

CS 286-002 Computer Organization & Architecture

Fall 2026

Programming Project #1

Final Submission Due: 3:00 p.m., September 23, 2026

Project Descriptions:

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

- (1) IF ~ THEN ~ ELSE program-flow control structure
- (2) WHILE loop structure
- (3) Controlling program-flow using 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 (both in integers). 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 repeat 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, an error message, "You made an invalid input for X. Enter a valid input for X.", should be displayed, where X is:

For the first input: “your current driving speed”

For the second input: “the speed limit for the road”

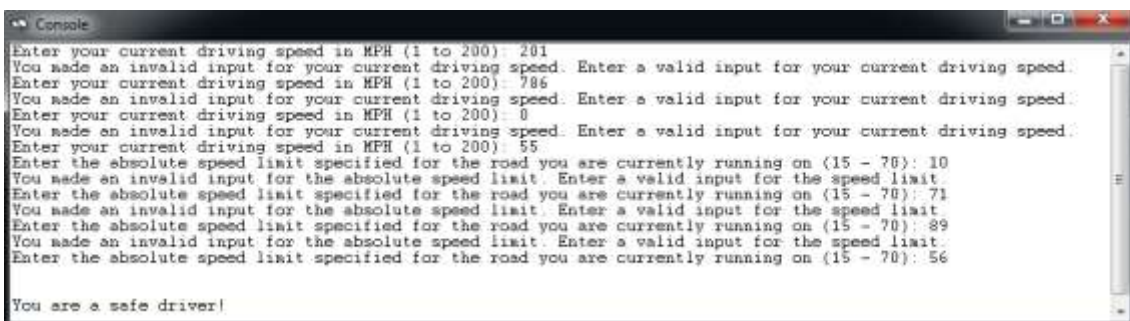
For Program #1, the following MIPS (PC-SPIM) instructions and system calls are suggested*1:

- 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
- add (addi) – add or 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 should be implemented using IF-THEN-ELSE structures implemented using MIPS assembly instructions.
- b. Each input should be tested to confirm its validity
- c. After the first input (“current diving speed”) is correctly made, the first input should never be repeated when a user makes an invalid input for the second input (“the speed limit”).
- 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.
- f. Your *.asm assembly source code file should be assembled and executed by PC-SPIM (but not by any other MIPS-R3000 emulator, such as qt-spim (no credit for violating this rule).



```
Console
Enter your current driving speed in MPH (1 to 200): 201
You made an invalid input for your current driving speed. Enter a valid input for your current driving speed.
Enter your current driving speed in MPH (1 to 200): 786
You made an invalid input for your current driving speed. Enter a valid input for your current driving speed.
Enter your current driving speed in MPH (1 to 200): 0
You made an invalid input for your current driving speed. Enter a valid input for your current driving speed.
Enter your current driving speed in MPH (1 to 200): 55
Enter the absolute speed limit specified for the road you are currently running on (15 - 70): 10
You made an invalid input for the absolute speed limit. Enter a valid input for the speed limit.
Enter the absolute speed limit specified for the road you are currently running on (15 - 70): 71
You made an invalid input for the absolute speed limit. Enter a valid input for the speed limit.
Enter the absolute speed limit specified for the road you are currently running on (15 - 70): 89
You made an invalid input for the absolute speed limit. Enter a valid input for the speed limit.
Enter the absolute speed limit specified for the road you are currently running on (15 - 70): 56
You are a safe driver!
```

(this section is continued to the next page)

```
Console
Enter your current driving speed in MPH (1 to 200): 201
You made an invalid input for your current driving speed. Enter a valid input for your current driving speed.
Enter your current driving speed in MPH (1 to 200): 988
You made an invalid input for your current driving speed. Enter a valid input for your current driving speed.
Enter your current driving speed in MPH (1 to 200): 202
You made an invalid input for your current driving speed. Enter a valid input for your current driving speed.
Enter your current driving speed in MPH (1 to 200): 0
You made an invalid input for your current driving speed. Enter a valid input for your current driving speed.
Enter your current driving speed in MPH (1 to 200): 203
You made an invalid input for your current driving speed. Enter a valid input for your current driving speed.
Enter your current driving speed in MPH (1 to 200): 65
Enter the absolute speed limit specified for the road you are currently running on (15 - 70): 10
You made an invalid input for the absolute speed limit. Enter a valid input for the speed limit.
Enter the absolute speed limit specified for the road you are currently running on (15 - 70): 71
You made an invalid input for the absolute speed limit. Enter a valid input for the speed limit.
Enter the absolute speed limit specified for the road you are currently running on (15 - 70): 14
You made an invalid input for the absolute speed limit. Enter a valid input for the speed limit.
Enter the absolute speed limit specified for the road you are currently running on (15 - 70): 22

Class A misdemeanor and carries up to one year in jail and a maximum $2,500 in fines
```

