

OneUp Wi-11 – Test Plan

Primary Author: Joshua Green

Table of Contents

Simulator Tests:	4
Simulator Test: testcase1.txt	5
Simulator Test: testcase2.txt	6
Assembler Tests:	7
Assembler Test: aliasing_an_alias.s	8
Assembler Test: case_symbol.s	9
Assembler Test: commented_nonsense.s	10
Assembler Test: exmaple1.s	11
Assembler Test: invalid_br1.s	12
Assembler Test: invalid_br3.s	13
Assembler Test: invalid_hex1.s	14
Assembler Test: invalid_hex2.s	15
Assembler Test: invalid_hex.s	16
Assembler Test: invalid_op1.s	17
Assembler Test: invalid_op2.s	18
Assembler Test: invalid_op3.s	19
Assembler Test: invalid_op4.s	20
Assembler Test: invalid_orig_decimal.s	21
Assembler Test: invalid_orig_decimal2.s	22
Assembler Test: invalid_orig_hex.s	23
Assembler Test: invalid_strz.s	24
Assembler Test: lab2EG.s	25
Assembler Test: lab2EG_invalid_alias.s	27
Assembler Test: lab2EG_invalid_register_alias.s	28
Assembler Test: lab2EG_invalid_segment_name.s	29
Assembler Test: lab2EG_invalid_whitespace.s	30
Assembler Test: lab2EG_relocatable.s	31
Assembler Test: lab2EG.s	33
Assembler Test: lab2EG_valid_whitespace.s	35
Assembler Test: no_closing_quote.s	37
Assembler Test: no_end_record.s	38
Assembler Test: nonsense.s	39
Assembler Test: page_branching.s	40
Assembler Test: random_neg.s	41
Assembler Test: relocate.s	42
Assembler Test: segment_length_overflow.s	43
Assembler Test: similar_symbols_ok.s	44
Assembler Test: starting_outside_bounds.s	46
Assembler Test: valid_br1.s	47
Assembler Test: valid_br2.s	48
Assembler Test: weird_quotation.s	49
Assembler Test: invalid_symbols.s	50
Assembler/Linker/Simulator Tests:	51
Assembler/Linker/Simulator Test: Hello Friend [Linked]	52
Assembler/Linker/Simulator Test: Sqrt and Math [Linked]	53
Assembler/Linker/Simulator Test: Sqrt and Math [Linked] (Valid OS Malloc)	55

Assembler/Linker/Simulator Test:	Sqrt and Math [Linked] (Valid OS Malloc #2).....	56
Assembler/Linker/Simulator Test:	Sqrt and Math [Linked] (Invalid OS Malloc).....	57

Simulator Tests:

Simulator Test: testcase1.txt

Description:

A simple test of the different ADD and AND instructions, as well as branching, debug statements, and the halt trap.

Input:

```
HTESTAA0001000D
T00011434
T000216BA
T000316C2
T000458C2
T00055B25
T00060000
T00070E0A
T0008F043
T0009F043
T000A8000
T000BF025
E0001
```

This test case corresponds to following instructions:

```
ADD r2 r0 -12
ADD r3 r2 -5
ADD r3 r3 r2
AND r4 r3 r2
AND r5 r4 5
BRx 0 0 0 0 (no op)
BRx 1 1 1 10 (break always, offset 10)
TRAP 43
TRAP 43
DEBUG
TRAP 43
```

Output (abbr.):

```
ADD op: R2 = R0 + 0xFFFF4 :: Successful.
ADD op: R3 = R2 + 0xFFFFA :: Successful.
ADD op: R3 = R3 + R2 :: Successful.
AND op: R4 = R3 & R2 :: Successful.
AND op: R5 = R4 & 0x0005 :: Successful.
BRx op: [0|0|0] [Offset=0x0000] To Address: 0x0000 :: Branch ignored.
BRx op: [1|1|1] [Offset=0x000A] To Address: 0x000A :: Successful.
DEBUG op: ...
PC = 0x000B
R0 = 0x0000        R1 = 0x0000
R2 = 0xFFFF4      R3 = 0xFFE2...
M[0x0001] = 0x1434        M[0x0002] = 0x16BA        M[0x0003] = 0x16C2 ...
TRAP(0x0025) op: Execution has been terminated (Halt).
```

Simulator Test: testcase2.txt

Description:

A test of several more instructions, including the random trap and the different load and store instructions.

Input:

```
HTEST0200010014
T0001F043
T00029200
T00035001
T0004C80F
T00052A0D
T0006BA0C
T00073A0B
T0008AC0D
T00094014
T000F1624
T0010182B
T0011360C
T0012760D
T0013D55D
T0014F025
E0001
```

This test case corresponds to the following instructions:

```
TRAP 43
NOT r1 r0
AND r0 r0 r1
JSRR 1 r0 15'
LD r5 13
STI r5 12
ST r5 11
LDI r6 13
JSR 0 20
ADD r3 r0 4
ADD r4 r0 1 11
ST r3 12
STR r3 13
RET
TRAP 25
```

Output (abbr.):

```
TRAP(0x0043) op: Successful.
NOT op: R1 = ~R0 :: Successful.
AND op: R0 = R0 & R1 :: Successful.
JSRR op: (Store PC=1) [Base Register=R0] [Index=0x000F] To Address: 0x000F :: Successful.
ADD op: R3 = R0 + 0x0004 :: Successful.
ADD op: R4 = R0 + 0x000B :: Successful.
ST op: Memory[0x000C] = R3 [Offset=0x000C] :: Successful.
STR op: Memory[R0 + 0x000D] = R3 :: Successful.
RET op: Successful.
LD op: R5 = Memory[0x000D] [Offset=0x000D] :: Successful.
STI op: Memory[0x0004] = R5 [Offset=0x000C] :: Successful.
ST op: Memory[0x000B] = R5 [Offset=0x000B] :: Successful.
LDI op: R6 = Memory[0x0004] [Indirect Address=0x000D] [Offset=0x000D] :: Successful.
JSR op: (Store PC=0) [Immediate=0x0014] To Address: 0x0014 :: Successful.
TRAP(0x0025) op: Execution has been terminated (Halt).
```

Assembler Tests:

Assembler Test: aliasing_an_alias.s

Description:

Test the validity of a label assigned by the value of another label.

