDS, UNSIGNED. Divides the content of the destination **double-word** operand by the contents of the source **word** operand, using unsigned arithmetic. It stores the quotient into the low-order word (i.e., the word with the lower address) of the destination operand and the remainder into the high-order word. The following two statements are performed concurrently.<br><br>(low word DEST) ← (DEST) / (SRC)<br>(high word DEST) ← (DEST) MOD (SRC) | DEST, SRC<br><br>DIVU lreg, waop<br><br>(100011aa) (waop) (lreg) |
| DIVUB | DIVIDE BYTES, UNSIGNED. This instruction divides the contents of the destination **word** operand by the contents of the source **byte** operand, using unsigned arithmetic. It stores the quotient into the low-order word (i.e., the word with the lower address) of the destination operand and the remainder into the high-order word. The following two statements are performed concurrently.<br><br>(low byte DEST) ← (DEST) / (SRC)<br>(high byte DEST)← (DEST) MOD (SRC) | DEST, SRC<br><br>DIVUB wreg, baop<br><br>(100111aa) (baop) (wreg) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| DJNZ | DECREMENT AND JUMP IF NOT ZERO. Decrements the value of the byte operand by 1. If the result is 0, control passes to the next sequential instruction. If the result is not equal to 0, the instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.<br><br>(COUNT) ← (COUNT)–1<br>if (COUNT) ≠ 0 then<br>  PC ← PC + disp (Note 1)<br>end_if | DJNZ breg,cadd<br>(11100000) (breg) (disp) |
| DJNZW | DECREMENT AND JUMP IF NOT ZERO WORD. Decrements the value of the word operand by 1. If the result is 0, control passes to the next sequential instruction. If the result is not equal to 0, the instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127<br><br>(COUNT) ← (COUNT)–1<br>if (COUNT) ≠ 0 then<br>  PC ← PC + disp (Note 1)<br>end_if | DJNZW wreg,cadd<br>(11100001) (wreg) (disp) |
| DPTS | DISABLE PERIPHERAL TRANSACTION SERVER (PTS). Disables the Peripheral Transaction Server (PTS).<br><br>PTS Disable (PSW.2) ← 0 | DPTS<br>(11101100) |
| EI | ENABLE INTERRUPTS. Enables interrupts following the execution of the next statement. Interrupt-calls cannot occur immediately following this instruction.<br><br>Interrupt Enable (PSW.1) ← 1 | EI<br>(11111011) |
| EPTS | ENABLE PERIPHERAL TRANSACTION SERVER (PTS). Enables the Peripheral Transaction Server (PTS).<br><br>PTS Enable (PSW.2) ← 1 | EPTS<br>(11101101) |
| EXT | SIGN-EXTEND INTEGER INTO LONG-INTEGER. Sign-extends the low-order word of the operand throughout the high-order word of the operand.<br><br>if (low word DEST) < 8000H then<br>  (high word DEST) ← 0<br>else<br>  (high word DEST) ← 0FFFFH<br>end_if | EXT lreg<br>(00000110) (lreg) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| EXTB | SIGN-EXTEND SHORT-INTEGER INTO INTEGER. Sign-extends the low-order byte of the operand throughout the high-order byte of the operand.<br><br>if (low byte DEST) < 80H then<br>  (high byte DEST) ← 0<br>else<br>  (high byte DEST) ← 0FFH<br>end_if | EXTB wreg<br><br>(00010110) (wreg) |
| IDLPD | IDLE/POWERDOWN. Depending on the 8-bit value of the KEY operand, this instruction causes the part<br><br>• to enter Idle mode (KEY=1),<br>• to enter Powerdown mode (KEY=2),<br>• to execute a reset sequence (KEY = any value other than 1 or 2).<br><br>The bus controller completes any prefetch cycle in progress before the CPU stops or resets.<br><br>if KEY = 1 then<br>  enter Idle<br>else<br>  if KEY = 2 then<br>  enter Powerdown<br>  else<br>  execute reset | IDLPD #key<br><br>(11110110) (key) |
| INC | INCREMENT WORD. Increments the value of the word operand by 1.<br><br>(DEST) ← (DEST) + 1 | INC wreg<br><br>(00000111) (wreg) |
| INCB | INCREMENT BYTE. Increments the value of the byte operand by 1.<br><br>(DEST) ← (DEST) + 1 | INCB breg<br><br>(00010111) (breg) |
| JBC | JUMP IF BIT IS CLEAR. Tests the specified bit. If the bit is set, control passes to the next sequential instruction. If the bit is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.<br><br>if (specified bit) = 0 then<br>  PC ← PC + disp (Note 1) | JBC breg,bitno,cadd<br><br>(00110bbb) (breg) (disp) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| JBS | JUMP IF BIT IS SET. Tests the specified bit. If the bit is clear, control passes to the next sequential instruction. If the bit is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.<br><br>if (specified bit) = 1 then<br>PC ← PC + disp (Note 1) | JBS breg,bitno,cadd<br>(00111bbb) (breg) (disp) |
| JC | JUMP IF CARRY FLAG IS SET. Tests the carry flag. If the carry flag is clear, control passes to the next sequential instruction. If the carry flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.<br><br>if C = 1 then<br>PC ← PC + disp (Note 1) | JC cadd<br>(11011011) (disp) |
| JE | JUMP IF EQUAL. Tests the zero flag. If the flag is clear, control passes to the next sequential instruction. If the zero flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.<br><br>if Z = 1 then<br>PC ← PC + disp (Note 1) | JE cadd<br>(11011111) (disp) |
| JGE | JUMP IF SIGNED GREATER THAN OR EQUAL. Tests the negative flag. If the negative flag is set, control passes to the next sequential instruction. If the negative flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.<br><br>if N = 0 then<br>PC ← PC + disp (Note 1) | JGE cadd<br>(11010110) (disp) |
| JGT | JUMP IF SIGNED GREATER THAN. Tests both the zero flag and the negative flag. If either flag is set, control passes to the next sequential instruction. If both flags are clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.<br><br>if N = 0 AND Z = 0 then<br>PC ← PC + disp (Note 1) | JGT cadd<br>(11010010) (disp) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| JH | JUMP IF HIGHER (UNSIGNED). Tests both the zero flag and the carry flag. If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction. If the carry flag is set and the zero flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.<br><br>if C = 1 AND Z = 0 then<br>PC ← PC + disp (Note 1) | JH cadd<br>(11011001) (disp) |
| JLE | JUMP IF SIGNED LESS THAN OR EQUAL. Tests both the negative flag and the zero flag. If both flags are clear, control passes to the next sequential instruction. If either flag is set, this instruction adds to the program counter the offset between the end of this instruction nad the target label, effecting the jump. The offset must be in the range of –128 to +127.<br><br>if N = 1 OR Z = 1 then<br>PC ← PC + disp (Note 1) | JLE cadd<br>(11011010) (disp) |
| JLT | JUMP IF SIGNED LESS THAN. Tests the negative flag. If the flag is clear, control passes to the next sequential instruction. If the flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –128 to +127.<br><br>if N = 1 then<br>PC ← PC + disp (Note 1) | JLT cadd<br>(11011110) (disp) |
| JNC | JUMP IF CARRY FLAG IS CLEAR. Tests the carry flag. If the flag is set, control passes to the next sequential instruction. If the carry flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label. The offset must be in the range of –128 to +127.<br><br>if C = 0 then<br>PC ← PC + disp (Note 1) | JNC cadd<br>(11010011) (disp) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| JNE | JUMP IF NOT EQUAL. Tests the zero flag. If the flag is set, control passes to the next sequential instruction. If the zero flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label. The offset must be in the range of –128 to +127.<br><br>if Z = 0 then<br>PC ← PC + disp (Note 1) | JNE cadd<br>(11010111) (disp) |
| JNH | JUMP IF NOT HIGHER (UNSIGNED). Tests both the zero flag and the carry flag. If the carry flag is set and the zero flag is clear, control passes to the next sequential instruction. If either the carry flag is set or the zero flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.<br><br>if C = 0 OR Z = 1 then<br>PC ← PC + disp (Note 1) | JNH cadd<br>(11010001) (disp) |
| JNST | JUMP IF STICKY BIT FLAG IS CLEAR. Tests the sticky bit flag. If the flag is set, control passes to the next sequential instruction. If the sticky bit flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.<br><br>if ST = 0 then<br>PC ← PC + disp (Note 1) | JNST cadd<br>(11010000) (disp) |
| JNV | JUMP IF OVERFLOW FLAG IS CLEAR. Tests the overflow flag. If the flag is set, control passes to the next sequential instruction. If the overflow flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.<br><br>if V = 0 then<br>PC ← PC + disp (Note 1) | JNV cadd<br>(11010101) (disp) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| JNVT | JUMP IF OVERFLOW-TRAP FLAG IS CLEAR. Tests the overflow-trap flag. If the flag is set, this instruction clears the flag and passes control to the next sequential instruction. If the overflow-trap flag is clear, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.<br><br>if VT = 0 then<br>PC ← PC + disp (Note 1) | JNVT cadd<br>11010100) (disp) |
| JST | JUMP IF STICKY BIT FLAG IS SET. Tests the sticky bit flag. If the flag is clear, control passes to the next sequential instruction. If the sticky bit flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127..<br><br>if ST = 1 then<br>PC ← PC + disp (Note 1) | JST cadd<br>(11011000) (disp) |