(2) WHILE structure

Develop a *.asm program (for PC-SPIM) that generates (calculates) a sequence of the numbers based on the following rules:

- Your program takes the following three parameters
 - The origin of a number sequence (as an integer) between 1 and 5 (i.e., no less than 1 and no greater than 5).
 - The addition factor (as an integer) between 2 and 7 (i.e., no less than 2 and no greater than 7).
 - The total number of the numbers to be generated (as an integer): between 3 and 30 (i.e., no less than 3 and no greater than 30).
- Then, your program generates the number sequence (and display them on the local monitor) by adding the addition factor to the origin number. A sample program output is shown in the figure below.

```
Console
Enter the origin of your number sequence: 3
Enter your multiple factor: 7
Enter the total number of the numbers: 15
3, 10, 17, 24, 31, 38, 45, 52, 59, 66, 73, 80, 87, 94, 101
Check-sum: 780
```

Requirements:

- (a) When an invalid input is detected, the input should be repeated until a valid (acceptable) input is made by a human user.
- (b) After the first input (“the origin number”) is correctly made, the first input should never be repeated when a user makes an invalid input for the second input (“the number of the numbers”).

- (c) All calculated numbers should be displayed correctly (no extra number(s), no missing number(s), and no any unnecessary outputs) as shown in the figure (note that there is no comma after the last number).
- (d) After all the numbers are generated and displayed, their “check sum” should be displayed (the correct check sum should be displayed). The check-sum is a total sum of all the numbers generated in a number sequence.
- (e) Your *.asm assembly source code file should be assembled and executed by PC-SPIM (but not by any other MIPS-R3000 emulator, such as “qt-spim” (no credit for violating this rule).

(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:

$$\begin{aligned}
 \text{The monthly interest} &= (\text{principal}) \times (\text{annual interest rate}) \times (30 \text{ days}) / (365 \text{ days}) \\
 &= \$3,000 \times 0.0672 \times 0.08219 \\
 &= \$16.57
 \end{aligned}$$

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

$$\begin{aligned}
 \text{The amount for payment to the principal} &= \$100.00 - \$16.57 \\
 &= \$83.43
 \end{aligned}$$

After the first monthly payment, the principal will be:

$$\begin{aligned}
 \text{The remaining principal} &= \$3,000 - \$83.43 \\
 &= \$2,916.57
 \end{aligned}$$

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 two other 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:

Payments	Principal	Annual Interest Rate	Monthly Payment	Monthly Interest Rate	Monthly Interest	Net Pay to Principal
1	\$3,000.00	6.720%	\$100.00	0.5523%	\$16.57	\$83.43
2	\$2,916.57	6.720%	\$100.00	0.5523%	\$16.11	\$83.89
3	\$2,832.68	6.720%	\$100.00	0.5523%	\$15.65	\$84.35
4	\$2,748.32	6.720%	\$100.00	0.5523%	\$15.18	\$84.82
5	\$2,663.50	6.720%	\$100.00	0.5523%	\$14.71	\$85.29
6	\$2,578.22	6.720%	\$100.00	0.5523%	\$14.24	\$85.76
7	\$2,492.46	6.720%	\$100.00	0.5523%	\$13.77	\$86.23
8	\$2,406.22	6.720%	\$100.00	0.5523%	\$13.29	\$86.71
9	\$2,319.51	6.720%	\$100.00	0.5523%	\$12.81	\$87.19
10	\$2,232.32	6.720%	\$100.00	0.5523%	\$12.33	\$87.67
11	\$2,144.65	6.720%	\$100.00	0.5523%	\$11.85	\$88.15
12	\$2,056.50	6.720%	\$100.00	0.5523%	\$11.36	\$88.64
13	\$1,967.86	6.720%	\$100.00	0.5523%	\$10.87	\$89.13
14	\$1,878.73	6.720%	\$100.00	0.5523%	\$10.38	\$89.62
15	\$1,789.10	6.720%	\$100.00	0.5523%	\$9.88	\$90.12
16	\$1,698.99	6.720%	\$100.00	0.5523%	\$9.38	\$90.62
17	\$1,608.37	6.720%	\$100.00	0.5523%	\$8.88	\$91.12
18	\$1,517.25	6.720%	\$100.00	0.5523%	\$8.38	\$91.62
19	\$1,425.63	6.720%	\$100.00	0.5523%	\$7.87	\$92.13
20	\$1,333.51	6.720%	\$100.00	0.5523%	\$7.37	\$92.63
21	\$1,240.87	6.720%	\$100.00	0.5523%	\$6.85	\$93.15
22	\$1,147.73	6.720%	\$100.00	0.5523%	\$6.34	\$93.66
23	\$1,054.07	6.720%	\$100.00	0.5523%	\$5.82	\$94.18
24	\$959.89	6.720%	\$100.00	0.5523%	\$5.30	\$94.70
25	\$865.19	6.720%	\$100.00	0.5523%	\$4.78	\$95.22
26	\$769.97	6.720%	\$100.00	0.5523%	\$4.25	\$95.75
27	\$674.22	6.720%	\$100.00	0.5523%	\$3.72	\$96.28
28	\$577.94	6.720%	\$100.00	0.5523%	\$3.19	\$96.81
29	\$481.14	6.720%	\$100.00	0.5523%	\$2.66	\$97.34
30	\$383.79	6.720%	\$100.00	0.5523%	\$2.12	\$97.88
31	\$285.91	6.720%	\$100.00	0.5523%	\$1.58	\$98.42
32	\$187.49	6.720%	\$100.00	0.5523%	\$1.04	\$98.96
33	\$88.53	6.720%	\$88.53	0.5523%	\$0.49	\$88.04
	\$0.00				\$289.02	