File Contents:

```
;Should be valid
Lab2EG    .ORIG x0
count     .FILL    #4
Begin     LD      ACC,count
ACC2      .EQU     #1
ACC       .EQU     ACC2           ; This should be okay.
         .END     Begin
```

Output:

```
(0000) 0004 00000000000000100 ( 2) Lab2EG    .ORIG x0
(0001) 2200 00100010000000000 ( 3) count     .FILL    #4
( 4) Begin     LD      ACC,count
( 5) ACC2      .EQU     #1
( 6) ACC       .EQU     ACC2           ; This should be okay.
( 7)          .END     Begin
```


Assembler Test: case_symbol.s

Description:

To test the sensitivity of capitalization.

File Contents:

```
; Symbols of different capitalization
Lab2EG .ORIG x0
Begin BRNZP next
NEXT BRNZP Begin
        .END Begin
```

Output:

```
                ( 2) Lab2EG .ORIG x0

--- In line: Begin BRNZP next
--- Error: next: Label not found.
((Forward reference to .FILL label? Case-sensitivity issue?))
```

Assembler Test: commented_nonsense.s

Description:

Test a file containing only comments.

File Contents:

```
;Mary had a little lamb. Her fleece was white as snow.  
;Everywhere that Mary went, her sheep were sure to go.
```

Output:

```
Error: Unexpected end of file.
```

Assembler Test: exmaple1.s

Description:

Test a valid program for success.

File Contents:

```
; A valid program.
Prog .ORIG x3020
HALT .EQU x25
Begin LD R0,N          ;R0 <- #13
      LD R1,#16        ;R1 <- #16
      ST R0,Ans        ;M[Ans]<-R0
      TRAP HALT
N      .FILL #13
Ans    .BLKW #1
      .END Begin
```

Output:

```
( 2) Prog .ORIG x3020
( 3) HALT .EQU x25
(3020) 2024 0010000000100100 ( 4) Begin LD R0,N          ;R0 <- #13
(3021) 2226 0010001000100110 ( 5)          LD R1,#16        ;R1 <- #16
(3022) 3025 0011000000100101 ( 6)          ST R0,Ans        ;M[Ans]<-R0
(3023) F025 1111000000100101 ( 7)          TRAP HALT
(3024) 000D 0000000000001101 ( 8) N      .FILL #13
(3025)          ( 9) Ans    .BLKW #1
(3026) 0010 0000000000010000 ( lit) <16>
          ( 10)          .END Begin

>>> out.o
HProg 30200007
T30202024
T30212226
T30223025
T3023F025
T3024000D
T30260010
E3020
```

Assembler Test: invalid_br1.s

Description:

Test for a invalid branch instruction by appending an invalid CCR mask.

File Contents:

```
; Invalid Branch Operation
Prog .ORIG
HALT .EQU x25
Begin BRNZPR Begin ; This shouldnt be valid
.END Begin
```

Output:

Error: Line 4: Invalid CCR mask for branch instruction.

Assembler Test: invalid_br3.s

Description:

Test for a invalid branch instruction by only an invalid CCR mask.

File Contents:

```
; Invalid Branch Qualifiers
Prog .ORIG
HALT .EQU x25
Begin BRG Begin ; This shouldnt be valid
.END Begin
```

Output:

Error: Line 4: Invalid CCR mask for branch instruction.

Assembler Test: invalid_hex1.s

Description:

Test for an invalid hex string that has valid range but lacking capitalization.

File Contents:

```
; Invalid hex string (not capitals)
Prog .ORIG
HALT .EQU xaa ; This isnt a valid hex string (not capital)
Begin TRAP x25
      .END Begin
```

Output:

Error: Line 3: Invalid hex following 'x'.

Assembler Test: invalid_hex2.s

Description:

Test for an invalid hex string that has non-hex values.

File Contents:

```
; Invalid hex string
Prog .ORIG
HALT .EQU xGG ; This isnt a valid hex string
Begin TRAP x25
      .END Begin
```

Output:

Error: Line 3: Invalid hex following 'x'.

Assembler Test: invalid_hex.s

Description:

Test the boundaries of hex values assigned to a label.

File Contents:

```
; Out of bounds .EQU hex string
Poop .ORIG      ; This is invalid.
gAH .EQU      x10000
Begin BRNZP    Begin
      .END     Begin
```

Output:

Error: Line 3: Invalid hex following 'x'.

Assembler Test: invalid_op1.s

Description:

Test the input of an invalid/undefined operation.

File Contents:

```
; Invalid Operation
Prog .ORIG
HALT .EQU x25
Begin UnDef Begin ; This command doesnt exist.
    .END Begin
```

Output:

```
Error: Line 4: Instruction not recognized.
```

Assembler Test: invalid_op2.s

Description:

Test the input of an invalid operation based on the capitalization of the command.

File Contents:

```
; Invalid Operation
Prog .ORIG
HALT .EQU x25
Begin Add Begin ; This command isnt all capitals.
      .END Begin
```

Output:

Error: Line 4: Instruction not recognized.

Assembler Test: invalid_op3.s

Description:

Test the input of an invalid operation which resembles a valid command.

File Contents:

```
; Invalid Operation
Prog .ORIG
HALT .EQU x25
Begin ADS Begin ; This command isnt defined but similar to a defined operation.
.END Begin
```

Output:

Error: Line 4: Instruction not recognized.

Assembler Test: invalid_op4.s

Description:

Test the input of an invalid operation which resembles a valid command.