| JV | JUMP IF OVERFLOW FLAG IS SET. Tests the overflow flag. If the flag is clear, control passes to the next sequential instruction. If the overflow flag is set, this instruction adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.<br><br>if V = 1 then<br>PC ← PC + disp (Note 1) | JV cadd<br>(11011101) (disp) |
| JVT | JUMP IF OVERFLOW-TRAP FLAG IS SET. Tests the overflow-trap flag. If the flag is clear, control passes to the next sequential instruction. If the overflow-trap flag is set, this instruction clears the flag and adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in range of –128 to +127.<br><br>if VT = 1 then<br>PC ← PC + disp (Note 1) | JVT cadd<br>(11011100) (disp) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| LCALL | LONG CALL. PUSHes the contents of the program counter (the return address) onto the stack, then adds to the program counter the offset between the end of this instruction and the target label, effecting the call. The operand may be any address in the entire address space.<br><br>SP ← SP – 2<br>(SP) ← PC<br>PC ← PC + disp | LCALL cadd<br>(11101111) (disp-low) (disp-high) |
| LD | LOAD WORD. Loads the value of the source word operand into the destination operand.<br><br>(DEST) ← (SRC) | DEST, SRC<br>LD wreg, waop<br>(101000aa) (waop) (wreg) |
| LDB | LOAD BYTE. Loads the value of the source byte operand into the destination operand.<br><br>(DEST) ← (SRC) | DEST, SRC<br>LDB breg, baop<br>(101100aa) (baop) (breg) |
| LDBSE | LOAD BYTE SIGN-EXTENDED. Sign-extends the value of the source **short-integer** operand and loads it into the destination **integer** operand.<br><br>(low byte DEST) ← (SRC)<br>if (SRC) < 80h then<br>(high byte DEST) ← 0<br>else<br>(high byte DEST) ← 0FFH<br>end_if | DEST, SRC<br>LDBSE wreg, baop<br>(101111aa) (baop) (wreg) |
| LDBZE | LOAD BYTE ZERO-EXTENDED. Zero-extends the value of the source **byte** operand and loads it into the destination **word** operand.<br><br>(low byte DEST) ← (SRC)<br>(high byte DEST) ← 0 | DEST, SRC<br>LDBZE wreg, baop<br>(101011aa) (baop) (wreg) |
| LJMP | LONG JUMP. Adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The operand may be any address in the entire address space.<br><br>PC ← PC + disp | LJMP cadd<br>(11100111) (disp-low) (disp-high) |
| MUL<br>(2 operands) | MULTIPLY INTEGERS. Multiplies the source and destination **integer** operands, using signed arithmetic, and stores the 32-bit result into the destination **long-integer** operand. The sticky bit flag is undefined after the instruction is executed.<br><br>(DEST) ← (DEST) × (SRC) | DEST, SRC<br>MUL lreg, waop<br>(11111110) (010111aa) (waop) (lreg) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| MUL<br>(3 operands) | MULTIPLY INTEGERS. Multiplies the two source **integer** operands, using signed arithmetic, and stores the 32-bit result into the destination **long-integer** operand. The sticky bit flag is undefined after the instruction is executed.<br>(DEST) ← (SRC1) x (SRC2) | DEST, SRC1, SRC2<br>MUL lreg, wreg, waop<br>(11111110) (010011aa) (waop) (wreg) (lreg) |
| MULB<br>(2 operands) | MULTIPLY SHORT-INTEGERS. Multiplies the source and destination **short-integer** operands, using signed arithmetic, and stores the 16-bit result into the destination **integer** operand. The sticky bit flag is undefined after the instruction is executed.<br>(DEST) ← (DEST) x (SRC) | DEST, SRC<br>MULB wreg, baop<br>(11111110) (011111aa) (baop) (wreg) |
| MULB<br>(3 operands) | MULTIPLY SHORT-INTEGERS. Multiplies the two source **short-integer** operands, using signed arithmetic, and stores the 16-bit result into the destination **integer** operand. The sticky bit flag is undefined after the instruction is executed.<br>(DEST) ← (SRC1) x (SRC2) | DEST, SRC1, SRC2<br>MULB wreg, breg, baop<br>(11111110) (010111aa) (baop) (breg) (wreg) |
| MULU<br>(2 operands) | MULTIPLY WORDS, UNSIGNED. Multiplies the source and destination **word** operands, using unsigned arithmetic, and stores the 32-bit result into the destination **double-word** operand. The sticky bit flag is undefined after the instruction is executed.<br>(DEST) ← (DEST) x (SRC) | DEST, SRC<br>MULU lreg, waop<br>(011011aa) (waop) (lreg) |
| MULU<br>(3 operands) | MULTIPLY WORDS, UNSIGNED. Multiplies the two source **word** operands, using unsigned arithmetic, and stores the 32-bit result into the destination **double-word** operand. The sticky bit flag is undefined after the instruction is executed.<br>(DEST) ← (SRC1) x (SRC2) | DEST, SRC1, SRC2<br>MULU lreg, wreg, waop<br>(010011aa) (waop) (wreg) (lreg) |
| MULUB<br>(2 operands) | MULTIPLY BYTES, UNSIGNED. Multiplies the source and destination **byte** operands, using unsigned arithmetic, and stores the **word** result into the destination operand. The sticky bit flag is undefined after the instruction is executed.<br>(DEST) ← (DEST) x (SRC) | DEST, SRC<br>MULUB wreg, baop<br>(011111aa) (baop) (wreg) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| MULUB (3 operands) | MULTIPLY BYTES, UNSIGNED. Multiplies the two source **byte** operands, using unsigned arithmetic, and stores the **word** result into the destination operand. The sticky bit flag is undefined after the instruction is executed.<br>(DEST) ← (SRC1) × (SRC2) | DEST, SRC1, SRC2<br>MULUB wreg, breg, baop<br>(010111aa) (baop) (breg) (wreg) |
| NEG | NEGATE INTEGER. Negates the value of the **integer** operand.<br>(DEST) ← –(DEST) | NEG wreg<br>(00000011) (wreg) |
| NEGB | NEGATE SHORT-INTEGER. Negates the value of the **short-integer** operand.<br>(DEST) ← –(DEST) | NEGB breg<br>(00010011) (breg) |
| NOP | NO OPERATION. Does nothing. Control passes to the next sequential instruction. | NOP<br>(11111101) |
| NORML | NORMALIZE LONG-INTEGER. Normalizes the source **(leftmost) long-integer** operand. (That is, it shifts the operand to the left until its most significant bit is "1" or until it has performed 31 shifts). If the most significant bit is still "0" after 31 shifts, the instruction stops the process and sets the zero flag. The instruction stores the actual number of shifts performed in the destination **(rightmost)** operand.<br>(COUNT) ← 0<br>do while (MSB(DEST) ← 0) AND (COUNT) < 31)<br>(DEST) ← (DEST) × 2<br>(COUNT) ← (COUNT) + 1<br>end_while | **SRC, DEST**<br>NORML lreg, breg<br>(00001111) (breg) (lreg) |
| NOT | COMPLEMENT WORD. Complements the value of the word operand (replaces each "1" with a "0" and each "0" with a "1").<br>(DEST) ← NOT (DEST) | NOT wreg<br>(00000010) (wreg) |
| NOTB | COMPLEMENT BYTE. Complements the value of the byte operand (replaces each "1" with a "0" and each "0" with a "1").<br>(DEST) ← NOT (DEST) | NOTB breg<br>(00010010) (breg) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| OR | LOGICAL OR WORDS. ORs the source word operand with the destination word operand and replaces the original destination operand with the result. The result has a "1" in each bit position in which either the source or destination operand had a "1".<br><br>(DEST) ← (DEST) OR (SRC) | DEST, SRC<br>OR wreg, waop<br>(100000aa) (waop) (wreg) |
| ORB | LOGICAL OR BYTES. ORs the source byte operand with the destination byte operand and replaces the original destination operand with the result. The result has a "1" in each bit position in which either the source or destination operand had a "1".<br><br>(DEST) ← (DEST) OR (SRC) | DEST, SRC<br>ORB breg, baop<br>(100100aa) (baop) (breg) |
| POP | POP WORD. Pops the word on top of the stack and places it at the destination operand.<br><br>(DEST) ← (SP)<br>SP ← SP + 2 | POP waop<br>(110011aa) (waop) |
| POPA | POP ALL. This instruction is used instead of POPF, to support the eight additional interrupts. It pops two words off the stack and places the first word into the INT_MASK1/WSR register and the second word into the PSW/INT_MASK register-pair. This instruction increments the SP by 4. Interrupt-calls cannot occur immediately following this instruction.<br><br>INT_MASK1/WSR ← (SP)<br>SP ← SP + 2<br>PSW/INT_MASK ← (SP)<br>SP ← SP + 2 | POPA<br>(11110101) |
| POPF | POP FLAGS. Pops the word on top of the stack and places it into the PSW. Interrupt-calls cannot occur immediately following this instruction.<br><br>(PSW) ← (SP)<br>SP ← SP + 2 | POPF<br>(11110011) |
| PUSH | PUSH WORD. Pushes the word operand onto the stack.<br><br>SP ← SP – 2<br>(SP) ← (DEST) | PUSH waop<br>(110010aa) (waop) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| PUSHA | PUSH ALL. This instruction is used instead of PUSHF, to support the eight additional interrupts. It pushes two words onto the stack — PSW/INT_MASK and the word formed by the INT_MASK1/WSR register-pair.<br><br>This instruction clears the PSW, INT_MASK, and INT_MASK1 registers and decrements the SP by 4. Interrupt-calls cannot occur immediately following this instruction.<br><br>SP ← SP 2<br>(SP) ← PSW/INT_MASK<br>PSW/INT_MASK ← 0<br>SP ← SP 2<br>(SP) ← INT_MASK1/WSR<br>INT_MASK1 ← 0 | PUSHA<br>(11110100) |
| PUSHF | PUSH FLAGS. Pushes the PSW onto the top of the stack, then sets it to zeros. This implies that all interrupts are disabled. Interrupt-calls cannot occur immediately following this instruction.<br><br>SP ← SP – 2<br>(SP) ← PSW/INT_MASK<br>PSW/INT_MASK ← 0 | PUSHF<br>(11110010) |
