### Testing

### Pipeline occurs in the correct order with E0 (execute) implemented

**Purpose:** Checks for successful order of pipeline with E0 implemented.

**Configuration:** When a program is loaded into the emulator, it uses #ifndef DEBUG print statements to display the stages occurring, as well as what task they performed, and the clock tick they occur on.

**Expected Results:** F0 (Fetch 0) and D0 (Decode 0) occur on even clock ticks. F1 (Fetch 1) and E0 (Execute 0) occur on odd clock ticks.

**Actual Results:**

A screenshot of a computer program

Description automatically generated

The pipeline stages occur in their expected order.

**Pass/Fail:** PASS

### MOVL, MOVLZ, MOVLS, and MOVH instructions work as expected

**Purpose:** Checks for successful execution of MOVL, MOVLZ, MOVLS, and MOVH instructions.

**Configuration:** The following program is loaded into the emulator. It performs MOVL, MOVLS, MOVLZ, and MOVH operations.

A screen shot of a computer

Description automatically generated

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1002, MOVL is performed:   * DST = 0xEE00 * DST.lowbyte <- 0x99   This will produce:   * DST = 0xFF99 |  |
| At address 1004, MOVH is performed:   * DST = 0xEE99 * DST.highbyte <- 0xAA   This will produce:   * DST = 0xAA99 |  |
| At address 1006, MOVLZ is performed:   * DST = 0xAA99 * DST.highbyte <- 0x00 * DST.lowbyte <- 0x88   This will produce:   * DST = 0xFF88 |  |
| At memory address 1008, MOVLS is performed:   * DST = 0xFF88 * DST.highbyte <- 0xFF * DST.lowbyte <- 0x77   This will produce:   * DST = 0xFF77 |  |

The MOVL, MOVLZ, MOVLS, and MOVH operations perform as expected. They do not set or clear any flags.

**Pass/Fail:** PASS

### ADD and ADDC instructions work as expected

**Purpose:** Checks for successful execution of ADD and ADDC instructions.

**Configuration:** The following program is loaded into the emulator. ADD and ADDC are executed with byte and word, register and constant sources, and related flags each set in at least one operation.

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1004, ADD is performed using values of size word and a register source:   * DST <- 0x9000 + 0x9000   This will produce:   * DST = 0x2000 * SET overflow flag * CLEAR negative flag * CLEAR zero flag * SET carry flag |  |
| At address 1006, ADDC is performed using values of size word with a constant value and a constant source:   * DST <- 0x2000 + $0 + CARRY   This will produce:   * DST = 0x2001 * CLEAR overflow flag * CLEAR negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At memory address 100A, ADD is performed using values of size byte and a register source:   * DST.lowbyte <- 0x01 + 0xFF   This will produce:   * DST.lowbyte = 0x00 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * SET carry flag |  |
| At memory address 100E, ADD is performed using values of size byte and a register source:   * DST.lowbyte <- 0x00 + 0xFF   This will produce:   * DST.lowbyte = 0xFF * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |

The ADD and ADDC instructions perform as expected. They produce the correct results when adding word or byte, and when there is a carry for ADDC. They also set the flags as expected.

**Pass/Fail:** PASS

### SUB and SUBC instructions work as expected

**Purpose:** Checks for successful execution of SUB and SUBC

**Configuration:** The following program is loaded into the emulator. SUB and SUBC are executed with byte and word, register and constant sources, and related flags each set in at least one operation.

**A screenshot of a computer program

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1004, SUB is performed using values of size word and register source:   * DST <- 0x8000 - 0x8000   This will produce:   * DST = 0x0000 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * SET carry flag |  |
| At address 1006, SUBC is performed using values of size word and a constant source:   * DST <- 0x2000 - $0 + CARRY   This will produce:   * RESULT = 0x2001 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * SET carry flag |  |
| At address 100C, SUB is performed using values of size byte and a register source:   * DST.lowbyte <- 0x30 – 0x80   This will produce:   * DST.lowbyte = 0x00B0 * SET overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |

The SUB and SUBC instructions perform as expected. They produce the correct results when subtracting word or byte, and when there is a carry for SUBC. They also set the flags as expected.

**Pass/Fail:** PASS

### DADD instruction works as expected

**Purpose:** Checks for successful execution of DADD instructions

**Configuration:** The following program is loaded into the emulator. DADD is executed with byte and word, register and constant sources, and related flags each set in at least one operation.