File Contents:

```
; Invalid Operation
Prog .ORIG
HALT .EQU x25
Begin ORR Begin ; This command isnt defined but similar to a defined operation.
.END Begin
```

Output:

Error: Line 4: Instruction not recognized.

Assembler Test: invalid_orig_decimal.s

Description:

Test the requirement of .ORIG to be hexadecimal and within a certain range value.

File Contents:

```
; Invalid .Orig definition -- required to be hex and out of bounds value
; Example Program
Lab2Example    .ORIG    #65536
Begin    BRNZP    Begin
            .END      Begin
```

Output:

```
Error: Line 3: Argument to ".ORIG" not hex.
```

Assembler Test: invalid_orig_decimal2.s

Description:

Test the requirement of .ORIG to be hexadecimal.

File Contents:

```
; Invalid .Orig definition -- required to be hex and out of bounds value
; Example Program
Lab2Example    .ORIG    #10
Begin    BRNZP    Begin
            .END      Begin
```

Output:

Error: Line 3: Argument to ".ORIG" not hex.

Assembler Test: invalid_orig_hex.s

Description:

Test the requirement of .ORIG to be within a valid range.

File Contents:

```
; Out of bounds hex string
Poop  .ORIG  x10000 ; This is invalid.
Begin BRNZP Begin
      .END  Begin
```

Output:

Error: Line 2: Invalid hex following 'x'.

Assembler Test: invalid_strz.s

Description:

Test the requirement .STRZ to be a ASCII string bounded by quotation marks.

File Contents:

```
; Invalid STRZ value
Lab2EG .ORIG x30B0
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ x68 ; This is an invalid string although its the letter h
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- 0xFFFF
      ST R0,Array ;M[Array] <- 0xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= 0xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

Error: Line 10: Argument to ".STRZ" is not a string.

Assembler Test: lab2EG.s

Description:

Test a relatively lengthy, but valid program.

File Contents:

```
; Should execute properly
; Example Program
Lab2EG .ORIG x30B0
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ "hi! "
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- xFFFF
      ST R0,Array ;M[Array] <- xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

```
( 2) Lab2EG .ORIG x30B0
(30B0) 0004 0000000000000100 ( 3) count .FILL #4
(30B1) 22B0 0010001010110000 ( 4) Begin LD ACC,count ;R1 <- 4
(30B2) E0B7 1110000010110111 ( 5) LEA R0,msg
(30B3) F022 1111000000100010 ( 6) loop TRAP x22 ;print "hi! "
(30B4) 127F 0001001001111111 ( 7) ADD ACC,ACC,#-1 ;R1--
(30B5) 02B3 0000001010110011 ( 8) BRP loop
(30B6) 40BC 0100000010111100 ( 9) JMP Next
(30B7) 0068 000000001101000 (10) msg .STRZ "hi! "
(30B8) 0069 000000001101001 (10)
(30B9) 0021 000000000100001 (10)
(30BA) 0020 000000000100000 (10)
(30BB) 0000 0000000000000000 (10)
(30BC) 5020 0101000000100000 (11) Next AND R0,R0,x0 ;R0 <- 0
(30BD) 9000 1001000000000000 (12) NOT R0,R0 ;R0 <- xFFFF
(30BE) 30C3 0011000011000011 (13) ST R0,Array ;M[Array] <- xFFFF
(30BF) EAC3 1110101011000011 (14) LEA R5,Array
(30C0) 2CC7 0010110011000111 (15) LD R6,#100 ;R6 <= #100
(30C1) 7141 0111000101000001 (16) STR R0,R5,#1 ;M[Array+1] <= xFFFF
(30C2) F025 1111000000100101 (17) TRAP x25
( 18) ACC .EQU #1
(30C3) ( 20) Array .BLKW #3
(30C6) 0010 0000000000010000 (21) .FILL x10
(30C7) 0064 0000000001100100 (lit) <100>
( 22) .END Begin
```

```
>>> out.o
Hlab2EG30B00018
T30B00004
T30B122B0
T30B2E0B7
T30B3F022
T30B4127F
```

T30B640BC
T30B70068
T30B80069
T30B90021
T30BA0020
T30BB0000
T30BC5020
T30BD9000
T30BE30C3
T30BFEAC3
T30C02CC7
T30C17141
T30C2F025
T30C60010
T30C70064
E30B0

Assembler Test: lab2EG_invalid_alias.s

Description:

Test the improper aliasing of a register.

File Contents:

```
;Example Program
Lab2EG .ORIG
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ "hi! "
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- 0xFFFF
      ST R0,Array ;M[Array] <- 0xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= 0xFFFF
      TRAP x25
ACC .EQU R1 ; This is an error.
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

```
Error: Line 18: R1: Label not found.
((Forward reference to .FILL label? Case-sensitivity issue?))
```

Assembler Test: lab2EG_invalid_register_alias.s

Description:

Test the usage of a label as a register which does not contain a register ID.

File Contents:

```
; Invalid use of register alias (ACC = 9)
Lab2EG .ORIG
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1-- ; This becomes an error.
      BRP loop
      JMP Next
msg .STRZ "hi! "
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- xFFFF
      ST R0,Array ;M[Array] <- xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= xFFFF
      TRAP x25
ACC .EQU #9 ; This is not a valid register.
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

```
( 2) Lab2EG .ORIG
(0000) 0004 0000000000000100 ( 3) count .FILL #4

--- In line: Begin LD ACC,count ;R1 <- 4
--- Error: Non-existent register as argument.
```

Assembler Test: lab2EG_invalid_segment_name.s

Description:

Test a segment name which is too long.

File Contents:

```
; Invalid Segment Name (Size is too long)
Lab2Example .ORIG x30B0
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ "hi! "
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- xFFFF
      ST R0,Array ;M[Array] <- xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

Error: Line 2: ".ORIG" label longer than six characters.

Assembler Test: lab2EG_invalid_whitespace.s

Description:

Test the improper placement of whitespace before a label.

File Contents:

```
; Invalid whitespace before symbols
Lab2EG .ORIG x30B0
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
      loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ "hi! "
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- 0xFFFF
      ST R0,Array ;M[Array] <- 0xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= 0xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

Error: Line 6: Instruction not recognized.