| RET | RETURN FROM SUBROUTINE. Pops the PC off the top of the stack.<br><br>PC ← (SP)<br>SP ← SP + 2 | RET<br>(11110000) |
| RST | RESET SYSTEM. Initializes the PSW to zero, the PC to 2080H, and the SFRs to their initial values. Executing this instruction causes the RESET# pin to be pulled low for 16 state times.<br><br>SFR Reset Status<br>Pin Reset Status<br>PSW ← 0<br>PC ← 2080H | RST<br>(11111111) |
| SCALL | SHORT CALL. Pushes the contents of the program counter (the return address) onto the stack, then adds to the program counter the offset between the end of this instruction and the target label. The offset must be in the range of –1024 to +1023, inclusive.<br><br>SP ← SP–2<br>(SP) ← PC<br>PC ← PC + disp (Note 1) | SCALL cadd<br>(00101xxx) (disp-low) |
| SETC | SET CARRY FLAG. Sets the carry flag.<br><br>C ← 1 | SETC<br>(11111001) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| SHL | SHIFT WORD LEFT. Shifts the destination word operand to the left as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register with a value in the range of 0 to 31 (1FH), inclusive. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.<br><br>Temp ← (COUNT)<br>do while Temp ≠ 0<br>C ← High order bit of (DEST)<br>(DEST) ← (DEST) x 2<br>Temp ← Temp – 1<br>end_while | SHL wreg,#count<br>(00001001) (count) (wreg)<br>**or**<br>SHL wreg,breg<br>(00001001) (breg) (wreg) |
| SHLB | SHIFT BYTE LEFT. Shifts the destination byte operand to the left as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register with a value in the range of 0 to 31 (1FH), inclusive. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.<br><br>Temp ← (COUNT)<br>do while Temp ≠ 0<br>C ← High order bit of (DEST)<br>(DEST) ← (DEST) x 2<br>Temp ← Temp – 1<br>end_while | SHLB breg,#count<br>(00011001) (count) (breg)<br>**or**<br>SHLB breg,breg<br>(00011001) (breg) (breg) |
| SHLL | SHIFT DOUBLE-WORD LEFT. Shifts the destination double-word operand to the left as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register with a value in the range of 0 to 31 (1FH), inclusive. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.<br><br>Temp ← (COUNT)<br>do while Temp ≠ 0<br>C ← High order bit of (DEST)<br>(DEST) ← (DEST) x 2<br>Temp ← Temp – 1<br>end_while | SHLL lreg,#count<br>(00001101) (count) (breg)<br>**or**<br>SHLL lreg,breg<br>(00001101) (breg) (lreg) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| SHR | LOGICAL RIGHT SHIFT WORD. Shifts the destination word operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register with a value in the range of 0 to 31 (1FH), inclusive. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.<br><br>This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a "1" is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.<br><br>Temp ← (COUNT)<br>do while Temp ≠ 0<br>C ← Low order bit of (DEST)<br>(DEST) ← (DEST)/2 (Note 2)<br>Temp ← Temp – 1<br>end_while | SHR wreg,#count<br>(00001000) (count) (wreg)<br>**or**<br>SHR wreg,breg<br>(00001000) (breg) (wreg) |
| SHRA | ARITHMETIC RIGHT SHIFT WORD. Shifts the destination word operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register with a value in the range of 0 to 31 (1FH), inclusive. If the original high order bit value was "0," zeroes are shifted in. If the value was "1," ones are shifted in. The last bit shifted out is saved in the carry flag.<br><br>This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a "1" is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.<br><br>Temp ← (COUNT)<br>do while Temp ≠ 0<br>C ← Low order bit of (DEST)<br>(DEST) ← (DEST)/2 (Note 3)<br>Temp ← Temp – 1<br>end_while | SHRA wreg,#count<br>(00001010) (count) (wreg)<br>**or**<br>SHRA wreg,breg<br>(00001010) (breg) (wreg) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| SHRAB | ARITHMETIC RIGHT SHIFT BYTE. Shifts the destination byte operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register with a value in the range of 0 to 31 (1FH), inclusive. If the original high order bit value was "0," zeroes are shifted in. If the value was "1," ones are shifted in. The last bit shifted out is saved in the carry flag.<br><br>This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a "1" is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.<br><br>Temp ← (COUNT)<br>do while Temp ≠ 0<br>C = Low order bit of (DEST)<br>(DEST) ← (DEST)/2 (Note 3)<br>Temp ← Temp – 1<br>end_while | SHRAB breg,#count<br>(00011010) (count) (breg)<br>**or**<br>SHRAB breg,breg<br>(00011010) (breg) (breg) |
| SHRAL | ARITHMETIC RIGHT SHIFT DOUBLE-WORD. Shifts the destination double-word operand to the right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register with a value in the range of 0 to 31 (1FH), inclusive. If the original high order bit value was "0," zeroes are shifted in. If the value was "1," ones are shifted in.<br><br>This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a "1" is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.<br><br>Temp ← (COUNT)<br>do while Temp ≠ 0<br>C ← Low order bit of (DEST)<br>(DEST) ← (DEST)/2 (Note 3)<br>Temp ← Temp – 1<br>end_while | SHRAL lreg,#count<br>(00001110) (count) (lreg)<br>**or**<br>SHRAL lreg,breg<br>(00001110) (breg) (lreg) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| SHRB | LOGICAL RIGHT SHIFT BYTE. Shifts the destination byte operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register with a value in the range of 0 to 31 (1FH), inclusive. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.<br><br>This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a "1" is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.<br><br>Temp ← (COUNT)<br>do while Temp ≠ 0<br>C ← Low order bit of (DEST)<br>(DEST) ← (DEST)/2 (Note 2)<br>Temp ← Temp–1<br>end_while | SHRB breg,#count<br>(00011000) (count) (breg)<br>**or**<br>SHRB breg,breg<br>(00011000) (breg) (breg) |
| SHRL | LOGICAL RIGHT SHIFT DOUBLE-WORD. Shifts the destination double-word operand to the right as many times as specified by the count operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH), inclusive, or as the content of any register with a value in the range of 0 to 31 (1FH), inclusive. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.<br><br>This instruction clears the sticky bit flag at the beginning of the instruction. If at any time during the shift a "1" is shifted into the carry flag and another shift cycle occurs, the instruction sets the sticky bit flag.<br><br>Temp ← (COUNT)<br>do while Temp ≠ 0<br>C ← Low order bit of (DEST)<br>(DEST) ← (DEST)/2 (Note 2)<br>Temp ← Temp – 1<br>end_while | SHRL lreg,#count<br>(00001100) (count) (lreg)<br>**or**<br>SHRL lreg,breg<br>(00001100) (breg) (lreg) |
| SJMP | SHORT JUMP. Adds to the program counter the offset between the end of this instruction and the target label, effecting the jump. The offset must be in the range of –1024 to +1023, inclusive.<br><br>PC ← PC + disp (Note 1) | SJMP cadd<br>(00100xxx) (disp-low) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| SKIP | TWO BYTE NO-OPERATION. Does nothing. Control passes to the next sequential instruction. This is actually a two-byte NOP in which the second byte can be any value and is simply ignored. | SKIP breg<br>(00000000) (breg) |
| ST | STORE WORD. Stores the value of the source **(leftmost)** word operand into the destination **(rightmost)** operand.<br>(DEST) ← (SRC) | **SRC, DEST**<br>ST wreg, waop<br>(110000aa) (waop) (wreg) |
| STB | STORE BYTE. Stores the value of the source **(leftmost)** byte operand into the destination **(rightmost)** operand.<br>(DEST) ← (SRC) | **SRC, DEST**<br>STB breg, baop<br>(110001aa) (baop) (breg) |
| SUB<br>(2 operands) | SUBTRACT WORDS. Subtracts the source word operand from the destination word operand, stores the result in the destination operand, and sets the carry flag as the complement of borrow.<br>(DEST) ← (DEST) – (SRC) | DEST, SRC<br>SUB wreg, waop<br>(011010aa) (waop) (wreg) |
| SUB<br>(3 operands) | SUBTRACT WORDS. Subtracts the first source word operand from the second, stores the result in the destination operand, and sets the carry flag as the complement of borrow.<br>(DEST) ← (SRC1) – (SRC2) | DEST, SRC1, SRC2<br>SUB Dwreg, Swreg, waop<br>(010010aa) (waop) (Swreg) (Dwreg) |
| SUBB<br>(2 operands) | SUBTRACT BYTES. Subtracts the source byte operand from the destination byte operand, stores the result in the destination operand, and sets the carry flag as the complement of borrow.<br>(DEST) ← (DEST) – (SRC) | DEST, SRC<br>SUBB breg, baop<br>(010110aa) (baop) (breg) |
| SUBB<br>(3 operands) | SUBTRACT BYTES. Subtracts the first source byte operand from the second, stores the result in the destination operand, and sets the carry flag as the complement of borrow.<br>(DEST) ← (SRC1) – (SRC2) | DEST, SRC1, SRC2<br>SUBB Dbreg, Sbreg, baop<br>(010110aa) (baop) (Sbreg) (Dbreg) |
| SUBC | SUBTRACT WORDS WITH BORROW. Subtracts the source word operand from the destination word operand. If the carry flag was clear, SUBC subtracts 1 from the result. It stores the result in the destination operand and sets the carry flag as the complement of borrow.<br>(DEST) ← (DEST) – (SRC) – (1–C) | DEST, SRC<br>SUBC wreg, waop<br>(101010aa) (waop) (wreg) |

