TESTING

Table Of Contents

***Content: Page Number***

Introduction **2**

Program Execution **2**

Assembly Input Syntax  **3**

Object Output **3**

Listing Output  **4**

Examples **4**

Introduction

Extensive testing was performed to uncover any logical errors within the execution of our assembler. Although we were unable to uncover any errors in execution this does not mean that our program is errorless. Furthermore, testing was performed to ensure that all syntax errors within an assembly language source file are found and reported to the user, in the listing-output-file, and that the output of the object file is correct and executable by the W10\_560\_Machine.

Although specific error messages are not referred to in this document a listing of all error messages as well as insight to the cause/resolution of the error is provided in the User’s Guide. This document is broken down into sections which explain the various types of testing methods, inputs and expected outputs:

1. **Program Execution**

-Explains testing performed on the actual execution of the assembler program from command line to termination.

1. **Assembly Input Syntax**

-Explains testing performed

1. **Object Output**

-Explains verification of correct object code output.

1. **Listing Output**

-Explains the output found in the listing file.

Program Execution

Testing was performed, using various command line inputs to determine whether or not the program will identify directories passed instead of files. Errors pertaining to incorrect execution of the program will be output to the stderr device. These tests passed for all three command line arguments passed resulting in an error message. Next a test to ensure that input files do exist was performed; upon execution with non-existent files as arguments the program returned error messages. A test to check read permissions on input files and write permissions on output files was performed with the error messages resulting when incorrect file permissions were set. Tests were run with input files containing undefined machine/pseudo operations and illegal operands to ensure that the program does not attempt to access indexed data out of range. In these cases error messages were reported in the listing-output pertaining to undefined operations and illegal operands; hence data was not indexed out of range. Finally, tests were run on individual components within the program to ensure that all while/for loops terminate as expected.

Although all of these tests passed as expected and no errors were uncovered it is still possible for logical errors to exist within the code. In the case that an error is uncovered a RESOLVE checking component will print meaningful information, pertaining to the source of the error, to the stderr device.

Assembly Input Syntax

The first test was to check for proper location of record elements. These tests were performed with a series of assembly-source-files containing records with elements not in proper position. These tests passed as the individual modules for parsing the label and operation returned errors when provided with incorrect input.

Next multiple tests were run on each of the machine/pseudo operations and synthetic instructions. These tests checked for the existence/non-existence of symbols on lines containing ORI, EQU and END pseudo operations, as well as the existence/non-existence of operands on all lines. A label which is already defined will produce an error message on any subsequent definitions of that label. It was observed that ORI and EQU require labels while END may not have a label and all operations/synthetic instructions, excluding ORI and END, must have operands. Furthermore, it was observed that all machine instructions must have at least two operands, R and S while only some require and X operand. Again error messages were observed when lines containing pseudo operations or synthetic instructions contained illegal operands.

Tests were conducted to ensure that one and only one ORI operation existed and that an END operation existed. These tests returned error messages when either condition was not met. In the case that lines preceded the ORI operation an error message was returned and in the case lines came after the END operation these lines were ignored.

Tests were run to ensure that an error message was produced if an empty line was encountered and that commented lines, beginning with ‘;’, produce no error messages. All of these tests passed.

In an effort to modify the assembler program, new pseudo operations, ENT and EXT, were added. The addition of these two new operations requires additional testing on the assembly input syntax to maximize the effectiveness and accuracy of the program. These test cases were designed for operation specific requirements, in an attempt to find any errors. These tests can be found in the testing documentation, under Lab\_3TestCases.txt.

Object Output

Verification of correct output given any errorless input file is necessary. To do this we ran multiple, correct, assembly source programs through the assembler. We visually checked that the object code corresponded correctly to the assembly source program; the first record began with ‘H’ and contained the execution-begin address, segment name, load address, segment length and an ‘M’ if the program is relocatable, all other records began with ‘T’, contained a valid address, five hex characters denoting the initial value at the specified address and an ‘M’ if the record is relocatable. It was ensured that no memory address appeared as the location address in more than one text record in a given object file. No spaces appear between consecutive records in an object file and the first line is the header record. All literals appear with addresses greater than the last operation and any RES operation will allocate a block of memory for which no records exist. Also ORI, END and EQU pseudo operations do not create records.

Listing Output

The listing-output file will contain any errors pertaining to incorrect syntax in the assembly-source file. This file will also contain a trace of object code creation, listing meaningful information about each specific line of object code produced as well as the line of source code used to create it. If errors exist in the listing-output file however no object code trace exists and the error occurred in the first pass of the assembler, likewise if an error exists and object code trace exists then the error occurred in the second pass of the assembler. This file can be helpful when debugging assembly-source input files. If the listing-output file contains no errors and only an object code trace then the assembler finished successfully and the produced object code is ready for execution.

Examples

**Assembly Input With Errors:**

The following assembly-source file contains multiple syntax errors all of which will not be found with a single execution of the assembler. Some errors need to be fixed in order for assembler to continue processing the file and find the remaining errors. This source file is not intended to have good runtime behavior, it is intended to expose syntax errors and verify error catching by the assembler.

;Example

Begin ORI

derka EQU Begin

RES 12

ADD 1,derka

LOL CCD Hi

LD 6,543(HI)

PTC ACC

MUL derka,LOL

RES LOL

AND LoL

Begin2 ORI 17

1bad SHR 1,2(3)

END BR 0,0

END BRS END

ACC NMD 9

END Begin

ADD 1,2

**Assembly Input Without Errors:**

The following assembly-source file contains no syntax errors and will assemble with a single execution of the assembler. After execution the listing-output file will contain a listing of all object code as well as all input records and the object-output file will contain the correct executable object code of the given assembly-source file.

atest1 ORI program header

X NMD 22 M[1] = 22

Y NMD -17 M[2] = -17

Z NMD 5 M[3] = 5

RES 6 skip 6 words

LD 0,Y R0 <-- (-17)

MUL 0,Z R0 <-- (-17)\*5 = (-85)

ST 0,4 M[4] <-- (-85)

LD 1,1 R1 <-- 22

DIV 1,3 R1 <-- 22/5 = 4

ST 1,5 M[5] = 4

LD 2,1 R2 <-- 22

AND 2,3 R2 <-- 0b10110 ^ 0b00101 = 0b00100 = 4

ST 2,6 M[6] <-- 4

LD 3,2 R3 <-- -17

OR 3,3 R3 <-- 0xFFFEF v 0x5 = 0xFFFEF = -17

ST 3,7 M[7] <-- -17

LD 1,Y R1 <-- -17

SHR 1,5 R1 <-- 0xFFFEF shf.rt.5 = 0xFFFFF = -1

ST 1,8 M[8] <-- -1

LD 0,X R0 <-- 22

SHL 0,3 R0 <-- 0b10110 shf.lf.3 = 0b10110000 = 176

ST 0,9 M[9] <-- 176

SHR 0,3(2) R0 <-- 0b10110000 shf.rt.7 = 0b1 = 1

; Print "Done" (next 9 instructions)

LDI 2,C use R2 as a base address for char data

LD 1,0(2) print from R1

IO 3,0(1) print 'D'

SHL 1,8 bring 'o' into position

PTC 1 print 'o'

LD 3,1(2) print from R3 this time ("ne" at offset 1)

PTC 3 print 'n'

SHL 3,8 bring 'e' into position

PTC 3 print 'e'

BR 0,0 halt quietly

C CCD Do M[39] contains "Do"

CCD ne M[40] contains "ne"

END 10 Execution is to start at Addr 0xA (= 10)