Assembler Test: lab2EG_relocatable.s

Description:

Test a valid relocatable source file.

File Contents:

```
;Example Program -- Relocatable: should be okay.
Lab2EG .ORIG
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ "hi! "
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- xFFFF
      ST R0,Array ;M[Array] <- xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

```
( 2) Lab2EG .ORIG
(0000) 0004 0000000000000100 ( 3) count .FILL #4
(0001) 2200 0010001000000000 ( 4) Begin LD ACC,count ;R1 <- 4
(0002) E007 1110000000000111 ( 5) LEA R0,msg
(0003) F022 1111000000100010 ( 6) loop TRAP x22 ;print "hi
! "
(0004) 127F 0001001001111111 ( 7) ADD ACC,ACC,#-1 ;R1--
(0005) 0203 0000001000000011 ( 8) BRP loop
(0006) 400C 01000000000001100 ( 9) JMP Next
(0007) 0068 00000000001101000 (10) msg .STRZ "hi! "
(0008) 0069 00000000001101001 (10)
(0009) 0021 0000000000100001 (10)
(000A) 0020 0000000000100000 (10)
(000B) 0000 0000000000000000 (10)
(000C) 5020 0101000000100000 (11) Next AND R0,R0,x0 ;R0 <- 0
(000D) 9000 1001000000000000 (12) NOT R0,R0 ;R0 <- xFF
FF
(000E) 3013 0011000000010011 (13) ST R0,Array ;M[Array]
<- xFFFF
(000F) EA13 1110101000010011 (14) LEA R5,Array
(0010) 2C17 0010110000010111 (15) LD R6,#100 ;R6 <= #10
0
(0011) 7141 0111000101000001 (16) STR R0,R5,#1 ;M[Array+1]
] <= xFFFF
(0012) F025 1111000000100101 (17) TRAP x25
( 18) ACC .EQU #1
(0013) ( 20) Array .BLKW #3
(0016) 0010 0000000000010000 ( 21) .FILL x10
(0017) 0064 00000000001100100 ( lit) <100>
( 22) .END Begin
```

```
>>> out.o
HLab2EG0018
T00000004
R00012200
R0002E007
T0003F022
T0004127F
R00050203
R0006400C
T00070068
T00080069
T00090021
T000A0020
T000B0000
T000C5020
T000D9000
R000E3013
R000FEA13
R00102C17
T00117141
T0012F025
T00160010
T00170064
E0000
```


Assembler Test: lab2EG.s

Description:

Test a valid non-relocatable source file.

File Contents:

```
; Example Program
Lab2EG .ORIG x30B0
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ "hi! "
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- xFFFF
      ST R0,Array ;M[Array] <- xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

```
( 3) Lab2EG .ORIG x30B0
(30B0) 0004 0000000000000100 ( 4) count .FILL #4
(30B1) 22B0 0010001010110000 ( 5) Begin LD ACC,count ;R1 <- 4
(30B2) E0B7 1110000010110111 ( 6) LEA R0,msg
(30B3) F022 1111000000100010 ( 7) loop TRAP x22 ;print "hi! "
(30B4) 127F 0001001001111111 ( 8) ADD ACC,ACC,#-1 ;R1--
(30B5) 02B3 0000001010110011 ( 9) BRP loop
(30B6) 40BC 0100000010111100 (10) JMP Next
(30B7) 0068 0000000001101000 (11) msg .STRZ "hi! "
(30B8) 0069 0000000001101001 (11)
(30B9) 0021 0000000000100001 (11)
(30BA) 0020 0000000000100000 (11)
(30BB) 0000 0000000000000000 (11)
(30BC) 5020 0101000000100000 (12) Next AND R0,R0,x0 ;R0 <- 0
(30BD) 9000 1001000000000000 (13) NOT R0,R0 ;R0 <- xFFFF
(30BE) 30C3 0011000011000011 (14) ST R0,Array ;M[Array] <- xFFFF
(30BF) EAC3 1110101011000011 (15) LEA R5,Array
(30C0) 2CC7 0010110011000111 (16) LD R6,#100 ;R6 <= #100
(30C1) 7141 0111000101000001 (17) STR R0,R5,#1 ;M[Array+1] <= xFFFF
(30C2) F025 1111000000100101 (18) TRAP x25
( 18) ACC .EQU #1
(30C3) ( 21) Array .BLKW #3
(30C6) 0010 0000000000010000 ( 22) .FILL x10
(30C7) 0064 0000000001100100 (lit) <100>
( 23) .END Begin
```

```
>>> out.o
HLab2EG30B00018
T30B00004
T30B122B0
T30B2E0B7
T30B3F022
T30B4127F
T30B502B3
T30B640BC
```

T30B70068
T30B80069
T30B90021
T30BA0020
T30BB0000
T30BC5020
T30BD9000
T30BE30C3
T30BFEAC3
T30C02CC7
T30C17141
T30C2F025
T30C60010
T30C70064
E30B0

Assembler Test: lab2EG_valid_whitespace.s

Description:

Test a valid use of tabs and spaces for whitespace.

