Assignments for CoSc 330, Computer Systems

J. Stanley Warford

November 2, 2021

Chapter, Section, and Exercise numbers in these assignments refer to the text for this course, *Computer Systems*, Fifth edition, J. Stanley Warford, Jones and Bartlett, 2017.

As the author of our text, if you purchase it new I will personally refund on your request 15% of the retail price you paid. (Current Pepperdine students only.)

Some exercises are to be typed and handed in electronically as a PDF file. Other exercises are to be hand written on paper. Problems are programs whose source files are always to be handed in electronically.

1. Study Sections 1.1, 1.2, 1.3, Chapter 2, Sections 3.1, 3.2.

2. Do Exercises 1.10, 1.13, 1.19, 2.2(c), 2.7.
   For Exercise 2.2(c) part (5), you only need to draw a snapshot of the stack frame at one point in time. You do not need to draw the entire sequence of stack frames for every call and return.

   For Exercise 2.7, see Figure 2.32 for a similar program.

   Hand in the exercises written on paper.

3. Do Problem 2.14.
   Write your program in C. Prompt the user exactly as shown on page 113. Name your program `Prob0214.c`. Note the uppercase `P`. See this link for instructions on how to set up C and complete the assignment.

   Hand in the `Prob0214.c` file electronically per the instructions for your course.

1. Study Sections 3.3, 3.4, 3.6

2. Do Exercises 3.7, 3.12, 3.14, 3.16, 3.18, 3.19, 3.21

   Type your solutions in a text editor and save it in a PDF file named `a02written.pdf`. Note the lowercase `a`. You cannot simply change the extension of your file name from `.docx`, or `.rtf`, or `.txt` to `.pdf`. You must *export* your document as a PDF file.

   Hand in the file electronically per the instructions for your course.

3. Do Problem 3.57.
   Although the problem says to write your program in C, write it in Java with IntelliJ by completing the code in `Prob0357Main.java`, which has the user interface already in place. Convert the eight characters in `line` to eight integers in the `binNum` array. Verify that each bit entered by the user is 0 or 1 and output an error message if it is not. If the user enters

```
11111100
```

   the output to the console should be

```
11111101
11111110
11111111
00000000
00000001
00000010
00000011
00000100
00000101
00000110
```

   Here is a link to Oracle's Java documentation for the `String` class. It lists the methods you can use for the `line` variable. See the `charAt()` method for extracting an individual character from `line`.

   Here is a link to documentation for the `PrintStream` class. You must use the formatting capabilities of the `System.out.printf()` method of this class even though it may no seem necessary. The `printf()` method will be necessary in later projects so you should start using it now. Also, you will be learning how `printf()` works in C because the `printf()` in Java has identical behavior.

   RESTRICTION: Do not use the Java function `parseInt()` because it does automatically what your program is supposed to do.

   Name your Java package `prob0357`. Note the lowercase `p`. The first line of your source file must be `package prob0357;`. Name your IntelliJ project `Prob0357` and the class that has the main program as `Prob0357Main`. Note the uppercase `P`.

   For your convenience, here is a IntelliJ project set up according to the above specifications.

   `https://cslab.pepperdine.edu/warford/cosc330/Prob0357.zip`

   See this link for instructions on how to set up Java and complete the assignment.

   Hand in the `Prob0357.jar` file electronically per the instructions for your course.

1. Study Sections 4.1, 4.2.

2. Do Exercises 3.23, 3.26, 3.28, 3.33, 3.35, 3.37, 4.4.
   For 3.23: Don't forget to convert the shifted values back to decimal. With ASL, show the effect on the NZVC bits. With ASR show the effect on the NZC bits. (ASR does not affect the V bit.)

   For these exercises, type your solutions in a text editor and save it in a file named `a03written.pdf`. Note the lowercase `a`. You cannot simply change the extension of your file name from `.docx`, or `.rtf`, or `.txt` to `.pdf`. You must *export* your document as a PDF file.

   Hand in the file electronically per the instructions for your course.

3. Do Problem 3.61.
   Write your program in Java with IntelliJ by completing the code of `Prob0361Main.java`, which has the user interface already in place. Convert the eight characters in `line` to eight integers in the `binNum` array. Verify that each bit entered by the user is 0 or 1 and output an error message if it is not. If the user enters

   ```
   11111101
   ```

   the output to the console should be

   ```
   11111101 (bin) = -3 (dec)
   ```

   RESTRICTION: Do not use the Java function `parseInt()` because it does what your program is supposed to do.

   Name your Java package `prob0361`. Note the lowercase `p`. The first line of your source file must be `package prob0361;`. Name your IntelliJ project `Prob0361` and the class that has the main program as `Prob0361Main`. Note the uppercase `P`. For your convenience, here is an IntelliJ project set up according to the above specifications.

