Project Descriptions:

In this project, we are going to develop three programs as specified below using PC-SPIM (MIPS assembly instructions):

(1) IF ~ THEN ~ ELSE structure
(2) WHILE structure
(3) floating-point fraction numbers

The descriptions and requirements for each control structure are as follow:

(1) IF ~ THEN structure

Illinois’s basic speeding law prohibits driving at a speed that is “greater than is reasonable and proper with regard to traffic conditions and the use of the highway, or endangers the safety of any person or property.” In other words, motorists must always drive at a safe speed. What a safe speed is will depend on the circumstances (625 Ill. Comp. Stat. Ann. § 5/11-601 (2017)).

When your program starts, your program prompts a user to enter two numbers. The first number is your current driving speed in MPH (1 to 200 MPH, where ‘1’ and ‘200’ are included). The second number is the absolute speed limit specified for the road you are currently running on (15 to 70 in MPH, where ‘15’ and ‘70’ are included). The two numbers should be prompted by your program, one at a time (your current driving speed, followed by the speed limit). When a user makes any invalid input, your program should detect it and repeat the prompt to a user. When a user made an invalid input to the speed limit (the second input), while the user had made a valid input to the first (i.e., your current driving speed), your program should prompt only for the second input, but never for the first input. After the two numbers are successfully entered, your program should determine the penalty you may receive based on the following criteria (730 Ill. Comp. Stat. §§ 5/5-4.5-55, 5/5-4.5-60 (2017)):

- Exceeding the speed limit by 1-20 MPH over the limit: “$120 fine”
- Exceeding the speed limit by 21-25 MPH over the limit: “$140 fine”
- Exceeding the speed limit by 26-34 MPH over the limit: “Class B misdemeanor and carries up to six months in jail and a maximum $1,500 in fines”.
- Exceeding the speed limit by 35 MPH or more over the limit: “Class A misdemeanor and carries up to one year in jail and a maximum $2,500 in fines”.

For example, if you are driving at 72MPH on a highway with 55MPH limit, a message: “you may receive a $120 fine” should be displayed on your local monitor. If your current speed is less than the speed limit, a message “you are a safe driver!” should be displayed. For each invalid input, “You made an invalid input for X. Enter a valid input for X.”, where X is:

For the first input: “your current driving speed”
For the second input: “the speed limit for the road”

For Program #1, the following MIPS (PC-SPIM) floating-point instructions and system calls are suggested:*1:

- li – load a constant signed two’s complement integer to a register
- la – load a constant unsigned integer (usually memory address) to a register
- move – copy the content of a register to another register
- sub – subtract the content of a register from the content of another register and save the result to a destination register
- j – let the processor jump to the instruction specified by “jump destination address”
- beq, bgt, blt, and etc. – conditional branch instructions for controlling flow of the program
- PC-SPIM system call #4 – output (print) a pre-declared text string to the local display
- PC-SPIM system call #5 – input a signed (negative and positive) integer from the keyboard

*1: a reference to those instructions and system calls are available in Appendix-B in the required textbook.

Requirements:

a. The above procedure must be implemented using IF-THEN structures implemented in MIPS assembly instructions.

b. Each input must be tested to confirm its validity

c. After the first input (a course score) is correctly made, the first input should never be repeated when a user makes an invalid input for the second input (i.e., “the speed limit of the road”).

d. Your program should output a correct message (no garbage output, and etc.). See the following screen snapshot of the program.

e. Your program should terminate without any error or a warning message.
(2) WHILE structure

This program calculates a product of two numbers without using assembly instruction for multiplication (e.g., “mul”). When your program starts, your program prompts a user to enter two numbers (each number should be larger than 0 and less than 51). The product of two numbers should be calculated by adding the first number as many times as the second number. When a user enters any invalid number, your program should detect it and repeat the prompt to a user.