File Contents:

```
; This should be fine. Tabs and whitespaces
Lab2EG .ORIG x30B0
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ "hi! "
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- xFFFF
      ST R0,Array ;M[Array] <- xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

```
( 2) Lab2EG .ORIG x30B0
(30B0) 0004 0000000000000100 ( 3) count .FILL #4
(30B1) 22B0 0010001010110000 ( 4) Begin LD ACC,count ;R1 <- 4
(30B2) E0B7 1110000010110111 ( 5) LEA R0,msg
(30B3) F022 1111000000100010 ( 6) loop TRAP x22 ;print
"hi! "
(30B4) 127F 0001001001111111 ( 7) ADD ACC,ACC,#-1 ;R1--
(30B5) 02B3 0000001010110011 ( 8) BRP loop
(30B6) 40BC 0100000010111100 ( 9) JMP Next
(30B7) 0068 000000001101000 (10) msg .STRZ "hi! "
(30B8) 0069 000000001101001 (10)
(30B9) 0021 000000000100001 (10)
(30BA) 0020 000000000100000 (10)
(30BB) 0000 0000000000000000 (10)
(30BC) 5020 0101000000100000 (11) Next AND R0,R0,x0 ;R0 <- 0
(30BD) 9000 1001000000000000 (12) NOT R0,R0 ;R0 <- xFFFF
(30BE) 30C3 0011000011000011 (13) ST R0,Array
;M[Array] <- xFFFF
(30BF) EAC3 1110101011000011 (14) LEA R5,Array
(30C0) 2CC7 0010110011000111 (15) LD R6,#100 ;R6 <=
#100
(30C1) 7141 0111000101000001 (16) STR R0,R5,#1 ;M[Array+1] <=
xFFFF
(30C2) F025 1111000000100101 (17) TRAP x25
( 18) ACC .EQU #1
(30C3) ( 20) Array .BLKW #3
(30C6) 0010 0000000000010000 (21) .FILL x10
(30C7) 0064 0000000001100100 (lit) <100>
( 22) .END Begin

>>> out.o
Hlab2EG30B00018
T30B00004
T30B122B0
```

T30B3F022
T30B4127F
T30B502B3
T30B640BC
T30B70068
T30B80069
T30B90021
T30BA0020
T30BB0000
T30BC5020
T30BD9000
T30BE30C3
T30BFEAC3
T30C02CC7
T30C17141
T30C2F025
T30C60010
T30C70064
E30B0

Assembler Test: no_closing_quote.s

Description:

Test a string not terminated by an end-quote.

File Contents:

```
; Example Program
Lab2EG .ORIG x30B0
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ "hi!hhh; no closing quote
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- xFFFF
      ST R0,Array ;M[Array] <- xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

Error: Line 10: End of string not found.

Assembler Test: no_end_record.s

Description:

Test a source file with a missing .END statement.

File Contents:

```
; No End Record
Lab2EG .ORIG x30B0
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ "hi! "
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- xFFFF
      ST R0,Array ;M[Array] <- xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
```

Output:

```
( 2) Lab2EG .ORIG x30B0
( 3) count .FILL #4
( 4) Begin LD ACC,count ;R1 <- 4
( 5) LEA R0,msg
( 6) loop TRAP x22 ;print "hi! "
( 7) ADD ACC,ACC,#-1 ;R1--
( 8) BRP loop
( 9) JMP Next
(10) msg .STRZ "hi! "
(10)
(10)
(10)
(10)
(11) Next AND R0,R0,x0 ;R0 <- 0
(12) NOT R0,R0 ;R0 <- xFFFF
(13) ST R0,Array ;M[Array] <- xFFFF
(14) LEA R5,Array
(15) LD R6,#100 ;R6 <= #100
(16) STR R0,R5,#1 ;M[Array+1] <= xFFFF
(17) TRAP x25
(18) ACC .EQU #1
(20) Array .BLKW #3
(21) .FILL x10
Error: File has no end record.
```

Assembler Test: nonsense.s

Description:

Test inputting a file that is not meant to be a source file.

File Contents:

Mary had a little lamb. Her fleece was white as snow.
Everywhere that Mary went, her sheep were sure to go.

Output:

Error: Line 1: Instruction not recognized.

Assembler Test: page_branching.s

Description:

Test branching to another page.

****Note:** The command line flag “-s10000” was used.

File Contents:

```
; This program attempts to branch to a different page
Test02  .ORIG  x0 ; 1501
Begin   AND    R0,R0,#0      ; Bitmask R0 to all zeroes
        .BLKW  xFFF
        TRAP   rND
        BRZP   Begin        ; If the random number generated was non-negative, branch
        TRAP   HALT         ; Otherwise, halt
HALT    .EQU   x25
rND     .EQU   x43
        .END    Begin
```

Output:

```
(0000) 5020 0101000000100000 ( 2) Test02  .ORIG  x0 ; 1501
(0001) 5021 0101000000100000 ( 3) Begin   AND    R0,R0,#0      ; Bitmask R
0 to all zeroes
(0001) 5022 0101000000100000 ( 4)          .BLKW  xFFF
(1000) F043 1111000001000011 ( 5)          TRAP   rND

--- In line:          BRZP   Begin        ; If the random number generated wa
s non-negative, branch
--- Error: Page Error: Address references a different page.
```


Assembler Test: random_neg.s

Description:

Test using a value that is not representable by the required size vector.

File Contents:

```
; This short program counts how many random numbers are generated before a
; negative is produced.
Test01  .ORIG  x30B0
inc     .FILL  #1
Begin   AND    R1,R1,#0      ; Bitmask R1 to all zeroes
Loop    TRAP   rND
        BRN    End          ; Branch to end of program if the N CCR is set
        ADD    R1,R1,inc     ; Increment R1
        BRNZP  Loop         ; Unconditionally branch back to Loop
End      AND    R0,R0,x1F     ; Bitmask R0 to all ones
        AND    R0,R0,R1      ; Copy the value of R1 into R0
        TRAP   OUTN          ; Print R0 to the console as a decimal integer
        TRAP   HALT
HALT    .EQU    x25
rND     .EQU    x43
        .END    Begin
```

Output:

```
( 3) Test01  .ORIG  x30B0
(30B0) 0001 0000000000000001 ( 4) inc     .FILL  #1
(30B1) 5260 0101001001100000 ( 5) Begin   AND    R1,R1,#0      ; Bitmask R1 to all zeroes
(30B2) F043 1111000001000011 ( 6) Loop    TRAP   rND
(30B3) 08B6 0000100010110110 ( 7)         BRN    End          ; Branch to end of program if the N
CCR is set

--- In line:   ADD    R1,R1,inc     ; Increment R1
--- Error: Immediate value not expressible in 5 bits.
```

Assembler Test: relocate.s

Description:

Test a valid relocatable program.