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
- lwc1 – 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

```
Console
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 = 2916.56958008
month 3: current principal = 2832.67822266
month 4: current principal = 2748.32348633
month 5: current principal = 2663.50292969
month 6: current principal = 2578.21386719
month 7: current principal = 2492.45385742
month 8: current principal = 2406.22021484
month 9: current principal = 2319.51025391
month 10: current principal = 2232.32128906
month 11: current principal = 2144.65087891
month 12: current principal = 2056.49609375
month 13: current principal = 1967.85449219
month 14: current principal = 1878.72326660
month 15: current principal = 1789.09973145
month 16: current principal = 1698.98120117
month 17: current principal = 1608.36499023
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 = 1054.06005859
month 24: current principal = 959.88183594
month 25: current principal = 865.18341064
month 26: current principal = 769.96197510
month 27: current principal = 674.21459961
month 28: current principal = 577.93841553
month 29: current principal = 481.13046265
month 30: current principal = 383.78784180
month 31: current principal = 285.90756226
month 32: current principal = 187.48667908
month 33: current principal = 88.52220154

It will take 33 months to complete the loan.
```

Requirements:

- The above program should be implemented using MIPS assembly instructions.
- If an invalid input is made, an error message should be displayed and your program should prompt a valid input.
- After the first or the second input 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.
- Outputs from your program should be as shown in the figure above (e.g., the correct number of monthly payments should be displayed at the end of each run).
- Your program should terminate normally (should not crash the system when your program is terminating).

- f. Your *.asm assembly source code file should be assembled and executed by PC-SPIM (but not by any other MIPS-R3000 emulator, such as “qt-spim” (no credit for violating this rule).

Submissions:

Your *.asm source code files (one *.asm source code file for each of the three programs) should be e-mailed to the course TA (Thota, Vamshi Anirudh (vthota@siue.edu) by 3:00 p.m. on September 23rd.

- For those who completed this project at least three days before the submission deadline, the TA will test your *.asm files for free (i.e., “free feedback”).

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 each of your program source code file, include your name and student ID in your program header (at the beginning of your *.asm source code files).

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 12 hours after the due (i.e., -10% for a submission within the first 12 hours after the due).
- Submission more than 24 hours after the due will not be accepted.

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 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%, 30%, and 45% for program 1 (if-then-else), program 2 (while), and program 3 (floating-point fraction numbers) respectively.

Expectations when a question is asked:

- (1) Dr. Fujinoki will not debug your programs. Please do not e-mail me your source code file(s) unless you are requested to do so.
- (2) If you have questions, stop by Dr. Fujinoki's office during the office hours (make sure to bring your *.asm source code files. Please avoid using e-mails for asking questions about the programming projects (there may be some exceptions).
- (3) To make your visits productive, the followings are suggested:
 - Identify where (in your source code) the problem exists
 - Describe the symptom(s) of the problem
 - Describe how the problem happens (always happen, sometime happen, the condition(s) for the problem to happen, etc.)
 - Describe what you tried (to understand and/or solve the problem)
- (4) Within 24 hours from the final submission due, Dr. Fujinoki will not respond to new questions (except any continued discussions for the questions you asked prior to the 24-hour time window).
- (5) Some questions about Project #1 will be asked in the midterm exam.