CMPE-250
Assembly and Embedded Programming
Spring 2021
Laboratory Exercise Eleven:
PWM and LEDs with Mixed Assembly Language and C

This exercise investigates pulse-width modulation (PWM) on the KL05Z board using the KL05 timer/pulse-width modulation module (TPM0). A PWM waveform from the KL05 is used to control the brightness of the red LED on the KL05Z board. The objective of this exercise is to set the brightness of the red LED to one of five levels. A C program is created to work with assembly language ROM lookup tables and previously written serial I/O driver, and the code is executed on the KL05Z board.

Prelab Work

Write C function `Init_TPM0()` to initialize the KL05 TPM0 to produce on channel 3 (`TPM0_CH3`) a 60-Hz PWM waveform ($16^2/_3$-ms period) and with a duty cycle of 0%, (i.e., 0 ms). Follow the class notes on the TPM posted on myCourses. You should use the `#define` names in the provided Exercise11_C_Stub.c file.

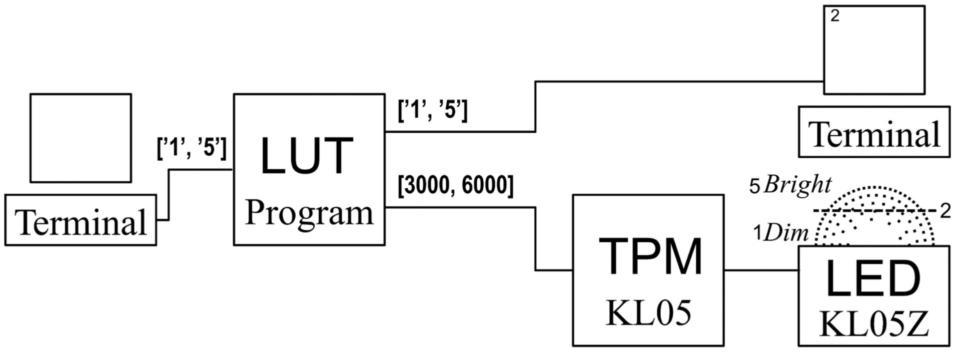
Application

For embedded systems programming, data often have to be transformed from one domain to another. For example, a temperature sensor may output "temperature" as an analog voltage that is proportional to the temperature measured, but a microcontroller must have data in digital format, (i.e., binary numbers), to use for calculations or to produce output for a user interface. As another example, a microcontroller program may represent a motor's speed as a digital value corresponding to desired revolutions per minute (RPM), but the physical motor needs a voltage proportional to the speed desired.

This exercise investigates the conversion of data between domains to control the brightness of an LED. The table that follows summarizes the transformation of data from user input to LED brightness level. Program code performs the conversions using calculations and/or look-up tables (LUTs). The figure that follows shows a block diagram of the LED brightness control system.

DATA DOMAINS FROM USER INPUT TO LED BRIGHTNESS LEVEL

Source	Data	Range	Destination
User (terminal keyboard)	ASCII	[‘1’, ‘5’]	Program
Program	Integer	[0, 4]	Program LUT for TPM0
Program LUT for TPM	16-Bit Integer	[49970, 0]	TPM0 on KL05
TPM0 on KL05	PWM	[100%, 0%] of $16^2/_3$ ms	Red LED on KL05Z
Red LED on KL05Z	Brightness	[1, 5]	User (LED brightness)



LED BRIGHTNESS CONTROL SYSTEM

Program Specification

Write a C program that uses the initialization code from prelab work to control the LED’s brightness using a position in the range [1, 5] typed by the user on the keyboard. The program must perform the following sequence of operations and must meet the requirements that follow.