File Contents:

```
; A valid relocatable program
Prog .ORIG
X .FILL Y
Halt .EQU x25
Start LD R1,X
      ST R1,Y
      TRAP Halt
Y .BLKW #1
  .END Start
```

Output:

```
( 2) Prog .ORIG
( 3) X .FILL Y
( 4) Halt .EQU x25
(0000) 0004 00000000000000100 ( 5) Start LD R1,X
(0001) 2204 00100010000000100 ( 6) ST R1,Y
(0002) 3204 00110010000000100 ( 7) TRAP Halt
(0003) F025 1111000000100101 ( 8) Y .BLKW #1
(0004) ( 9) .END Start
```

```
>>> out.o
HProg 0005
W00000004
R00012204
R00023204
T0003F025
E0000
```

Assembler Test: segment_length_overflow.s

Description:

Test a source file which produces too large of a object-file segment size.

File Contents:

```
; Object File Length too large...
Lab2EG .ORIG x30B0
count .BLKW xFFFF
Begin LD R1,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD R1,R1,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ "hi! "
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- xFFFF
      ST R0,Array ;M[Array] <- xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

```
Error: (Max Length: 65535): Maximum object file size reached.
((Alter with '-s?'))
```

Assembler Test: similar_symbols_ok.s

Description:

Test a using symbols that resemble each other but are different cases.

File Contents:

```
; Similar symbols but should be okay.
Lab2EG .ORIG x30B0
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg BRNZP mSg
mSg .STRZ "hi! "
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- xFFFF
      ST R0,Array ;M[Array] <- xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

```
( 2) Lab2EG .ORIG x30B0
(30B0) 0004 0000000000000100 ( 3) count .FILL #4
(30B1) 22B0 0010001010110000 ( 4) Begin LD ACC,count ;R1 <- 4
(30B2) E0B7 1110000010110111 ( 5)      LEA R0,msg
(30B3) F022 1111000000100010 ( 6) loop TRAP x22 ;print
"hi! "
(30B4) 127F 0001001001111111 ( 7)      ADD ACC,ACC,#-1 ;R1--
(30B5) 02B3 0000001010110011 ( 8)      BRP loop
(30B6) 40BD 0100000010111101 ( 9)      JMP Next
(30B7) 0EB8 0000111010111000 (10) msg BRNZP mSg
(30B8) 0068 0000000001101000 (11) mSg .STRZ "hi! "
(30B9) 0069 0000000001101001 (11)
(30BA) 0021 0000000000100001 (11)
(30BB) 0020 0000000000100000 (11)
(30BC) 0000 0000000000000000 (11)
(30BD) 5020 0101000000100000 (12) Next AND R0,R0,x0 ;R0 <- 0
(30BE) 9000 1001000000000000 (13)      NOT R0,R0 ;R0 <- xFFFF
(30BF) 30C4 0011000011000100 (14)      ST R0,Array
;M[Array] <- xFFFF
(30C0) EAC4 1110101011000100 (15)      LEA R5,Array
(30C1) 2CC8 001011100110010000 (16)      LD R6,#100 ;R6 <=
#100
(30C2) 7141 0111000101000001 (17)      STR R0,R5,#1 ;M[Array+1] <=
xFFFF
(30C3) F025 1111000000100101 (18)      TRAP x25
( 19) ACC .EQU #1
(30C4) ( 21) Array .BLKW #3
(30C7) 0010 0000000000010000 ( 22)      .FILL x10
(30C8) 0064 0000000001100100 ( lit) <100>
( 23)      .END Begin
```

```
>>> out.o
HLab2EG30B00019
T30B00004
T30B122B0
```

T30B2E0B7
T30B3F022
T30B4127F
T30B502B3
T30B640BD
T30B70EB8
T30B80068
T30B90069
T30BA0021
T30BB0020
T30BC0000
T30BD5020
T30BE9000
T30BF30C4
T30C0EAC4
T30C12CC8
T30C27141
T30C3F025
T30C70010
T30C80064
E30B0

Assembler Test: starting_outside_bounds.s

Description:

Test an invalid .END record that references a memory location outside the program.

File Contents:

```
BEGIN   .ORIG x10  
      .END #0
```

Output:

Error: Line 3: Argument to ".END" instruction is outside declared memory.

Assembler Test: valid_br1.s

Description:

Test the design of repetitive branching masks.

File Contents:

```
; Valid Branch Operation
Prog .ORIG
HALT .EQU x25
Begin BRNZPZZPPNNZP Begin ; This should be valid
.END Begin
```

Output:

```
( 2) Prog .ORIG
( 3) HALT .EQU x25
(0000) 0E00 0000111000000000 ( 4) Begin BRNZPZZPPNNZP Begin ; This should be valid
( 5) .END Begin

>>> out.o
HProg 0001
R00000E00
E0000
```

Assembler Test: valid_br2.s

Description:

Test the design of no branch masks – a NOP.

File Contents:

```
; Invalid Branch Qualifiers
Prog .ORIG
HALT .EQU x25
Begin BR Begin ; This should be valid
.END Begin
```

Output:

```
( 2) Prog .ORIG
( 3) HALT .EQU x25
(0000) 0000 0000000000000000 ( 4) Begin BR Begin ; This should be valid
( 5) .END Begin

>>> out.o
HProg 0001
R00000000
E0000
```


Assembler Test: weird_quotation.s

Description:

Test the case of extra characters found after the end of a string.