**A screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1008, DADD is performed using values of size word with a register source:   * DST <- 1234 + 8766   This will produce:   * DST = 0000 * CLEAR overflow flag * CLEAR negative flag * CLEAR zero flag * SET carry flag |  |
| At address 100A, DADD is performed using values of size word with a constant source:   * DST.lowbyte <- 34 + $4   This will produce:   * DST = 0x1238 * CLEAR overflow flag * CLEAR negative flag * CLEAR zero flag * CLEAR carry flag |  |

The DADD instruction works as expected. It can perform binary coded decimal addition with both sources and words, as well as with register and constant sources. It also sets the flags as expected.

**Pass/Fail:** PASS

### CMP instruction works as expected

**Purpose:** Checks for successful execution of CMP

**Configuration:** The following program is loaded into the emulator. CMP is executed with byte and word, register and constant sources, and related flags each set in at least one operation. Tests are conducted for when DST and SRC have a difference of 0, more than 0, and less than 0.

**A screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1004, CMP is performed using values of size byte and register source:   * 0x08 - 0x08   This will produce:   * CLEAR overflow flag * CLEAR negative flag * SET zero flag * SET carry flag |  |
| At address 100C, CMP is performed using values of size word and a register source:   * 0x0708 – 0x0800   This will produce:   * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At address 10010, CMP is performed using values of size word and a constant source:   * 0x0009 - $1   This will produce:   * CLEAR overflow flag * CLEAR negative flag * CLEAR zero flag * SET carry flag |  |

The CMP instructions perform as expected. They also set the flags as expected.

**Pass/Fail:** PASS

### XOR instruction works as expected

**Purpose:** Checks for successful execution of XOR instruction.

**Configuration:** The following program is loaded into the emulator. XOR is executed with byte and word, register and constant sources, and related flags each set in at least one operation.

**A screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1006, XOR is performed using values of size byte with a register source:   * DST.lowbyte <- 0x08 ^ 0x08   This will produce:   * DST.lowbyte = 0x00 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * CLEAR carry flag |  |
| At address 100A, XOR is performed using values of size word with a constant source:   * DST <- 0xFF00 ^ $0   This will produce:   * DST = 0xFF00 * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |

The XOR instruction works as expected. It can execute with both sources and words, as well as with register and constant sources. It also sets the flags as expected.

**Pass/Fail:** PASS

### AND instruction works as expected

**Purpose:** Checks for successful execution of AND instruction.

**Configuration:** The following program is loaded into the emulator. AND is executed with byte and word, register and constant sources, and related flags each set in at least one operation.

**A screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1006, AND is performed using values of size byte with a register source:   * DST <- 0x008 & 0xFFFF   This will produce:   * DST = 0x0008 * CLEAR overflow flag * CLEAR negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At address 100A, AND is performed using values of size byte with a register source:   * DST.lowbyte <- 0xF0 & 0xFF   This will produce:   * DST.lowbyte = 0xF0 * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At address 100A, AND is performed using values of size word with a constant source:   * DST.lowbyte <- 0xFFFF & $0   This will produce:   * RESULT = 0x0000 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * CLEAR carry flag |  |

The AND instruction works as expected. It can execute with both sources and words, as well as with register and constant sources. It also sets the flags as expected.

**Pass/Fail:** PASS

### OR instruction works as expected

**Purpose:** Checks for successful execution of OR instruction.

**Configuration:** The following program is loaded into the emulator. OR is executed with byte and word, register and constant sources, and related flags each set in at least one operation.

**A screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1002, OR is performed using values of size word with a register source:   * DST <- 0x00FF | 0xFF00   This will produce:   * DST = 0xFFFF * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At address 1006, OR is performed using values of size byte with a constant source:   * DST.lowbyte <- 0x00 | $4   This will produce:   * DST.lowbyte = 0x04 * CLEAR overflow flag * CLEAR negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At address 1008, OR is performed using values of size byte with a constant source:   * DST.lowbyte <- 0x00 | $0   This will produce:   * DST.lowbyte = 0x00 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * CLEAR carry flag |  |

The OR instruction works as expected. It can execute with both sources and words, as well as with register and constant sources. It also sets the flags as expected.

**Pass/Fail:** PASS

### BIT instruction works as expected

**Purpose:** Checks for successful execution of BIT instruction.

**Configuration:** The following program is loaded into the emulator. BIT is executed with byte and word, register and constant sources, and related flags each set in at least one operation.