## Table A-2. Instruction Set (Continued)

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| SUBCB | SUBTRACT BYTES WITH BORROW. Subtracts the source byte operand from the destination byte operand. If the carry flag was clear, SUBCB subtracts 1 from the result. It stores the result in the destination operand and sets the carry flag as the complement of borrow.<br>(DEST) ← (DEST) – (SRC) – (1–C) | DEST, SRC<br>SUBCB breg, baop<br>(101110aa) (baop) (breg) |
| TIJMP | TABLE INDIRECT JUMP. Causes execution to continue at an address selected from a table of addresses. The first word register, TBASE, contains the 16-bit address of the beginning of the table. The second word register, INDEX, contains the 16-bit address of a byte that contains the index into the table. The INDEX value must be between 0 and 128. The #byte operand, INDEX_MASK, is 8-bit immediate data to mask INDEX.<br>INDEX_MASK is ANDed with INDEX to determine the offset (OFFSET). OFFSET is multiplied by two, then added to the base address (TBASE) to determine the destination address (DEST X).<br>[INDEX] AND INDEX_MASK = OFFSET<br>(2 × OFFSET) + TBASE = DEST X<br>PC ← (DEST X) | TBASE, INDEX, INDEX_MASK<br>TIJMP wreg, [wreg], #byte<br>(11100010) (wreg) (#byte) [wreg] |
| TRAP | SOFTWARE TRAP. This instruction causes an interrupt-call that is vectored through location 2010H. The operation of this instruction is not affected by the state of the interrupt enable flag in the PSW (I). Interrupt-calls cannot occur immediately following this instruction.<br>SP ← SP–2<br>(SP) ← PC<br>PC ← (2010H) | TRAP<br>(11110111)<br>**NOTE:** This instruction is not supported by revision 1.2 of the 8096 assembly language. The TRAP instruction is intended for use by Intel-provided tools. These tools will not support user-application of this instruction. |
| XCH | EXCHANGE WORD. Exchanges the value of the source word operand with that of the destination word operand.<br>(DEST) ← (SRC) | DEST, SRC<br>XCH wreg, waop<br>(00000100) (waop) (wreg)<br>(00001011) (waop) (wreg) |
| XCHB | EXCHANGE BYTE. Exchanges the value of the source byte operand with that of the destination byte operand.<br>(DEST) ← (SRC) | DEST, SRC<br>XCHB breg, baop<br>(00010100) (baop) (breg)<br>(00011011) (baop) (breg) |

**Table A-2. Instruction Set (Continued)**

| Mnemonic | Operation | Instruction Format |
|---|---|---|
| XOR | LOGICAL EXCLUSIVE-OR WORDS. XORs the source word operand with the destination word operand and stores the result in the destination operand. The result has ones in the bit positions in which either operand (but not both) had a "1" and zeros in all other bit positions.<br><br>(DEST) ← (DEST) XOR (SRC) | DEST, SRC<br><br>XOR wreg, waop<br><br>(100001aa) (waop) (wreg) |
| XORB | LOGICAL EXCLUSIVE-OR BYTES. XORs the source byte operand with the destination byte operand and stores the result in the destination operand. The result has ones in the bit positions in which either operand (but not both) had a "1" and zeros in all other bit positions.<br><br>(DEST) ← (DEST) XOR (SRC) | DEST, SRC<br><br>XORB breg, baop<br><br>(100101aa) (baop) (breg) |

**NOTES:**

1. The displacement (disp) is sign-extended to 16 bits.
2. In this operation, DEST/2 represents unsigned division.
3. In this operation, DEST/2 represents signed division.

Tables A-3 and A-4 define the abbreviations and symbols used in Tables A-5 and A-6. Table A-5 shows the effect of each instruction on the Program Status Word flags, and Table A-6 shows the effect of the PSW flags or a specified register bit on conditional jump instructions. (For additional information about the PSW flags, refer to the Program Status Word description in Appendix C.)

**Table A-3. PSW Flag Abbreviations**

| Abbreviation | PSW Flag Name |
|---|---|
| C | Carry Flag |
| N | Negative Flag |
| ST | Sticky Bit Flag |
| V | Overflow Flag |
| VT | Overflow-Trap Flag |
| Z | Zero Flag |

**Table A-4. PSW Flag Setting Symbols**

| Symbol | Description |
|---|---|
| √ | The instruction sets or clears the flag, as appropriate. |
| — | The instruction does not modify the flag. |
| ↓ | The instruction may clear the flag, if it is appropriate, but cannot set it. |
| ↑ | The instruction may set the flag, if it is appropriate, but cannot clear it. |
| 1 | The instruction sets the flag. |
| 0 | The instruction clears the flag. |
| ? | The instruction leaves the flag in an indeterminate state. |

## Table A-5. Effects of Instructions on PSW Flag Settings

| Mnemonic | PSW Flag Settings | | | | | |
|---|---|---|---|---|---|---|
| | Z | N | C | V | VT | ST |
| ADD, ADDB | √ | √ | √ | √ | ↑ | — |
| ADDC, ADDCB | ↓ | √ | √ | √ | ↑ | — |
| AND, ANDB | √ | √ | 0 | 0 | — | — |
| BMOV, BMOVI | — | — | — | — | — | — |
| BR (Indirect) | — | — | — | — | — | — |
| CLR, CLRB | 1 | 0 | 0 | 0 | — | — |
| CLRC | — | — | 0 | — | — | — |
| CLRVT | — | — | — | — | 0 | — |
| CMP, CMPB | √ | √ | √ | √ | ↑ | — |
| CMPL | √ | √ | √ | √ | √ | — |
| DEC, DECB | √ | √ | √ | √ | ↑ | — |
| DI | — | — | — | — | — | — |
| DIV, DIVB, DIVU, DIVUB | — | — | — | √ | ↑ | — |
| DJNZ, DJNZW | — | — | — | — | — | — |
| DPTS | — | — | — | — | — | — |
| EI | — | — | — | — | — | — |
| EPTS | — | — | — | — | — | — |
| EXT, EXTB | √ | √ | 0 | 0 | — | — |
| IDLPD | | | | | | |
| Legal Key | — | — | — | — | — | — |
| Illegal Key | 0 | 0 | 0 | 0 | 0 | 0 |
| INC | √ | √ | √ | √ | ↑ | 0 |
| INCB | √ | √ | √ | √ | ↑ | — |
| JBC, JBS, JC, JE, JGE, JGT, JH, JLE, JLT, JNC, JNE, JNH, JNST | — | — | — | — | — | — |
| JNV | — | — | — | — | — | — |
| JNVT | — | — | — | — | 0 | — |
| JST, JV | — | — | — | — | — | — |
| JVT | — | — | — | — | 0 | — |

| Mnemonic | PSW Flag Settings | | | | | |
|---|---|---|---|---|---|---|
| | Z | N | C | V | VT | ST |
| LCALL, LD, LDB, LDBSE, LDBZE | — | — | — | — | — | — |
| LJMP | — | — | — | — | — | ? |
| MUL, MULB, MULU, MULUB | — | — | — | — | — | ? |
| NEG, NEGB | √ | √ | √ | √ | ↑ | — |
| NOP | — | — | — | — | — | — |
| NORML | √ | ? | 0 | — | — | — |
| NOT, NOTB | √ | √ | 0 | 0 | — | — |
| OR, ORB | √ | √ | 0 | 0 | — | — |
| POP | — | — | — | — | — | — |
| POPA, POPF | √ | √ | √ | √ | √ | √ |
| PUSH | — | — | — | — | — | — |
| PUSHA, PUSHF | 0 | 0 | 0 | 0 | 0 | 0 |
| RET | — | — | — | — | — | — |
| RST | 0 | 0 | 0 | 0 | 0 | 0 |
| SCALL | — | — | — | — | — | — |
| SETC | — | — | 1 | — | — | — |
| SHL, SHLB, SHLL | √ | ? | √ | √ | ↑ | — |
| SHR | √ | 0 | √ | 0 | — | √ |
| SHRA, SHRAB, SHRAL | √ | √ | √ | 0 | — | √ |
| SHRB, SHRL | √ | 0 | √ | 0 | — | √ |
| SJMP | — | — | — | — | — | — |
| SKIP | — | — | — | — | — | — |
| ST, STB | — | — | — | — | — | — |
| SUB, SUBB | √ | √ | √ | √ | ↑ | — |
| SUBC, SUBCB | ↓ | √ | √ | √ | ↑ | — |
| TIJMP | — | — | — | — | — | — |
| TRAP | — | — | — | — | — | — |
| XCH, XCHB | — | — | — | — | — | — |
| XOR, XORB | √ | √ | 0 | 0 | — | — |

## Table A-6. Effect of PSW Flags or Specified Bits on Conditional Jump Instructions

| Instruction | Jumps to Destination if | Continues if |
|---|---|---|
| DJNZ | decremented byte ≠ 0 | decremented byte = 0 |
| DJNZW | decremented word ≠ 0 | decremented word = 0 |
| JBC | specified register bit = 0 | specified register bit = 1 |
| JBS | specified register bit = 1 | specified register bit = 0 |
| JNC | C = 0 | C = 1 |
| JNH | C = 0 OR Z = 1 | C = 1 AND Z = 0 |
| JC | C = 1 | C = 0 |
| JH | C = 1 AND Z = 0 | C = 0 OR Z = 1 |
| JGE | N = 0 | N = 1 |
| JGT | N = 0 AND Z = 0 | N = 1 OR Z = 1 |
| JLT | N = 1 | N = 0 |
| JLE | N = 1 OR Z = 1 | N = 0 AND Z = 0 |
| JNST | ST = 0 | ST = 1 |
| JST | ST = 1 | ST = 0 |
| JNV | V = 0 | V = 1 |
| JV | V = 1 | V = 0 |
| JNVT | VT = 0 | VT = 1 (clears VT) |
| JVT | VT = 1 (clears VT) | VT = 0 |
| JNE | Z = 0 | Z = 1 |
| JE | Z = 1 | Z = 0 |

Table A-7 lists the instruction opcodes, in hexadecimal order, along with the corresponding instruction mnemonics.