File Contents:

```
; Example Program
Lab2EG .ORIG x30B0
count .FILL #4
Begin LD ACC,count ;R1 <- 4
      LEA R0,msg
loop TRAP x22 ;print "hi! "
      ADD ACC,ACC,#-1 ;R1--
      BRP loop
      JMP Next
msg .STRZ "hi"!hhh ; weird closing quote
Next AND R0,R0,x0 ;R0 <- 0
      NOT R0,R0 ;R0 <- xFFFF
      ST R0,Array ;M[Array] <- xFFFF
      LEA R5,Array
      LD R6,#100 ;R6 <= #100
      STR R0,R5,#1 ;M[Array+1] <= xFFFF
      TRAP x25
ACC .EQU #1
; ----- Scratch Space -----
Array .BLKW #3
      .FILL x10
      .END Begin
```

Output:

Error: Line 10: Extra characters found after end of string.

Assembler Test: invalid_symbols.s

Description:

Test the invalid naming of symbols

File Contents:

```
; A invalid program.
Prog .ORIG x3020
HALT .EQU x25
xBegin LD R0,N                ;R0 <- #13
        LD R1,#16            ;R1 <- #16
        ST R0,Ans            ;M[Ans]<-R0
        TRAP HALT
N        .FILL #13
Ans      .BLKW #1
        .END xBegin
```

Output:

Error: Line 4: Label starting with 'R' or 'x'.

Assembler/Linker/Simulator Tests:

Assembler/Linker/Simulator Test: Hello Friend [Linked]

Description:

Linking two separate files, testing external symbols and entry points as well as linker offsetting and .MAIN designation.

Prints, "Hello friend."

Output:

Run with > ./will -v -t -ox link.o sys_test/link1.s sys_test/link2.s

Assembling...

```
-- sys_test/link1.s:
( 1) .MAIN
( 2) link1 .ORIG
( 3) .EXT foo,bar
(0000) 0020 0000000000100000 ( 4) friend .STRZ " friend."
(0001) 0066 0000000001100110 ( 4)
(0002) 0072 0000000001110010 ( 4)
(0003) 0069 0000000001101001 ( 4)
(0004) 0065 0000000001100101 ( 4)
(0005) 006E 0000000001101110 ( 4)
(0006) 0064 0000000001100100 ( 4)
(0007) 002E 0000000000101110 ( 4)
(0008) 0000 0000000000000000 ( 4)
(0009) 4800 0100100000000000 ( 5) JSR foo
(000A) E000 1110000000000000 ( 6) LEA R0,friend
(000B) 4800 0100100000000000 ( 7) JSR bar
(000C) F025 1111000000100101 ( 8) TRAP halt
( 9) .END
```

```
-- sys_test/link2.s:
( 1) link2 .ORIG
( 2) .ENT foo,bar
(0000) 0068 0000000001101000 ( 3) hello .STRZ "hello"
(0001) 0065 0000000001100101 ( 3)
(0002) 006C 0000000001101100 ( 3)
(0003) 006C 0000000001101100 ( 3)
(0004) 006F 0000000001101111 ( 3)
(0005) 0000 0000000000000000 ( 3)
(0006) E000 1110000000000000 ( 4) foo LEA R0,hello
(0007) F022 1111000000100010 ( 5) bar TRAP puts
(0008) D000 1101000000000000 ( 6) RET
( 7) .END
```

Linking... Done.

Executing...

Requesting address for program length of 22 (Ex: 0x0000) : 0x0000

hello friend.

Assembler/Linker/Simulator Test: Sqrt and Math [Linked]

Description:

Linking two separate files, testing external symbols and entry points as well as linker offsetting and .MAIN designation. Obtains the square root of a user-defined integer at run-time.

Output:

Run with > ./will -d -ox sqrt.o sys_test/sqrt.s sys_test/math.s

Assembling...

-- sys_test/sqrt.s:

```
( 10) .MAIN
( 11) runsqt .ORIG
( 12) .EXT    Sqrt,PRESQRT,PSTSQRT
(0000) 0069 0000000001101001 ( 14) imgnry  .STRZ    "i"          ; An i for if the square root is
imaginary
(0001) 0000 0000000000000000 ( 14)
(0002) 000A 0000000000001010 ( 15) newln   .FILL    xA          ; ASCII value for a newline
( 17) out     .EQU    x21
( 18) puts    .EQU    x22
( 19) halt    .EQU    x25
( 20) outn    .EQU    x31
( 21) inn     .EQU    x33
(0003) F033 1111000000110011 ( 23) start   TRAP     inn          ; print prompt & get number
(0004) 1820 0001100000100000 ( 24)         ADD     R4,R0,#0      ; R4 = R0
(0005) E000 1110000000000000 ( 25)         LEA     R0,PRESQRT     ; Get beginning output text
(0006) F022 1111000000100010 ( 26)         TRAP    puts          ; Print "SQRT("
(0007) 5020 0101000000100000 ( 27)         AND     R0,R0,#0       ; Clear R0
(0008) 1120 0001000100100000 ( 28)         ADD     R0,R4,#0       ; R0 = R4 (input number)
(0009) F031 1111000000110001 ( 29)         TRAP    outn         ; Print input number -- now have
"SQRT(#"
(000A) E000 1110000000000000 ( 30)         LEA     R0,PSTSQRT     ; Get last bit of text before
printing answer
(000B) F022 1111000000100010 ( 31)         TRAP    puts          ; Print ") = " -- now have "SQRT(#) =
"
(000C) 1920 0001100100100000 ( 32)         ADD     R4,R4,#0       ; Set CCR for R4 -- input number
(000D) 080F 0000100000001111 ( 35)         BRN     neg          ;
(000E) 0612 0000011000010010 ( 36)         BRZP    go          ;
(000F) 9900 1001100100000000 ( 38) neg     NOT     R4,R4       ;
(0010) 1921 0001100100100001 ( 39)         ADD     R4,R4,#1       ;
(0011) 14A1 0001010010100001 ( 40)         ADD     R2,R2,#1       ; R2 = 1 -- boolean for negative
(0012) 4800 0100100000000000 ( 42) go      JSR     SQRT        ;
(0013) F031 1111000000110001 ( 44)         TRAP    outn         ;
(0014) 14A0 0001010010100000 ( 45)         ADD     R2,R2,#0       ; Set CCR for R2
(0015) 0418 0000010000011000 ( 47)         BRZ     end          ;
(0016) E000 1110000000000000 ( 48)         LEA     R0,imgnry     ;
(0017) F022 1111000000100010 ( 49)         TRAP    puts          ;
(0018) 2002 0010000000000010 ( 51) end     LD     R0,newln     ; load a newline into R0
(0019) F021 1111000000100001 ( 52)         TRAP    out          ; print newline
(001A) F025 1111000000100101 ( 53)         TRAP    halt        ;
( 54)         .END     start
```