<table>
<thead>
<tr>
<th>Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter the first number (0-50): 60</td>
</tr>
<tr>
<td>The number entered is not in 0-50. Enter another number.</td>
</tr>
<tr>
<td>Enter the first number (0-50): -16</td>
</tr>
<tr>
<td>The number entered is not in 0-50. Enter another number.</td>
</tr>
<tr>
<td>Enter the second number (0-50): 20</td>
</tr>
<tr>
<td>Enter the second number (0-50): 78</td>
</tr>
<tr>
<td>The number entered is not in 0-50. Enter another number.</td>
</tr>
<tr>
<td>Enter the second number (0-50): -6</td>
</tr>
<tr>
<td>The number entered is not in 0-50. Enter another number.</td>
</tr>
<tr>
<td>Enter the second number (0-50): 30</td>
</tr>
<tr>
<td>600</td>
</tr>
</tbody>
</table>

Requirements:

a. The above program should be implemented using a loop structure implemented in MIPS “add” assembly instructions, but not using any multiplication instruction.

b. Each input must be tested to confirm it is a valid input.

c. If an input number is less than 0, display an error message, “the entered number is less than 0”. Then, prompt a user to make another input.

d. If an input number is larger than 50, display an error message, “the entered number is larger than 50”. Then, prompt a user to make another input.

e. After the first input is correctly made, the first input should never be repeated when a user makes an invalid input for the second input.

f. Your program should correctly calculate the product of the two numbers by using only instruction for addition (no multiplication).

g. Your program should terminate without any error or a warning message.
(3) floating-point fraction numbers

Most processors handle fraction numbers using a different set of instructions (different from the instructions for integers), called “floating-point instructions”. The third program calculates the number of fixed-monthly-payments for loans using floating-point instructions in a MIPS R-3000 processor. One way to calculate the monthly interest for a loan is as follows:

For the following loan:

- Principal: $3,000.00
- Monthly payment: $100.00
- Annual interest rate: 6.72%

When the first (monthly) payment was made, its interest is calculated as follows:

\[
\text{The monthly interest} = (\text{principal}) \times (\text{annual interest rate}) \times (30 \text{ days})/(365 \text{ days})
\]
\[
= 3,000 \times 0.0672 \times 0.08219
\]
\[
= 16.57
\]

Then, the amount for payment to the principal is calculated by subtracting the monthly interest from your monthly payment as follows:

\[
\text{The amount for payment to the principal} = 100.00 - 16.57
\]
\[
= 83.43
\]

After the first monthly payment, the principal will be:

\[
\text{The remaining principal} = 3,000 - 83.43
\]
\[
= 2,916.57
\]

The above process of the monthly payments will be repeated until the principal becomes $0.00. The number of repeats in the process is the number of monthly payments before we complete the payments for the loan. For example, the above loan needs a total of 33 monthly payments to complete, while the total interest paid is $289.02 (we pay a total of $3,289.02). Your program should prompt the following three inputs after it starts in the order described next. The first one is the principal ($100.00 to $1,000,000.00). The second input is the annual interest rate (0.005 (0.5%) to 0.399 (39.9%)). The third input is monthly payment amount ($1.00 to $2,000.00). The check to see if the monthly payment does not exceed the principal is NOT necessary (to simplify this project). Just like the other two programs in this project, any invalid input should be detected and they should be repeated until a valid input is made. Any valid input that has been made should not be re-prompted when an invalid input is made for other inputs.