**Table A-7. 8XC196KC/KD Instruction Opcodes**

| Hex Code | Instruction Mnemonic |
|---|---|
| 00 | SKIP |
| 01 | CLR |
| 02 | NOT |
| 03 | NEG |
| 04 | XCH |
| 05 | DEC |
| 06 | EXT |
| 07 | INC |
| 08 | SHR |
| 09 | SHL |
| 0A | SHRA |
| 0B | XCH |
| 0C | SHRL |
| 0D | SHLL |
| 0E | SHRAL |
| 0F | NORML |
| 10 | Reserved |
| 11 | CLRB |
| 12 | NOTB |
| 13 | NEGB |
| 14 | XCHB |
| 15 | DECB |
| 16 | EXTB |
| 17 | INCB |
| 18 | SHRB |
| 19 | SHLB |
| 1A | SHRAB |
| 1B | XCHB |
| 1C–1F | Reserved |
| 20–27 | SJMP |
| 28–2F | SCALL |
| 30–37 | JBC |
| 38–3F | JBS |
| 40 | AND Direct (3 ops) |
| 41 | AND Immediate (3 ops) |
| 42 | AND Indirect (3 ops) |
| 43 | AND Indexed (3 ops) |
| 44 | ADD Direct (3 ops) |
| 45 | ADD Immediate (3 ops) |
| 46 | ADD Indirect (3 ops) |
| 47 | ADD Indexed (3 ops) |
| 48 | SUB Direct (3 ops) |
| 49 | SUB Immediate (3 ops) |
| 4A | SUB Indirect (3 ops) |
| 4B | SUB Indexed (3 ops) |
| 4C | MULU Direct (3 ops) |
| 4D | MULU Immediate (3 ops) |
| 4E | MULU Indirect (3 ops) |
| 4F | MULU Indexed (3 ops) |
| 50 | ANDB Direct (3 ops) |
| 51 | ANDB Immediate (3 ops) |
| 52 | ANDB Indirect (3 ops) |
| 53 | ANDB Indexed (3 ops) |
| 54 | ADDB Direct (3 ops) |
| 55 | ADDB Immediate (3 ops) |
| 56 | ADDB Indirect (3 ops) |
| 57 | ADDB Indexed (3 ops) |
| 58 | SUBB Direct (3 ops) |
| 59 | SUBB Immediate (3 ops) |
| 5A | SUBB Indirect (3 ops) |

## Table A-7. 8XC196KC/KD Instruction Opcodes (Continued)

| Hex Code | Instruction Mnemonic |
|---|---|
| 5B | SUBB Indexed (3 ops) |
| 5C | MULUB Direct (3 ops) |
| 5D | MULUB Immediate (3 ops) |
| 5E | MULUB Indirect (3 ops) |
| 5F | MULUB Indexed (3 ops) |
| 60 | AND Direct (2 ops) |
| 61 | AND Immediate (2 ops) |
| 62 | AND Indirect (2 ops) |
| 63 | AND Indexed (2 ops) |
| 64 | ADD Direct (2 ops) |
| 65 | ADD Immediate (2 ops) |
| 66 | ADD Indirect (2 ops) |
| 67 | ADD Indexed (2 ops) |
| 68 | SUB Direct (2 ops) |
| 69 | SUB Immediate (2 ops) |
| 6A | SUB Indirect (2 ops) |
| 6B | SUB Indexed (2 ops) |
| 6C | MULU Direct (2 ops) |
| 6D | MULU Immediate (2 ops) |
| 6E | MULU Indirect (2 ops) |
| 6F | MULU Indexed (2 ops) |
| 70 | ANDB Direct (2 ops) |
| 71 | ANDB Immediate (2 ops) |
| 72 | ANDB Indirect (2 ops) |
| 73 | ANDB Indexed (2 ops) |
| 74 | ADDB Direct (2 ops) |
| 75 | ADDB Immediate (2 ops) |
| 76 | ADDB Indirect (2 ops) |
| 77 | ADDB Indexed (2 ops) |
| 78 | SUBB Direct (2 ops) |
| 79 | SUBB Immediate (2 ops) |
| 7A | SUBB Indirect (2 ops) |
| 7B | SUBB Indexed (2 ops) |

| Hex Code | Instruction Mnemonic |
|---|---|
| 7C | MULUB Direct (2 ops) |
| 7D | MULUB Immediate (2 ops) |
| 7E | MULUB Indirect (2 ops) |
| 7F | MULUB Indexed (2 ops) |
| 80 | OR Direct |
| 81 | OR Immediate |
| 82 | OR Indirect |
| 83 | OR Indexed |
| 84 | XOR Direct |
| 85 | XOR Immediate |
| 86 | XOR Indirect |
| 87 | XOR Indexed |
| 88 | CMP Direct |
| 89 | CMP Immediate |
| 8A | CMP Indirect |
| 8B | CMP Indexed |
| 8C | DIVU Direct |
| 8D | DIVU Immediate |
| 8E | DIVU Indirect |
| 8F | DIVU Indexed |
| 90 | ORB Direct |
| 91 | ORB Immediate |
| 92 | ORB Indirect |
| 93 | ORB Indexed |
| 94 | XORB Direct |
| 95 | XORB Immediate |
| 96 | XORB Indirect |
| 97 | XORB Indexed |
| 98 | CMPB Direct |
| 99 | CMPB Immediate |
| 9A | CMPB Indirect |
| 9B | CMPB Indexed |
| 9C | DIVUB Direct |

## Table A-7. 8XC196KC/KD Instruction Opcodes (Continued)

| Hex Code | Instruction Mnemonic |
|---|---|
| 9D | DIVUB Immediate |
| 9E | DIVUB Indirect |
| 9F | DIVUB Indexed |
| A0 | LD Direct |
| A1 | LD Immediate |
| A2 | LD Indirect |
| A3 | LD Indexed |
| A4 | ADDC Direct |
| A5 | ADDC Immediate |
| A6 | ADDC Indirect |
| A7 | ADDC Indexed |
| A8 | SUBC Direct |
| A9 | SUBC Immediate |
| AA | SUBC Indirect |
| AB | SUBC Indexed |
| AC | LDBZE Direct |
| AD | LDBZE Immediate |
| AE | LDBZE Indirect |
| AF | LDBZE Indexed |
| B0 | LDB Direct |
| B1 | LDB Immediate |
| B2 | LDB Indirect |
| B3 | LDB Indexed |
| B4 | ADDCB Direct |
| B5 | ADDCB Immediate |
| B6 | ADDCB Indirect |
| B7 | ADDCB Indexed |
| B8 | SUBCB Direct |
| B9 | SUBCB Immediate |
| BA | SUBCB Indirect |
| BB | SUBCB Indexed |
| BC | LDBSE Direct |
| BD | LDBSE Immediate |

| Hex Code | Instruction Mnemonic |
|---|---|
| BE | LDBSE Indirect |
| BF | LDBSE Indexed |
| C0 | ST Direct |
| C1 | BMOV |
| C2 | ST Indirect |
| C3 | ST Indexed |
| C4 | STB Direct |
| C5 | CMPL |
| C6 | STB Indirect |
| C7 | STB Indexed |
| C8 | PUSH Direct |
| C9 | PUSH Immediate |
| CA | PUSH Indirect |
| CB | PUSH Indexed |
| CC | POP Direct |
| CD | BMOVI |
| CE | POP Indirect |
| CF | POP Indexed |
| D0 | JNST |
| D1 | JNH |
| D2 | JGT |
| D3 | JNC |
| D4 | JNVT |
| D5 | JNV |
| D6 | JGE |
| D7 | JNE |
| D8 | JST |
| D9 | JH |
| DA | JLE |
| DB | JC |
| DC | JVT |
| DD | JV |
| DE | JLT |

**Table A-7. 8XC196KC/KD Instruction Opcodes (Continued)**

| Hex Code | Instruction Mnemonic |
|---|---|
| DF | JE |
| E0 | DJNZ |
| E1 | DJNZW |
| E2 | TIJMP |
| E3 | BR (Indirect) |
| E4–E6 | Reserved |
| E7 | LJMP |
| E8–EB | Reserved |
| EC | DPTS |
| ED | EPTS |
| EE | Reserved * |
| EF | LCALL |
| F0 | RET |
| F1 | Reserved |

* Opcode EE is reserved; however, it does not generate an unimplemented opcode interrupt.

| Hex Code | Instruction Mnemonic |
|---|---|
| F2 | PUSHF |
| F3 | POPF |
| F4 | PUSHA |
| F5 | POPA |
| F6 | IDLPD |
| F7 | TRAP |
| F8 | CLRC |
| F9 | SETC |
| FA | DI |
| FB | EI |
| FC | CLRVT |
| FD | NOP |
| FE | *DIV/DIVB/MUL/MULB |
| FF | RST |

* Signed multiplication and division are two-byte instructions. For each signed instruction, the first byte is "FE" and the second is the opcode of the corresponding unsigned instruction. For example, the opcode for MULU (3 operands) direct is "4C," so the opcode for MUL (3 operands) direct is "FE 4C."