   `https://cslab.pepperdine.edu/warford/cosc330/Prob0361.zip`

   Export the source file in a JAR file named `Prob0361.jar`. For this problem and all future Java problems, be sure to include the `.java` source files in the `.jar` file as described in Assignment 2. Hand in `Prob0361.jar` electronically per the instructions for your course.

1. Study Sections 4.3, 4.4.

2. Do Exercises 4.1, 4.2, 4.6
   For these exercises, type your solutions in a text editor and save it in a file named `a04written.pdf`. Note the lowercase `a`. Hand in the file electronically per the instructions for your course.

3. Do Problem 4.11.
   Download the Pep/9 system and use it to write your machine language program.

   `https://ComputerSystemsBook.com`

   You must use direct addressing.

   Name your program `Prob0411.pepo`. Note the uppercase `P`. In general, pep8 assembly language source files end in `.pep`, machine language object files end in `.pepo`, and assembler listing files end in `.pepl`. Hand in the `.pepo` file electronically per the instructions for your course.

1. Study Section 5.1.
2. Do Exercise 4.9.
   For this exercise, type your solutions in a text editor and save it in a file named `a05written.pdf`. Note the lowercase `a`. Hand in the file electronically per the instructions for your course.
3. Do Problem 4.14.
   Store the -3 in hexadecimal. Do not use the subtract, negate, or invert instructions. You must use direct addressing.

   Name your program `Prob0414.pepo` and hand it in electronically per the instructions for your course.

1. Study Section 5.2.
2. Do Exercises 5.2, 5.4, 5.6.
   For these exercises, type your solutions in a text editor and save it in a file named `a06written.pdf`. Hand in the file electronically per the instructions for your course.
3. Do Problem 5.20.
   Name your program `Prob0520.pep`. Hand in the `.pep` source file (*not* the `.pepo` file) electronically per the instructions for your course.

1. Do Exercises 5.11, 5.12.
   For these exercises, type your solutions in a text editor and save it in a file named `a07written.pdf`. Hand in the file electronically per the instructions for your course.

2. Do Problem 5.21.
   Name your program `Prob0521.pep`. Hand in the `.pep` source file (*not* the `.pepo` file) electronically per the instructions for your course.

3. Do Problem 5.22.
   Name your program `Prob0522.pep`. Hand in the `.pep` source file (*not* the `.pepo` file) electronically per the instructions for your course.

IMPORTANT: From now on, trace tags for all variables and parameters are required for full credit.

1. Study Sections 5.3, 5.4
2. Do Exercises 5.14, 5.19.
   Hand in the exercises in a file named `a08written.pdf` electronically per the instructions for your course.
3. Do Problem 5.27.
   Name your program `Prob0527.pep`. Hand in the `.pep` source file (*not* the `.pepo` file) electronically per the instructions for your course.
4. Do Problem 5.29.
   Name your program `Prob0529.pep`. Hand in the `.pep` source file (*not* the `.pepo` file) electronically per the instructions for your course.

1. Study Section 7.1.
2. Do Exercises 7.1, 7.2, 7.4(b), 7.6(b, d, f), 7.7(b).
   Hand in the exercises written on paper.

1. Study Section 7.2.
2. Do Exercises 7.11, 7.12, 7.13.
   Hint: In 7.13, you should use only four states in all parts of the exercise. Also note that your finite state machine *must* be deterministic.

   Hand in the exercises written on paper.

1. Study Section 7.3.

2. Do Problem 7.15.
   Name your Java package `prob0715`. Note the lowercase `p`. The first line of your source file must be `package prob0715;`. Name your IntelliJ project `Prob0715` and the class that has the main program as `Prob0715Main`. For your convenience, here is an IntelliJ project set up according to the above specifications with the code from Figure 7.28 for you to modify.

   `https://cslab.pepperdine.edu/warford/cosc330/Prob0715.zip`

   Export the source file in a JAR file named `Prob0715.jar`. Hand in the `.jar` file electronically per the instructions for your course.

1. Do Problem 7.18.
   If the user enters

   ```
   0d
   ```

   the output to the console should be

   ```
   0d is valid: 13
   ```

   If the user enters

   ```
   1D8G
   ```

   the output to the console should be

   ```
   1D8G is not valid.
   ```

   To guarantee that the hexadecimal value does not occupy more than two bytes, check that the *decimal* value is not more than 65535. Note that 00000A is a valid hexadecimal constant even though it is longer than four characters.

   You are not allowed to use the Java method `parseInt()` as the purpose of the program is to implement a parser yourself.