Sample outputs for a case (principal = $3,000.00, monthly payment = $100.00, and 6.72% annual interest rate):
<table>
<thead>
<tr>
<th>Payments</th>
<th>Principal</th>
<th>Annual Interest Rate</th>
<th>Monthly Payment</th>
<th>Monthly Interest Rate</th>
<th>Monthly Interest</th>
<th>Net Pay to Principal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$3,000.00</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$16.57</td>
<td>$83.43</td>
</tr>
<tr>
<td>2</td>
<td>$2,916.57</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$16.11</td>
<td>$83.89</td>
</tr>
<tr>
<td>3</td>
<td>$2,832.68</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$15.65</td>
<td>$84.35</td>
</tr>
<tr>
<td>4</td>
<td>$2,748.32</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$15.18</td>
<td>$84.82</td>
</tr>
<tr>
<td>5</td>
<td>$2,663.50</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$14.71</td>
<td>$85.29</td>
</tr>
<tr>
<td>6</td>
<td>$2,578.22</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$14.24</td>
<td>$85.76</td>
</tr>
<tr>
<td>7</td>
<td>$2,492.46</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$13.77</td>
<td>$86.23</td>
</tr>
<tr>
<td>8</td>
<td>$2,406.22</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$13.29</td>
<td>$86.71</td>
</tr>
<tr>
<td>9</td>
<td>$2,319.51</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$12.81</td>
<td>$87.19</td>
</tr>
<tr>
<td>10</td>
<td>$2,232.32</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$12.33</td>
<td>$87.67</td>
</tr>
<tr>
<td>11</td>
<td>$2,144.65</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$11.85</td>
<td>$88.15</td>
</tr>
<tr>
<td>12</td>
<td>$2,056.50</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$11.36</td>
<td>$88.64</td>
</tr>
<tr>
<td>13</td>
<td>$1,967.86</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$10.87</td>
<td>$89.13</td>
</tr>
<tr>
<td>14</td>
<td>$1,878.73</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$10.38</td>
<td>$89.62</td>
</tr>
<tr>
<td>15</td>
<td>$1,789.10</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$9.88</td>
<td>$90.12</td>
</tr>
<tr>
<td>16</td>
<td>$1,698.99</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$9.38</td>
<td>$90.62</td>
</tr>
<tr>
<td>17</td>
<td>$1,608.37</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$8.88</td>
<td>$91.12</td>
</tr>
<tr>
<td>18</td>
<td>$1,517.25</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$8.38</td>
<td>$91.62</td>
</tr>
<tr>
<td>19</td>
<td>$1,425.63</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$7.87</td>
<td>$92.13</td>
</tr>
<tr>
<td>20</td>
<td>$1,333.51</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$7.37</td>
<td>$92.63</td>
</tr>
<tr>
<td>21</td>
<td>$1,240.87</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$6.85</td>
<td>$93.15</td>
</tr>
<tr>
<td>22</td>
<td>$1,147.73</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$6.34</td>
<td>$93.66</td>
</tr>
<tr>
<td>23</td>
<td>$1,054.07</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$5.82</td>
<td>$94.18</td>
</tr>
<tr>
<td>24</td>
<td>$959.89</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$5.30</td>
<td>$94.70</td>
</tr>
<tr>
<td>25</td>
<td>$865.19</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$4.78</td>
<td>$95.22</td>
</tr>
<tr>
<td>26</td>
<td>$769.97</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$4.25</td>
<td>$95.75</td>
</tr>
<tr>
<td>27</td>
<td>$674.22</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$3.72</td>
<td>$96.28</td>
</tr>
<tr>
<td>28</td>
<td>$577.94</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$3.19</td>
<td>$96.81</td>
</tr>
<tr>
<td>29</td>
<td>$481.14</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$2.66</td>
<td>$97.34</td>
</tr>
<tr>
<td>30</td>
<td>$383.79</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$2.12</td>
<td>$97.88</td>
</tr>
<tr>
<td>31</td>
<td>$285.91</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$1.58</td>
<td>$98.42</td>
</tr>
<tr>
<td>32</td>
<td>$187.49</td>
<td>6.720%</td>
<td>$100.00</td>
<td>0.5523%</td>
<td>$1.04</td>
<td>$98.96</td>
</tr>
<tr>
<td>33</td>
<td>$88.53</td>
<td>6.720%</td>
<td>$88.53</td>
<td>0.5523%</td>
<td>$0.49</td>
<td>$99.52</td>
</tr>
<tr>
<td>34</td>
<td>$0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$289.02</td>
</tr>
</tbody>
</table>

Some sample PC-SPIM assembly source code files will be provided for this program*2.

*2: “for loading a constant fraction number to a floating-point register” and “for copying the content of a floating-point register to another floating-point register”.

For Program #3, the following MIPS (PC-SPIM) floating-point instructions and system calls are suggested:

- add.s – single-precision floating-point addition
- sub.s – single-precision floating-point subtraction
- mul.s – single-precision floating-point multiplication
- lwcl – loading a single-precision floating-point constant from a memory address
- c.lt.s – floating-point numerical comparison of two floating-point registers
- bc1t – a conditional branch instruction used with c.lt.s
- mov.s – “move” for single-precision floating point registers
- PC-SPIM system call #2 – output (print) a fraction number to the local display
- PC-SPIM system call #6 – input a fraction number from the keyboard
Enter the principal in $ (100.00 - 1,000,000.00): 3000
Enter the annual interest rate (0.005 - 0.399): 0.0672
Enter the monthly payment amount in $ (1.00 - 2,000,000.00): 100