Table A-8 is a map of the 8XC196KC/KD opcodes. The first digit of the opcode is listed vertically, and the second digit is listed horizontally. The related instruction mnemonic is shown at the intersection of the two digits. Shading indicates reserved opcodes.

## Table A-8. 8XC196KC/KD Opcode Map (Left Half)

| Op-code | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
|---|---|---|---|---|---|---|---|---|
| 0x | SKIP | CLR | NOT | NEG | XCH<br>di | DEC | EXT | INC |
| 1x | | CLRB | NOTB | NEGB | XCHB<br>di | DECB | EXTB | INCB |
| 2x | SJMP | | | | | | | |
| 3x | JBC | | | | | | | |
| | bit 0 | bit 1 | bit 2 | bit 3 | bit 4 | bit 5 | bit 6 | bit 7 |
| 4x | AND 3op | | | | ADD 3op | | | |
| | di | im | in | ix | di | im | in | ix |
| 5x | ANDB 3op | | | | ADDB 3op | | | |
| | di | im | in | ix | di | im | in | ix |
| 6x | AND 2op | | | | ADD 2op | | | |
| | di | im | in | ix | di | im | in | ix |
| 7x | ANDB 2op | | | | ADDB 2op | | | |
| | di | im | in | ix | di | im | in | ix |
| 8x | OR | | | | XOR | | | |
| | di | im | in | ix | di | im | in | ix |
| 9x | ORB | | | | XORB | | | |
| | di | im | in | ix | di | im | in | ix |
| Ax | LD | | | | ADDC | | | |
| | di | im | in | ix | di | im | in | ix |
| Bx | LDB | | | | ADDCB | | | |
| | di | im | in | ix | di | im | in | ix |
| Cx | ST<br>di | BMOV | ST<br>in | ix | STB<br>di | CMPL | STB<br>in | ix |
| Dx | JNST | JNH | JGT | JNC | JNVT | JNV | JGE | JNE |
| Ex | DJNZ | DJNZW | TIJMP | BR<br>in | | | | LJMP |
| Fx | RET | | PUSHF | POPF | PUSHA | POPA | IDLPD | TRAP |

Shading indicates reserved opcodes.

## Table A-8. 8XC196KC/KD Opcode Map (Right Half)

| Op-code | x8 | x9 | xA | xB | xC | xD | xE | xF |
|---|---|---|---|---|---|---|---|---|
| 0x | SHR | SHL | SHRA | XCH ix | SHRL | SHLL | SHRAL | NORML |
| 1x | SHRB | SHLB | SHRAB | XCHB ix | | | | |
| 2x | SCALL | | | | | | | |
| 3x | JBS | | | | | | | |
| | bit 0 | bit 1 | bit 2 | bit 3 | bit 4 | bit 5 | bit 6 | bit 7 |
| 4x | SUB 3op | | | | MULU 3op * | | | |
| | di | im | in | ix | di | im | in | ix |
| 5x | SUBB 3op | | | | MULUB 3op * | | | |
| | di | im | in | ix | di | im | in | ix |
| 6x | SUB 2op | | | | MULU 2op * | | | |
| | di | im | in | ix | di | im | in | ix |
| 7x | SUBB 2op | | | | MULUB 2op * | | | |
| | di | im | in | ix | di | im | in | ix |
| 8x | CMP | | | | DIVU * | | | |
| | di | im | in | ix | di | im | in | ix |
| 9x | CMPB | | | | DIVUB * | | | |
| | di | im | in | ix | di | im | in | ix |
| Ax | SUBC | | | | LDBZE | | | |
| | di | im | in | ix | di | im | in | ix |
| Bx | SUBCB | | | | LDBSE | | | |
| | di | im | in | ix | di | im | in | ix |
| Cx | PUSH | | | | POP | BMOVI | POP | |
| | di | im | in | ix | di | | in | ix |
| Dx | JST | JH | JLE | JC | JVT | JV | JLT | JE |
| Ex | | | | | DPTS | EPTS | ** | LCALL |
| Fx | CLRC | SETC | DI | EI | CLRVT | NOP | signed multiply/ divide* | RST |

* Signed multiplication and division are two-byte instructions. The first byte is "FE" and the second is the opcode of the corresponding unsigned instruction.
** Opcode EE is reserved, but it does not generate an unimplemented opcode interrupt.

Table A-9 lists instructions along with their lengths and opcodes for each applicable addressing mode. There are two columns for each addressing mode. The first column lists the instruction length, and the second column lists the hexadecimal opcodes. For indexed instructions, the first column lists instruction lengths as *S/L*, where *S* is the short-indexed instruction length and *L* is the long-indexed instruction length. A dash in any column (—) indicates "not applicable."

**Table A-9. Instruction Lengths and Hexadecimal Opcodes**

| Arithmetic (Group I) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect** (1) | | **Indexed** S/L (1) | |
| ADD (2 ops) | 3 | 64 | 4 | 65 | 3 | 66 | 4/5 | 67 |
| ADD (3 ops) | 4 | 44 | 5 | 45 | 4 | 46 | 5/6 | 47 |
| ADDB (2 ops) | 3 | 74 | 3 | 75 | 3 | 76 | 4/5 | 77 |
| ADDB (3 ops) | 4 | 54 | 4 | 55 | 4 | 56 | 5/6 | 57 |
| ADDC | 3 | A4 | 4 | A5 | 3 | A6 | 4/5 | A7 |
| ADDCB | 3 | B4 | 3 | B5 | 3 | B6 | 4/5 | B7 |
| CMP | 3 | 88 | 4 | 89 | 3 | 8A | 4/5 | 8B |
| CMPB | 3 | 98 | 3 | 99 | 3 | 9A | 4/5 | 9B |
| SUB (2 ops) | 3 | 68 | 4 | 69 | 3 | 6A | 4/5 | 6B |
| SUB (3 ops) | 4 | 48 | 5 | 49 | 4 | 4A | 5/6 | 4B |
| SUBB (2 ops) | 3 | 78 | 3 | 79 | 3 | 7A | 4/5 | 7B |
| SUBB (3 ops) | 4 | 58 | 4 | 59 | 4 | 5A | 5/6 | 5B |
| SUBC | 3 | A8 | 4 | A9 | 3 | AA | 4/5 | AB |
| SUBCB | 3 | B8 | 3 | B9 | 3 | BA | 4/5 | BB |

## Table A-9. Instruction Lengths and Hexadecimal Opcodes (Continued)

| Arithmetic (Group II) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect** (1) | | **Indexed S/L** (1) | |
| DIV | 4 | FE 8C | 5 | FE 8D | 4 | FE 8E | 5/6 | FE 8F |
| DIVB | 4 | FE 9C | 4 | FE 9D | 4 | FE 9E | 5/6 | FE 9F |
| DIVU | 3 | 8C | 4 | 8D | 3 | 8E | 4/5 | 8F |
| DIVUB | 3 | 9C | 3 | 9D | 3 | 9E | 4/5 | 9F |
| MUL (2 ops) | 4 | FE 6C | 5 | FE 6D | 4 | F3 6E | 5/6 | FE 6F |
| MUL (3 ops) | 5 | FE 4C | 6 | FE 4D | 5 | FE 4E | 6/7 | FE 4F |
| MULB (2 ops) | 4 | FE 7C | 4 | FE 7D | 4 | FE 7E | 5/6 | FE 7F |
| MULB (3 ops) | 5 | FE 5C | 5 | FE 5D | 5 | FE 5E | 6/7 | FE 5F |
| MULU (2 ops) | 3 | 6C | 4 | 6D | 3 | 6E | 4/5 | 6F |
| MULU (3 ops) | 4 | 4C | 5 | 4D | 4 | 4E | 5/6 | 4F |
| MULUB (2 ops) | 3 | 7C | 3 | 7D | 3 | 7E | 4/5 | 7F |
| MULUB (3 ops) | 4 | 5C | 4 | 5D | 4 | 5E | 5/6 | 5F |
| **Logical** | | | | | | | | |
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect** (1) | | **Indexed S/L** (1) | |
| AND (2 ops) | 3 | 60 | 4 | 61 | 3 | 62 | 4/5 | 63 |
| AND (3 ops) | 4 | 40 | 5 | 41 | 4 | 42 | 5/6 | 43 |
| ANDB (2 ops) | 3 | 70 | 3 | 71 | 3 | 72 | 4/4 | 73 |
| ANDB (3 ops) | 4 | 50 | 4 | 51 | 4 | 52 | 5/5 | 53 |
| OR | 3 | 80 | 4 | 81 | 3 | 82 | 4/5 | 83 |
| ORB | 3 | 90 | 3 | 91 | 3 | 92 | 4/5 | 93 |
| XOR | 3 | 84 | 4 | 85 | 3 | 86 | 4/5 | 87 |
| XORB | 3 | 94 | 3 | 95 | 3 | 96 | 4/5 | 97 |