**A screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1004, BIT is performed using values of size word with a constant source:   * DST <- 0xFFFF & (1 << $8)   This will produce:   * DST = 0x0100 * CLEAR overflow flag * CLEAR negative flag * CLEAR zero flag * CLEAR carry flag | A screenshot of a computer  Description automatically generated |
| At address 100A, BIT is performed using values of size byte with a register source:   * DST.lowbyte <- 0x00 & (1 << 0x02)   This will produce:   * DST = 0x00 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * CLEAR carry flag | A screen shot of a computer  Description automatically generated |

The BIT instruction works as expected. It can execute with both sources and words, as well as with register and constant sources. It also sets the flags as expected.

**Pass/Fail:** PASS

### BIC instructions work as expected

**Purpose:** Checks for successful execution of BIC instruction.

**Configuration:** The following program is loaded into the emulator. BIC is executed with byte and word, register and constant sources, and related flags each set in at least one operation.

**A screenshot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1004, BIC is performed using values of size word with a constant source:   * DST <- 0xFF0F & ~(1 << $8)   This will produce:   * DST = 0xFE0F * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At address 1006, BIC is performed using values of size byte with a constant source:   * DST.lowbyte <- 0x0F| (1 << $2)   This will produce:   * DST = 0x0B * CLEAR overflow flag * CLEAR negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At address 1006, BIC is performed using values of size byte with a register source:   * DST.lowbyte <- 0x02 | (1 << 0x01)   This will produce:   * DST = 0x00 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * CLEAR carry flag |  |

The BIC instruction works as expected. It can execute with both sources and words, as well as with register and constant sources. It also sets the flags as expected.

**Pass/Fail:** PASS

### BIS instructions work as expected

**Purpose:** Checks for successful execution of BIS instruction.

**Configuration:** The following program is loaded into the emulator. BIS is executed with byte and word, register and constant sources, and related flags each set in at least one operation.

**A screenshot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1002, BIS is performed using values of size byte with a register source:   * DST.lowbyte <- 0x00 | (1 << 0xFF)   This will produce:   * DST.lowbyte = 0x00 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * CLEAR carry flag |  |
| At address 1006, BIS is performed using values of size word with a constant source:   * DST <- 0x0008 | (1 << $8)   This will produce:   * DST = 0x0108 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * CLEAR carry flag |  |
| At address 1006, BIS is performed using values of size byte with a register source:   * DST.lowbyte <- 0x0000 | (1 << 0x07)   This will produce:   * DST = 0x80 * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |

The BIS instruction works as expected. It can execute with both sources and words, as well as with register and constant sources. It also sets the flags as expected and does not change the destination value when source or constant value is larger than max number of bits.

**Pass/Fail:** PASS

### BIS instruction works as expected

**Purpose:** Checks for successful execution of BIS instruction.

**Configuration:** The following program is loaded into the emulator. BIS is executed with byte and word, register and constant sources, and related flags each set in at least one operation.

**A screenshot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1002, BIS is performed using values of size byte with a register source:   * DST.lowbyte <- 0x00 | (1 << 0xFF)   This will produce:   * DST.lowbyte = 0x00 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * CLEAR carry flag | A screenshot of a computer  Description automatically generated |
| At address 1006, BIS is performed using values of size word with a constant source:   * DST <- 0x0008 | (1 << $8)   This will produce:   * DST = 0x0108 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * CLEAR carry flag | A computer screen shot of a black screen  Description automatically generated |
| At address 1008, OR is performed using values of size byte with a register source:   * DST.lowbyte <- 0x0000 | (1 << 0x07)   This will produce:   * DST = 0x80 * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag | A computer screen shot of numbers and letters  Description automatically generated |

The BIT instruction works as expected. It can execute with both sources and words, as well as with register and constant sources. It also sets the flags as expected.

**Pass/Fail:** PASS

### MOV instructions work as expected

**Purpose:** Checks for successful execution of BIS instruction.

**Configuration:** The following program is loaded into the emulator. BIS is executed with byte and word, register and constant sources, and related flags each set in at least one operation.

A computer screen shot of a computer

Description automatically generated

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1006, MOV is performed using values of size word:   * DST = 0x0065 * DST = FF0F   This will produce:   * DST = 0xFF0F |  |
| At address 100A, MOV is performed using values of size byte:   * DST.lowbyte = 0x0F * DST.lowbyte <- 0x65   This will produce:   * DST.lowbyte = 0x65 |  |

The MOV instruction works as expected. It can execute with both bytes and words.