month 1: current principal = 3000.00000000
month 2: current principal = 2916.56956000
month 3: current principal = 2832.67822556
month 4: current principal = 2748.32348633
month 5: current principal = 2663.50292959
month 6: current principal = 2578.21386719
month 7: current principal = 2492.45385742
month 8: current principal = 2406.22021434
month 9: current principal = 2319.51025391
month 10: current principal = 2232.32120900
month 11: current principal = 2144.65087891
month 12: current principal = 2056.49609375
month 13: current principal = 1967.85449219
month 14: current principal = 1878.72326860
month 15: current principal = 1789.09731445
month 16: current principal = 1699.98120117
month 17: current principal = 1609.36459023
month 18: current principal = 1517.24829102
month 19: current principal = 1425.62829590
month 20: current principal = 1333.50231934
month 21: current principal = 1240.86743164
month 22: current principal = 1147.72094727
month 23: current principal = 1054.0605859
month 24: current principal = 960.98185394
month 25: current principal = 865.10341064
month 26: current principal = 769.96197510
month 27: current principal = 674.21459951
month 28: current principal = 577.93841553
month 29: current principal = 481.13046235
month 30: current principal = 383.7984180
month 31: current principal = 285.90756226
month 32: current principal = 187.49667900
month 33: current principal = 80.52220154

It will take 33 months to complete the loan.

Requirements:

a. The above program should be implemented using MIPS assembly instructions.

b. If an invalid input is made, an error message should be displayed and your program should prompt a valid input.

c. After the first input or the second is successfully made, the first or the second input should never be prompted, when an invalid input is made to the second or the third input.

d. The correct number of monthly payments should be displayed at the end of each run.

e. Your program should terminate normally (should not crash the system when your program leaves the system).

Submissions:

Your source code files (one *.asm source code file for each of the three programs) should be e-mailed to Dr. Fujinoki (hfujino@siue.edu) by 9:45 A.M. on July 1st.
• Extra-Credit (+4 points to your midterm exam score) Early-Submission*3 Deadline: All three *.asm source code files should be e-mailed to hfujino@siue.edu by 11:59:59 P.M., June 21st. Note that the +4 extra credit points are awarded only if all three programs are completed (no partial extra credit).

• Free-Feedback Early-Submission*3 Deadline: Your *.asm source code files should be e-mailed to hfujino@siue.edu 11:59:59 P.M., June 26th.

• Final Deadline: Your *.asm source code files should be e-mailed to hfujino@siue.edu 9:45 A.M., July 1st.

Note 1: Each program flow-control should be implemented as an independent program (each of you is expected to submit three *.asm source code files).

Note 2: In your program source code, include your name and student ID in your program header (at the beginning of your *.asm source code file).

Note 3: It is required to use PC-SPIM posted to the CS286-001 course homes (programs developed for different versions of PC-SPIM will NOT be accepted for grading).

Note 4: This programming project is an individual programming project (no collaboration to complete the programming project is allowed). Violation of this rule is considered academic dishonesty

Note 5: Academic dishonesty in Project #1 will at least result in 0 point for the entire programming project credit for this course (not only for Project #1). Serious academic dishonesty will result in a failing grade (‘F’ grade) for this course and the names of people will be reported to the CS department and the School of Engineering Dean’s Office.

Late Submission:

• Penalty of -10% will be given for every 24 hours after the due (i.e., -10% for a submission within the first 24 hours after the due).

• Submission more than 48 hours after the due will not be accepted.

*3: No “late submission” for early deadlines. The extra credit will be given only if all the three programs are completed by the extra-credit early submission deadline. No “partial extra credit” will be given for this extra credit opportunity.

Grading Criteria:

• Assembler error: If Dr. Fujinoki can not run your assembler source code file due to any assembler error: 0 ~ 20% of the credit for this assignment phase will be given.

• Run-time error: After your assembly source code file is successfully assembled to machine codes, if your program does not satisfy any requirement: 5% penalty for each minor problem. For any major failure, penalty depends on each such major problem.

• The weight of the three programs is: 25%, 25%, and 50% for program 1 (if-then), program 2 (while), and program 3 (floating-point) respectively.
Expectations when a question is asked:

1. Identify where (in your source code) the problem exists
2. Describe the symptom(s) of the problem
3. Describe how the problem happens (always happen, sometime happen, the condition(s) for the problem to happen, etc.)
4. Describe what you tried (to understand and/or solve the problem)
5. Stop by Dr. Fujinoki’s office (no question through e-mail).
6. Bring your source code (either hard copy or soft copy)