## Table A-9. Instruction Lengths and Hexadecimal Opcodes (Continued)

| Stack | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect** (1) | | **Indexed S/L** (1) | |
| POP | 2 | CC | — | — | 2 | CE | 3/4 | CF |
| POPA | — | — | — | — | 1 | F5 | — | — |
| POPF | — | — | — | — | 1 | F3 | — | — |
| PUSH | 2 | C8 | 3 | C9 | 2 | CA | 3/4 | CB |
| PUSHA | — | — | — | — | 1 | F4 | — | — |
| PUSHF | — | — | — | — | 1 | F2 | — | — |
| **Data** | | | | | | | | |
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect** (1) | | **Indexed S/L** (1) | |
| LD | 3 | A0 | 4 | A1 | 3 | A2 | 4/5 | A3 |
| LDB | 3 | B0 | 3 | B1 | 3 | B2 | 4/5 | B3 |
| LDBSE | 3 | BC | 3 | BD | 3 | BE | 4/5 | BF |
| LDBZE | 3 | AC | 3 | AD | 3 | AE | 4/5 | AF |
| ST | 3 | C0 | — | — | 3 | C2 | 4/5 | C3 |
| STB | 3 | C4 | — | — | 3 | C6 | 4/5 | C7 |
| XCH | 3 | 04 | — | — | — | — | 4/5 | 0B |
| XCHB | 3 | 14 | — | — | — | — | 4/5 | 1B |
| **Jump** | | | | | | | | |
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect** (1) | | **Indexed S/L** (1) | |
| BR (Indirect) | — | — | — | — | 2 | E3 | 2 | E3 |
| LJMP | — | — | — | — | — | — | —/2 | E7 (2) |
| SJMP | — | — | — | — | — | — | 2/— | 20–27 (2) |
| TIJMP | 4 | E2 | 4 | E2 | — | — | —/4 | E2 |

## Table A-9. Instruction Lengths and Hexadecimal Opcodes (Continued)

| Call | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect** (1) | | **Indexed** S/L (1) | |
| LCALL | — | — | — | — | — | — | —/3 | FF (2) |
| SCALL | — | — | — | — | — | — | 2/— | 28–2F (2) |
| RET | — | — | — | — | 1 | F0 | — | — |
| TRAP | 1 | F7 | — | — | — | — | — | — |
| **Conditional Jump** | | | | | | | | |
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect** (1) | | **Indexed** S/L (1) | |
| DJNZ | — | — | — | — | — | — | 3/— | E0 |
| DJNZW | — | — | — | — | — | — | 3/— | E1 |
| JBC | — | — | — | — | — | — | 3/— | 30–37 |
| JBS | — | — | — | — | — | — | 3/— | 38–3F |
| JC | — | — | — | — | — | — | 1/— | DB |
| JE | — | — | — | — | — | — | 1/— | DF |
| JGE | — | — | — | — | — | — | 1/— | D6 |
| JGT | — | — | — | — | — | — | 1/— | D2 |
| JH | — | — | — | — | — | — | 1/— | D9 |
| JLE | — | — | — | — | — | — | 1/— | DA |
| JLT | — | — | — | — | — | — | 1/— | DE |
| JNC | — | — | — | — | — | — | 1/— | D3 |
| JNE | — | — | — | — | — | — | 1/— | D7 |
| JNH | — | — | — | — | — | — | 1/— | D1 |
| JNST | — | — | — | — | — | — | 1/— | D0 |
| JNV | — | — | — | — | — | — | 1/— | D5 |
| JNVT | — | — | — | — | — | — | 1/— | D4 |
| JST | — | — | — | — | — | — | 1/— | D8 |
| JV | — | — | — | — | — | — | 1/— | DD |
| JVT | — | — | — | — | — | — | 1/— | DC |

**Table A-9. Instruction Lengths and Hexadecimal Opcodes (Continued)**

| Shift | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect (1)** | | **Indexed S/L (1)** | |
| NORML | 3 | 0F | — | — | — | — | — | — |
| SHL | 3 | 09 | — | — | — | — | — | — |
| SHLB | 3 | 19 | — | — | — | — | — | — |
| SHLL | 3 | 0D | — | — | — | — | — | — |
| SHR | 3 | 08 | — | — | — | — | — | — |
| SHRA | 3 | 0A | — | — | — | — | — | — |
| SHRAB | 3 | 1A | — | — | — | — | — | — |
| SHRAL | 3 | 0E | — | — | — | — | — | — |
| SHRB | 3 | 18 | — | — | — | — | — | — |
| SHRL | 3 | 0C | — | — | — | — | — | — |
| **Block** | | | | | | | | |
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect (1)** | | **Indexed S/L (1)** | |
| BMOV | — | — | — | — | — | — | —/3 | C1 |
| BMOVI | — | — | — | — | — | — | —/3 | CD |
| **Special** | | | | | | | | |
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect (1)** | | **Indexed S/L (1)** | |
| CLRC | 1 | F8 | — | — | — | — | — | — |
| CLRVT | 1 | FC | — | — | — | — | — | — |
| DI | 1 | FA | — | — | — | — | — | — |
| EI | 1 | FB | — | — | — | — | — | — |
| IDLPD | — | — | 1 | F6 | — | — | — | — |
| NOP | 1 | FD | — | — | — | — | — | — |
| RST | 1 | FE | — | — | — | — | — | — |
| SETC | 1 | F9 | — | — | — | — | — | — |
| SKIP | 2 | 00 | — | — | — | — | — | — |
| **PTS** | | | | | | | | |
| **Mnemonic** | **Direct** | | **Immediate** | | **Indirect (1)** | | **Indexed S/L (1)** | |
| DPTS | 1 | EC | — | — | — | — | — | — |
| EPTS | 1 | ED | — | — | — | — | — | — |

## Table A-9. Instruction Lengths and Hexadecimal Opcodes (Continued)

| Single | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Mnemonic | Direct | | Immediate | | Indirect (1) | | Indexed S/L (1) | |
| CLR | — | 01 | — | — | — | — | — | — |
| CLRB | — | 11 | — | — | — | — | — | — |
| CMPL | — | C5 | — | — | — | — | — | — |
| DEC | — | 05 | — | — | — | — | — | — |
| DECB | — | 15 | — | — | — | — | — | — |
| EXT | — | 06 | — | — | — | — | — | — |
| EXTB | — | 16 | — | — | — | — | — | — |
| INC | — | 07 | — | — | — | — | — | — |
| INCB | — | 17 | — | — | — | — | — | — |
| NEG | — | 03 | — | — | — | — | — | — |
| NEGB | — | 13 | — | — | — | — | — | — |
| NOT | — | 02 | — | — | — | — | — | — |
| NOTB | — | 12 | — | — | — | — | — | — |

**NOTES:**

1. Indirect normal and indirect auto-increment share the same opcodes, as do short- and long-indexed modes. To use indirect normal or short-indexed mode, the second byte of the instruction must be even. To use indirect auto-increment or long-indexed mode, the second byte of the instruction must be odd.
2. For these instructions (SCALL, SJMP, LCALL, LJMP), the 3 least-significant bits of the opcode are concatenated with the 8 bits to form an 11-bit, 2's complement offset.

Table A-10 lists instructions alphabetically within groups, along with their execution times, expressed in state times. The table denotes execution times for indirect and indexed addressing modes as *R*/*M*, where *R* is the execution time using SFRs and internal RAM (0H–1FFH) and *M* is the execution time using the memory controller (200H–0FFFFH).

**Table A-10. 8XC196KC/KD Instruction Execution Times (in State Times)**

| Arithmetic (Group I) | | | | | | |
|---|---|---|---|---|---|---|
| | | | Indirect | | Indexed | |
| Mnemonic | Direct | Immed. | Normal | Auto-Inc. | Short | Long |
| ADD (2 ops) | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| ADD (3 ops) | 5 | 6 | 7/10 | 8/11 | 7/10 | 8/11 |
| ADDB (2 ops) | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| ADDB (3 ops) | 5 | 6 | 7/10 | 8/11 | 7/10 | 8/11 |
| ADDC | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| ADDCB | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| CMP | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| CMPB | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| SUB (2 ops) | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| SUB (3 ops) | 5 | 6 | 7/10 | 8/11 | 7/10 | 8/11 |
| SUBB (2 ops) | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| SUBB (3 ops) | 5 | 6 | 7/10 | 8/11 | 7/10 | 8/11 |
| SUBC | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| SUBCB | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |

**Table A-10. 8XC196KC/KD Instruction Execution Times (Continued)**

| Arithmetic (Group II) | | | | | | |
|---|---|---|---|---|---|---|
| | | | Indirect | | Indexed | |
| Mnemonic | Direct | Immed. | Normal | Auto-Inc. | Short | Long |
| DIV | 26 | 27 | 28/31 | 29/32 | 29/32 | 30/33 |
| DIVB | 18 | 18 | 20/23 | 21/24 | 21/24 | 22/25 |
| DIVU | 24 | 25 | 26/29 | 27/30 | 27/30 | 28/31 |
| DIVUB | 16 | 16 | 18/21 | 19/22 | 19/22 | 20/23 |
| MUL (2 ops) | 16 | 17 | 18/21 | 19/22 | 19/22 | 20/23 |
| MUL (3 ops) | 16 | 17 | 18/21 | 19/22 | 19/22 | 20/23 |
| MULB (2 ops) | 12 | 12 | 14/17 | 15/18 | 15/18 | 16/19 |
| MULB (3 ops) | 12 | 12 | 14/17 | 15/18 | 15/18 | 16/19 |
| MULU (2 ops) | 14 | 15 | 16/19 | 17/19 | 17/20 | 18/21 |
| MULU (3 ops) | 14 | 15 | 16/19 | 17/19 | 17/20 | 18/21 |
| MULUB (2 ops) | 10 | 10 | 12/15 | 13/15 | 12/16 | 14/17 |
| MULUB (3 ops) | 10 | 10 | 15/15 | 12/16 | 12/16 | 14/17 |
| **Logical** | | | | | | |
| | | | Indirect | | Indexed | |
| Mnemonic | Direct | Immed. | Normal | Auto-Inc. | Short | Long |
| AND (2 ops) | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| AND (3 ops) | 5 | 6 | 7/10 | 8/11 | 7/10 | 8/11 |
| ANDB (2 ops) | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| ANDB (3 ops) | 5 | 6 | 7/10 | 8/11 | 7/10 | 8/11 |
| OR | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| ORB | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| XOR | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |
| XORB | 4 | 5 | 6/8 | 7/9 | 6/8 | 7/9 |