**Pass/Fail:** PASS

### SWAP instructions work as expected

**Purpose:** Checks for successful execution of SWAP instruction.

**Configuration:** The following program is loaded into the emulator. SWAP is executed.

**A screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1006, SWAP is performed:   * SRC = 0x0065 * DST = 0xFF0F   This will produce:   * SRC = 0xFF0F * DST = 0x0065 |  |

The SWAP instruction works as expected.

**Pass/Fail:** PASS

### SRA instructions work as expected

**Purpose:** Checks for successful execution of SRA instruction.

**Configuration:** The following program is loaded into the emulator. SRA is executed with byte and word.

**A computer screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1002, SRA is performed using values of size word:   * DST = 0xFF00 >> 1   This will produce:   * DST = 0x7F80 * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At address 1006, SRA is performed using values of size byte.   * DST.lowbyte <- 0x01 >> 1   This will produce:   * RESULT = 0x00 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * SET carry flag |  |

The SRA instruction works as expected. It can execute with both bytes and words, and sets the flags as expected.

**Pass/Fail:** PASS

### RRC instructions work as expected

**Purpose:** Checks for successful execution of RRC instruction.

**Configuration:** The following program is loaded into the emulator. RRC is executed with byte and word.

**A screenshot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At memory address 1006, RRC is performed using values of size word:   * DST = 0x7FFD * DST <- 0x7FFD >> 1 through carry(0)   This will produce:   * DST = 0x3FFE * CLEAR overflow flag * CLEAR negative flag * CLEAR zero flag * SET zero flag * SET carry flag |  |
| At address 1008, RRC is performed using values of size byte:   * DST.lowbyte = 0xFE * DST.lowbyte <- 0xFE >> 1 through carry (1)   This will produce:   * DST.lowbyte = 0xFF * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At address 100C, RRC is performed using values of size word:   * DST = 0x0001 * DST <- 0xF001 >> 1 through carry (0)   This will produce:   * DST = 0x0000 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * SET carry flag |  |

The RRC instruction works as expected. It can execute with both bytes and words, and sets the flags as expected.

**Pass/Fail:** PASS

### SWPB instructions work as expected

**Purpose:** Checks for successful execution of SWPB instruction.

**Configuration:** The following program is loaded into the emulator. SWPB is executed.

**A screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At address 1000, SWPB is performed:   * DST = 0x0000   This will produce:   * DST = 0x0000 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * CLEAR carry flag |  |
| At address 1004, SWPB is performed:   * DST = 0x00FA   This will produce:   * DST = 0xFA00 * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At address 1008, SWPB is performed:   * DST = 0xFA09   This will produce:   * DST = 0x09FA * CLEAR overflow flag * CLEAR negative flag * CLEAR zero flag * CLEAR carry flag |  |

The SWPB instruction works as expected. It sets registers as expected

**Pass/Fail:** PASS

### SXT instructions work as expected

**Purpose:** Checks for successful execution of SXT instruction.

**Configuration:** The following program is loaded into the emulator. SXT is executed.

**A computer screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| At memory address 1000, SXT is performed:   * DST = 0x0000   This will produce:   * DST = 0x0000 * CLEAR overflow flag * CLEAR negative flag * SET zero flag * CLEAR carry flag |  |
| At address 1004, SXT is performed:   * DST = 0x00FA   This will produce:   * RESULT = 0xFFFA * CLEAR overflow flag * SET negative flag * CLEAR zero flag * CLEAR carry flag |  |
| At address 1008, SXT is performed:   * DST = 0x000A   This will produce:   * RESULT = 0x000A * CLEAR overflow flag * CLEAR negative flag * CLEAR zero flag * CLEAR carry flag |  |

The SXT instruction works as expected. It sets registers as expected.

**Pass/Fail:** PASS

### Encounter instruction not a part of A2 requirements

**Purpose:** Checks for successful execution of BIS instruction.

**Configuration:** The following program is loaded into the emulator. BIS is executed with byte and word, register and constant sources, and related flags each set in at least one operation.

**A screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **Expected Results** | **Actual Results** |
| * The emulator will not execute any commands. |  |

The emulator behaved as expected.

**Pass/Fail:** PASS