Pullet16 Assembler User-Manual

By: Alec Baker, James Cochran, James Thompson, Ethan Weaver, and Chris Walls

How Our Pullet16 Assembler Works?

An assembler will take a given programming language (in this case Assembly Code) and convert it into machine code. This assembler requires three file names as parameters that have to be included when first running the program. These parameters are the source file for the assembly code to be translated, the out file for the machine code, and the log file. The second two files will be created when the assembler program is ran. The file for the second parameter will be machine code that is generated from the Assembly source file. It can be thought of as being analogous to Windows '.exe' files. These are essentially files that the computer can execute. It will have two file types, a '.txt' and a '.bin' file. The log file records what is happening in the background when running the Assembler. The Assembler will also catch various errors that are in the Assembly source file. If an error is found, no machine code will be generated. Instead, the errors that were detected will be indicated in the log file.

The Pullet16 assembler is a two-pass assembler. A two-pass assembler scans/reads the source code twice. During the first time it scans (i.e. pass one), it determines how long the machine code will be and finds all of the addresses of the symbols and creates a table of the symbols. The second pass of the assembler uses the symbol table to generate the most efficient machine code possible.

Pass One

Pass one of the assembler takes in the scanner of the source code. With the scanner, it first loops through the source code file, parses the codeline string into definitive codeline attributes according to the length of the line, creates the codeline object, and stores the codeline into the ‘codelines_’ vector. While this is occuring, the symbol table is set up based on the symbols included in the source code. Several errors are also checked during pass one. Invalid symbols are caught and documented and also checks if a symbol is multiply defined documenting the error. It also checks if the ORG to ‘pc_’ value is accessible and checks if the DS command has proper allocation.

Pass Two

Pass two of the assembler does various error checking and generates the final machine code. It does so by looping through the ‘codelines_’ vector that was set up in pass one. During each iteration, the mnemonic or instruction for the codeline is validated. Any symbol used is checked to make sure it is defined. The address is validated if indirect addressing, in which the contents of a memory location are taken to be not the data but the address at which the data is to be found. Along with pass one, pass two checks if the ORG to ‘pc_’ value is accessible and checks if the DS command has proper allocation. If no errors were found during pass two, the machine code is then generated.

Errors the Pullet16 Assembler Catches

Pass One Error Catching

- Checks if each symbol has been defined before using the symbol.
    - Iterates through ‘symboltable_’ and, if the symbol was not found, set error messages.
- Checks if each symbol is a valid symbol.
    - When the symbol is constructed in pass one, ‘CheckInvalid()’ within the constructor for symbol test that the symbol is between 1-3, begins with an alpha character, makes sure all characters are alphanumeric, and middle character cannot be a space.
- Checks if the source code contains an ‘END’ statement.
    - If it finds an END statement, it flags a boolean variable to true. If it does not, it keeps the boolean false and prints an error in ‘PrintMachineCode()’.

Pass Two Error Catching

- Checks if the mnemonic exists.
    - If a mnemonic is passed in that is not handled by the assembler, an error flag is set and the pc is incremented. A message is also sent to the log file indicating that it was not a valid mnemonic.
- Checks if a symbol has been defined more than once.
    - Iterates through symboltable. If the symbol was not found, it will set error messages and add one to the program counter.
- Catches if addressing is ONLY if it is direct or indirect.
    - An address is only allowed to be direct or indirect. These are represented by a “ ” and a “*” respectively. If it reads in a character that is neither of these, an error is flagged and a message is sent to the log file.
- Checks the validity of hex operands.
    - Hex operands must be 5 characters long. It should begin with either a “+” or “-” following this should be some combinations of “0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F”. If the hex operand is invalid, an error is flagged, and a message is sent to the log file.

Error Catching Contained in Both Passes:

- Checks if ORG and DS stay within addresses 0 and 4096. If it goes outside of the address bounds, then it flags an error and sends an address out-of-bounds message to the log file.
    - Passes the code line into ‘SetNewPC()’, setting ‘pc_’ to hex value only if it’s between 0 and ‘kMaxMemory’ (which is equivalent to 4096).
- Checks if the PC stays within addresses 0 and 4096.

- In pass one, if the pc reaches 4096 the assembler will stop reading in input and go straight to pass two. In pass two if the pc somehow gets out of bounds, flags an error message for that instance of codeline and adds dummy code for that line of code into the machine code map.

How To Use

In order to use the Pullet16 assembler, you must first have a '.txt' file containing your program's source code, as the assembler needs only the source code to generate the machine code. To run the program;

./Aprog infilename.txt outfilename logfilename.txt

where 'infilename' is the source code, 'outfilename' is an output file and the assembler produces a '.bin' and '.txt' version output of machine code.

Source Code Format

If input source code does not follow these rules it will result in an error and will not print the respected machine code for the source code.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
l	l	l	b	m	m	m	b	a	b	s	s	s	b	pm	h	h	h	h	b	c

lll = optional label, left justified, which is alphanumeric beginning with an alpha character

b = blank space

mmm = the mnemonic opcode

a = blank (direct addressing) or asterisk (indirect addressing)

sss = optional symbolic operand, same rules as for labels

pm = plus sign or minus sign for the optional hex operand

hhhh = the four hex digits of the optional hex operand

c = comment (and continuing beyond column 21)

Mnemonics:

BAN, SUB, STC, AND, ADD, LD, BR, STP, RD, WRT, HEX, END, ORG, DS.