**Table A-10. 8XC196KC/KD Instruction Execution Times (Continued)**

| Stack (Internal Stack) | | | | | | |
|---|---|---|---|---|---|---|
| | | | Indirect | | Indexed | |
| Mnemonic | Direct | Immed. | Normal | Auto-Inc. | Short | Long |
| POP | 8 | — | 10/12 | 11/13 | 11/13 | 12/14 |
| POPA | 12 | — | — | — | — | — |
| POPF | 7 | — | — | — | — | — |
| PUSH | 6 | 7 | 9/12 | 10/13 | 10/13 | 11/14 |
| PUSHA | 12 | — | — | — | — | — |
| PUSHF | 6 | — | — | — | — | — |
| **Stack (External Stack)** | | | | | | |
| | | | Indirect | | Indexed | |
| Mnemonic | Direct | Immed. | Normal | Auto-Inc. | Short | Long |
| POP | 11 | — | 13/15 | 14/16 | 14/16 | 15/17 |
| POPA | 18 | — | — | — | — | — |
| POPF | 10 | — | — | — | — | — |
| PUSH | 8 | 9 | 11/14 | 12/15 | 12/15 | 13/16 |
| PUSHA | 18 | — | — | — | — | — |
| PUSHF | 8 | — | — | — | — | — |
| **Data** | | | | | | |
| | | | Indirect | | Indexed | |
| Mnemonic | Direct | Immed. | Normal | Auto-Inc. | Short | Long |
| LD | 4 | 5 | 5/8 | 6/8 | 6/9 | 7/10 |
| LDB | 4 | 5 | 5/8 | 6/8 | 6/9 | 7/10 |
| LDBSE | 4 | 4 | 5/8 | 6/8 | 6/9 | 7/10 |
| LDBZE | 4 | 4 | 5/8 | 6/8 | 6/9 | 7/10 |
| ST | 4 | — | 5/8 | 6/9 | 6/9 | 7/10 |
| STB | 4 | — | 5/8 | 6/9 | 6/9 | 7/10 |
| XCH | 5 | — | — | — | 8/13 | 9/14 |
| XCHB | 5 | — | — | — | 8/13 | 9/14 |

**Table A-10. 8XC196KC/KD Instruction Execution Times (Continued)**

| Jump | | | | | | |
|---|---|---|---|---|---|---|
| | | | Indirect | | Indexed | |
| Mnemonic | Direct | Immed. | Normal | Auto-Inc. | Short | Long |
| BR (Indirect) | — | — | 7 | 7 | 7 | 7 |
| LJMP | — | — | — | — | — | 7 |
| SJMP | — | — | — | — | 7 | — |
| TIJMP<br>internal/internal<br>external/internal<br>external/external | <br>15<br>18<br>21 | <br>15<br>18<br>21 | — | — | — | — |
| **Call (Internal Stack)** | | | | | | |
| | | | Indirect | | Indexed | |
| Mnemonic | Direct | Immed. | Normal | Auto-Inc. | Short | Long |
| LCALL | — | — | — | — | — | 11 |
| RET | — | — | 11 | — | — | — |
| SCALL | — | — | — | — | 11 | — |
| TRAP | 16 | — | — | — | — | — |
| **Call (External Stack)** | | | | | | |
| | | | Indirect | | Indexed | |
| Mnemonic | Direct | Immed. | Normal | Auto-Inc. | Short | Long |
| LCALL | — | — | — | — | — | 13 |
| RET | — | — | 14 | — | — | — |
| SCALL | — | — | — | — | 13 | — |
| TRAP | 18 | — | — | — | — | — |

**Table A-10. 8XC196KC/KD Instruction Execution Times (Continued)**

| Conditional Jump | |
|---|---|
| **Mnemonic** | **Short-Indexed** |
| DJNZ | 5 (jump not taken), 9 (jump taken) |
| DJNZW<br>80C196KC only<br>8XC196KC/KD | <br>7 (jump not taken), 11 (jump taken)<br>6 (jump not taken), 10 (jump taken) |
| JBC | 5 (jump not taken), 9 (jump taken) |
| JBS | 5 (jump not taken), 9 (jump taken) |
| JC | 4 (jump not taken), 8 (jump taken) |
| JE | 4 (jump not taken), 8 (jump taken) |
| JGE | 4 (jump not taken), 8 (jump taken) |
| JGT | 4 (jump not taken), 8 (jump taken) |
| JH | 4 (jump not taken), 8 (jump taken) |
| JLE | 4 (jump not taken), 8 (jump taken) |
| JLT | 4 (jump not taken), 8 (jump taken) |
| JNC | 4 (jump not taken), 8 (jump taken) |
| JNE | 4 (jump not taken), 8 (jump taken) |
| JNH | 4 (jump not taken), 8 (jump taken) |
| JNST | 4 (jump not taken), 8 (jump taken) |
| JNV | 4 (jump not taken), 8 (jump taken) |
| JNVT | 4 (jump not taken), 8 (jump taken) |
| JST | 4 (jump not taken), 8 (jump taken) |
| JV | 4 (jump not taken), 8 (jump taken) |
| JVT | 4 (jump not taken), 8 (jump taken) |

## Table A-10. 8XC196KC/KD Instruction Execution Times (Continued)

| Shift | |
|---|---|
| **Mnemonic** | **Direct** |
| NORML | 8 + 1 per shift (9 for 0 shift) |
| SHL | 6 + 1 per shift (7 for 0 shift) |
| SHLB | 6 + 1 per shift (7 for 0 shift) |
| SHR | 6 + 1 per shift (7 for 0 shift) |
| SHRA | 6 + 1 per shift (7 for 0 shift) |
| SHRAB | 6 + 1 per shift (7 for 0 shift) |
| SHRAL | 7 + 1 per shift (8 for 0 shift) |
| SHRB | 6 + 1 per shift (7 for 0 shift) |
| SHRL | 7 + 1 per shift (8 for 0 shift) |

| Block | |
|---|---|
| **Mnemonic** | **Long-Indexed** |
| BMOV | internal/internal 6 + 8 per word<br>external/internal 6 + 11 per word<br>external/external 6 + 14 per word |
| BMOVI | internal/internal 7 + 8 per word + 14 per interrupt<br>external/internal 7 + 11 per word + 14 per interrupt<br>external/external 7 + 14 per word + 14 per interrupt |

| Special | | | | | | |
|---|---|---|---|---|---|---|
| | | | **Indirect** | | **Indexed** | |
| **Mnemonic** | **Direct** | **Immed.** | **Normal** | **Auto-Inc.** | **Short** | **Long** |
| CLRC | 2 | — | — | — | — | — |
| CLRVT | 2 | — | — | — | — | — |
| DI | 2 | — | — | — | — | — |
| EI | 2 | — | — | — | — | — |
| IDLPD<br>Valid key<br>Invalid key | <br>—<br>— | <br>8<br>25 | <br>—<br>— | <br>—<br>— | <br>—<br>— | <br>—<br>— |
| NOP | 2 | — | — | — | — | — |
| RST (including fetch of configuration byte) | 20 | — | — | — | — | — |
| SETC | 2 | — | — | — | — | — |
| SKIP | 3 | — | — | — | — | — |

**Table A-10. 8XC196KC/KD Instruction Execution Times (Continued)**

| PTS | | | | | | |
|---|---|---|---|---|---|---|
| | | | Indirect | | Indexed | |
| Mnemonic | Direct | Immed. | Normal | Auto-Inc. | Short | Long |
| DPTS | 2 | — | — | — | — | — |
| EPTS | 2 | — | — | — | — | — |
| **Single** | | | | | | |
| | | | Indirect | | Indexed | |
| Mnemonic | Direct | Immed. | Normal | Auto-Inc. | Short | Long |
| CLR | 3 | — | — | — | — | — |
| CLRB | 3 | — | — | — | — | — |
| CMPL | 7 | — | — | — | — | — |
| DEC | 3 | — | — | — | — | — |
| DECB | 3 | — | — | — | — | — |
| EXT | 4 | — | — | — | — | — |
| EXTB | 4 | — | — | — | — | — |
| INC | 3 | — | — | — | — | — |
| INCB | 3 | — | — | — | — | — |
| NEG | 3 | — | — | — | — | — |
| NEGB | 3 | — | — | — | — | — |
| NOT | 3 | — | — | — | — | — |
| NOTB | 3 | — | — | — | — | — |

## Table A-11. 8XC196KC/KD PTS Cycle Execution Times

| PTS Cycles | |
|---|---|
| **PTS Mode** | **Execution Time (in State Times)** |
| Single Transfer mode | |
| internal/internal | 18 |
| external/internal | 21 |
| external/external | 24 |
| Block Transfer mode | |
| internal/internal | 13 + 7 per transfer (1 minimum) |
| external/internal | 16 + 7 per transfer (1 minimum) |
| external/external | 19 + 7 per transfer (1 minimum) |
| A/D Scan mode | |
| SFRs/Internal RAM | 21 |
| Memory Controller | 25 |
| HSI mode | |
| SFRs/Internal RAM | 12 + 10 per transfer (1 minimum) |
| Memory Controller | 16 + 10 per transfer (1 minimum) |
| HSO mode | |
| SFRs/Internal RAM | 11 + 10 per transfer (1 minimum) |
| Memory Controller | 15 + 10 per transfer (1 minimum) |