   Name your Java package `prob0718`. Note the lowercase `p`. The first line of your source file must be `package prob0718;`. Name your IntelliJ project `Prob0718` and the class that has the main program as `Prob0718Main`. For your convenience, here is an IntelliJ project set up according to the above specifications for you to modify.

   `https://cslab.pepperdine.edu/warford/cosc330/Prob0718.zip`

   Export the source file in a JAR file named `Prob0718.jar`. Hand in the `.jar` file electronically per the instructions for your course.

1. Study the Java program in Figures 7.34 – 36.
2. Begin Assignment 14.

1. Do Problem 7.19(a).

   Note that the problem description continues on page 462.

   You must store `hexValue` as an integer as in the previous programming assignment.

   Your lexical analyzer should recognize all the valid tokens and reject all the invalid tokens. The list of valid tokens in the problem description is not exhaustive. There are many more strings of characters that should be accepted.

   Here is a list of invalid tokens that your lexical analyzer should handle properly.

   Tokenizer input

```
alpha$beta
0xQ
0x12345
-32769
65536
<empty line>
```

   Tokenizer output

```
Identifier = alpha
Syntax error
Syntax error
Syntax error
Syntax error
Syntax error
Empty token
```

   There are many other strings of characters that should not be accepted. No string of input characters should crash your program.

   Restriction: For all phases of this project you may not use any Java library functions that parse strings, for example `Integer.parseInt()`. That would defeat the purpose of the program, which is to parse the source.

   Name your Java package `prob0719`. Note the lowercase `p`. Create a separate source file for each Java class. See this link for instructions on how to create a new class with IntelliJ.

   Name your IntelliJ project `Prob0719` and the class that has the main program as `Prob0719Main` in a file named `Prob0719Main.java`. For your convenience, here is a IntelliJ project set up according to the above specifications for you to modify.

   `https://cslab.pepperdine.edu/warford/cosc330/Prob0719.zip`

   Export the source file in a JAR file named `Prob0719.jar`. Hand in the `.jar` file electronically per the instructions for your course.

1. Study Section 7.4.

2. Study the Java Map hash table example.

   `https://cslab.pepperdine.edu/warford/cosc330/MapDemo.zip`

3. Study the Java version of the program in Figures 7.38 – 48.

4. Do Problem 7.19(b). Note that Problem 7.19(c) is the specification for the second milestone of your project on which this assignment is based.

   There is one additional requirement for Problems 7.19(b) and (c). The branch instructions that you will implement are `BR`, `BRLT`, `BREQ`, and `BRLE`. The legal addressing modes for these instructions are immediate and indexed as specified by the addressing-a field shown in Figure 4.8 (page 191). It is legal to omit the addressing mode for these instructions in the source code. If the addressing mode is omitted then immediate addressing is the default. Your finite state machine for the parser must include the possibility that a legal program might include a branch instruction without an addressing mode.

   Hand in the exercise on paper or in a PDF file named *xx*`a15written.pdf` where *xx* is your two-digit ID number.

1. Study Sections 6.1, 6.2.
2. Do Problem 6.12.
   Name your program `Prob0612.pep`. Hand in the `.pep` source file (*not* the `.pepo` file) electronically per the instructions for your course.
3. Do Problem 6.13.
   Name your program `Prob0613.pep`. Hand in the `.pep` source file (*not* the `.pepo` file) electronically per the instructions for your course.

1. Study Section 6.3.
2. Do Problem 6.16.
   Name your program `Prob0616.pep`. Hand in the `.pep` source file (*not* the `.pepo` file) electronically per the instructions for your course.

1. Study Section 6.3.
2. Do Problem 6.17.
   Name your program `Prob0617.pep`. Hand in the `.pep` source file electronically per the instructions for your course.

1. Do Problem 7.19(c).

   The sample code in the text has an abstract class named `AArg` for abstract argument with one method named `generateCode()`. Only this single method is needed for the example language to produce both the listing and the object code.

   In Pep/9 assembly language, the translator will need two methods, `generateListing()`, which returns a string value of the object to be output for the listing, and `getIntValue()`, which returns the integer value to be used for the code generator. In this parsing phase of your project, you only need to implement `generateListing()`.

   Here is a sample run with some valid input.

   Parser input

```
br 0x17, i
.BLOck 0x3
.block 1
deci 0x03,d
LDWA 0x0003,d
adda 0x9, i

deco 0xaB, s
stop
.end
```

   Parser output

```
Program Listing
BR      0x0017
.BLOCK  0x0003
.BLOCK  1
DECI    0x0003,d
LDWA    0x0003,d
ADDA    0x0009,i

DECO    0x00AB,s
STOP
.END
```

   Here is a sample run with some invalid input.