-- sys_test/math.s:

```
( 14) MATH    .ORIG
( 15)         .ENT     Sqrt,MULT,PRESQRT,PSTSQRT
(0000) 0053 0000000001010011 ( 16) PRESQRT .STRZ    "SQRT("      ; first thing to be printed
(0001) 0051 0000000001010001 ( 16)
(0002) 0052 0000000001010010 ( 16)
(0003) 0054 0000000001010100 ( 16)
(0004) 0028 0000000000101000 ( 16)
(0005) 0000 0000000000000000 ( 16)
(0006) 0029 0000000000101001 ( 17) PSTSQRT .STRZ    ") = "      ; to be printed after the input
number
(0007) 0020 0000000000100000 ( 17)
(0008) 003D 0000000000111101 ( 17)
(0009) 0020 0000000000100000 ( 17)
(000A) 0000 0000000000000000 ( 17)
(000B)         ( 18) sqrtpc  .BLKW    #1          ; store R7 while executing square
root
(000C) 5B60 0101101101100000 ( 21) SQRT    AND     R5,R5,#0      ; Clear R5
(000D) 1920 0001100100100000 ( 22)         ADD     R4,R4,#0      ; set CCR for R4
```

```

(000E) 3E0B 0011111000001011 ( 23)      ST      R7,sqrtpc    ; Save R7, to allow for calling MULT
(000F) 0419 0000010000011001 ( 24)      BRZ      sqfin      ; if R4 = 0, return 0
(0010) 1B61 0001101101100001 ( 25) sqnext ADD      R5,R5,#1    ; R5++
(0011) 1D60 0001110101100000 ( 26)      ADD      R6,R5,#0    ; R6 = R5
(0012) 481C 0100100000011100 ( 27)      JSR      MULT      ; R0 = R5*R6
(0013) 9000 1001000000000000 ( 29)      NOT      R0,R0      ; ~R0
(0014) 1021 0001000000100001 ( 30)      ADD      R0,R0,#1    ; R0++
(0015) 1004 0001000000000100 ( 32)      ADD      R0,R0,R4    ; R0 = R0 + R4
(0016) 0210 0000001000010000 ( 33)      BRP      sqnext
(0017) 0419 0000010000011001 ( 35)      BRZ      sqfin      ; don't decrement R5 if they are
equal
(0018) 1B7F 0001101101111111 ( 36)      ADD      R5,R5,#-1   ; R5-- (because R0 is too large)
(0019) 1160 0001000101100000 ( 37) sqfin  ADD      R0,R5,#0    ; R0 = R5
(001A) 2E0B 0010111000001011 ( 38)      LD        R7,sqrtpc    ; Restore R7
(001B) D000 1101000000000000 ( 39)      RET
(001C) 5020 0101000000100000 ( 42) MULT  AND      R0,R0,#0    ; Clear R0
(001D) 1DA0 0001110110100000 ( 43) mnnext ADD      R6,R6,#0    ; CCR corresponds to R6 value
(001E) 0C22 0000110000100010 ( 44)      BRNZ     done      ; while R6 > 0 -- shouldn't be
negative, though
(001F) 1005 0001000000000101 ( 45)      ADD      R0,R0,R5    ; R0 = R0 + R6
(0020) 1DBF 0001110110111111 ( 46)      ADD      R6,R6,#-1   ; R6--
(0021) 0E1D 0000111000011101 ( 47)      BRNZP    mnnext    ; recurse
(0022) D000 1101000000000000 ( 48) done  RET
( 49)      .END                      ; Should theoretically never get here

```

Linking... Done.

Executing...

Requesting address for program length of 62 (Ex: 0x0000) : 0x0000

Enter Integer: 9

SQRT(7) = 3

Assembler/Linker/Simulator Test: Sqrt and Math [Linked] (Valid OS Malloc)

Description:

Linking two separate files, testing external symbols and entry points as well as linker offsetting and .MAIN designation. Obtains the square root of a user-defined integer at run-time. **Run with a different OS Malloc value other than 0x000.**

Output:

```
Run with > ./will -x sqrt.o
```

```
Requesting address for program length of 62 (Ex: 0x0000) : 0x1234
```

```
Enter Integer: 689
```

```
SQRT(689) = 26
```

Assembler/Linker/Simulator Test: Sqrt and Math [Linked] (Valid OS Malloc #2)

Description:

Linking two separate files, testing external symbols and entry points as well as linker offsetting and .MAIN designation. Obtains the square root of a user-defined integer at run-time. **Run with a different OS Malloc value other than 0x000.**

Output:

```
Run with > ./will -x sqrt.o
```

```
Requesting address for program length of 62 (Ex: 0x0000) : 0x1234
```

```
Enter Integer: 689
```

```
SQRT(689) = 26
```


Assembler/Linker/Simulator Test: Sqrt and Math [Linked] (Invalid OS Malloc)

Description:

Linking two separate files, testing external symbols and entry points as well as linker offsetting and .MAIN designation. Obtains the square root of a user-defined integer at run-time. **Run with an invalid OS Malloc value (Page Error).**

Output:

```
Run with > ./will -x sqrt.o
```

```
Requesting address for program length of 62 (Ex: 0x0000) : 0xFFFF
```

```
Error: Bad Malloc from OS.
```