   Parser input

```
brr  0x7,  i
.BLOck   0xG
deci  0x03, e
```

Parser output

```
ERROR: Illegal mnemonic
ERROR: Illegal hex constant
ERROR: Illegal addressing mode
ERROR: Missing .END sentinel
```

Your program will be tested more extensively than the short examples above. Be sure to test it thoroughly.

Restriction: For all phases of this project you may not use any Java library functions that parse strings, for example `Integer.parseInt()`. That would defeat the purpose of the program, which is to parse the source.

Restriction: The code in `actionPerformed()` for this milestone is different from the code in the first milestone. For this assignment *do not* comment out the old code from the first milestone and replace it with the code from the new assignment. Instead, remove the old code from `actionPerformed()` *completely*. The automated testing system detects the occurrence of the `inBuffer` constructor and literally changes your source code to include the unit tests for the assignment. The system cannot detect whether any code is commented. So, if you include your code from the first milestone in your source, even if it is commented out, the system will automatically alter it such that your program will not compile.

Name your Java package `prob0719`. Note the lowercase `p`. Create a separate source file for each Java class. The first line of your source files must be `package prob0719;`. Name your IntelliJ project `Prob0719` and the class that has the main program as `Prob0719Main` in a file named `Prob0719Main.java`.

Export the source files in a JAR file named `Prob0719.jar`. Hand in the `.jar` file.

1. Study Section 6.3.
2. Do Problem 6.18.
   The test

   ```
   if (mpr % 2 == 1)
   ```

   checks if `mpr` is odd, which it is when its least significant bit is 1. You can test that bit by ANDing `mpr` with the mask `0x0001`, which sets all the bits except the rightmost bit to zero, and then comparing the result to zero with `BREQ`.

   Name your program `Prob0618.pep`. Hand in the `.pep` source file electronically per the instructions for your course.
3. Do Problem 6.22.
   Name your program `Prob0622.pep`. Hand in the `.pep` source file electronically per the instructions for your course. Also resubmit your program `Prob0214.c` from Assignment 1. Your assembly language program `Prob0622.pep` must be an exact translation of your C program with symbols in assembly language corresponding to variable names in C. Input the number of disks, the source peg, and the destination peg in that order.

1. Study Section 6.4.
   Skip the section, Translating Arrays Passed as Parameters.

2. Do Problem 6.24.
   Name your program `Prob0624.pep`. Hand in the `.pep` source file electronically per the instructions for your course.

3. Do Problem 6.25.
   Name your program `Prob0625.pep`. Hand in the `.pep` source file electronically per the instructions for your course.

1. Do Problem 7.19(d).

   Restriction: For all phases of this project you may not use any Java library functions that parse strings, for example `Integer.parseInt()`. That would defeat the purpose of the program, which is to parse the source.

   Additional restriction: For this phase, you may not use any print statements like `System.out.printf()` in your `generateCode()` methods. All printing must be done in method `translate()`. The purpose of `generateCode()` is to return the string to be output. Use the formatting features of `String.format()` to compose the string to be returned by `generateCode()`.

   Name your Java package `prob0719`. Note the lowercase `p`. Create a separate source file for each Java class. The first line of your source files must be `package prob0719;`. Name your IntelliJ project `Prob0719` and the class that has the main program as `Prob0719Main` in a file named `Prob0719Main.java`.

   Export the source files in a JAR file named `Prob0719.jar`. Hand in the `.jar` file.

1. Do Problem 6.27.

   Name your program `Prob0627.pep`. Hand in the `.pep` source file electronically per the instructions for your course.

2. Do Problem 6.28.

   Name your program `Prob0628.pep`. Hand in the `.pep` source file electronically per the instructions for your course.

1. Do Problem 6.33.

   To save some typing, you can get the source code for Figures 6.40 from the Help system in Pep/9. Name your program `Prob0633.pep`. Hand in the `.pep` source file electronically per the instructions for your course.

2. Do Problem 6.39.

   To save some typing, you can get the source code for Figures 6.48 from the Help system in Pep/9. Name your program `Prob0639.pep`.

   Note that there is a typo in the text. The C code `printf("\n")` should be the first line of code inserted, not the last, as follows:

```
printf("\n");
first2 = 0; p2 = 0;
for (p = first; p != 0; p = p->next) {
   p2 = first2;
   first2 = (struct node *) malloc(sizeof (struct node));
   first2->data = p->data;
   first2->next = p2;
}
for (p2 = first2; p2 != 0; p2 = p2->next) {
   printf("%d ", p2->data);
}
```

   So, if the input is

```
10 20 30 40 50 -9999
```

   the output should be

```
50 40 30 20 10
10 20 30 40 50
```

   Hand in the `.pep` source file electronically per the instructions for your course.