1. Using assembly language subroutine `Init_UART0_IRQ`, initialize the KL05 for interrupt-based serial I/O with UART0 at 9600 baud using a format of one start bit, eight data bits, no parity, and one stop bit. (In the assembly language source code, also include UART0_ISR. Add and export the label UART0_IRQHandler for it to match the Keil C project’s default KL05 vector table.)
2. Using `Init_TPM0()` from prelab work, initialize the KL05 TPM0 to produce on channel 3 (TPM0_CH3) a 60-Hz PWM waveform ($16^2/_3$-ms period) with a duty period of 0%, (i.e., 0 ms), which should set the LED to brightness level 5, (i.e., always on).
3. Output the character string shown below to the terminal screen.
```
Type a number from 1 to 5: 
```
4. Get the character typed from the user.
5. If the character is not in the range [‘1’, ‘5’], repeat from step 4. Otherwise, echo the character to the screen, and move the cursor to the beginning of the next line, as shown below for user input of “1.”
```
Type a number from 1 to 5:  1

```

6. Using an assembly language ROM look-up table, scale the user's ASCII character input to a TPM PWM duty count value for a duty cycle between 100% and 0%. One way is first to convert the ASCII character input to an index between 0 and 4 (inclusive, which corresponds to LED brightness levels between 1 and 5, respectively) and then to use the index to access the look-up table of PWM duty period values shown in the myCourses notes on the TPM.
7. Write the duty count value to `TPM0_C3V`, (`TPM0->CONTROLS[3].CnV` in C), to set the LED brightness level requested by the user.
8. Output `LED brightness:` followed by the LED brightness level, (which is the ASCII character input by the user in step 4), and then advance the cursor to the beginning of the second line down, as shown below.

```
Type a number from 1 to 5:  1
           LED brightness:  1


```

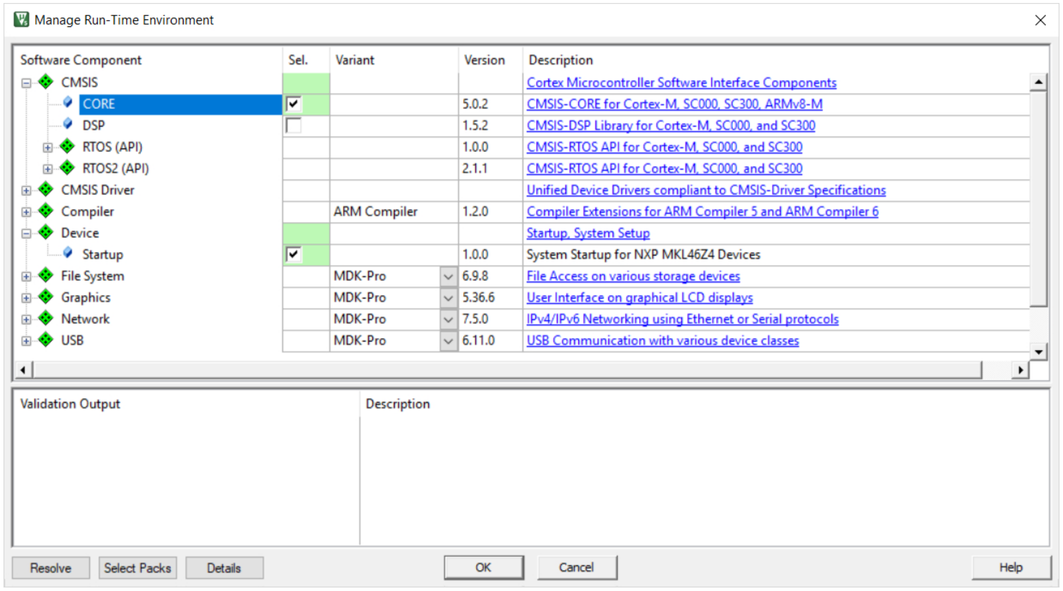
9. Repeat steps 3–9.

In addition, the program must meet these requirements.

- It must use the C function from prelab work to initialize TPM0. As specified in class notes, this function does not have any parameters.
- It must use the assembly language serial driver, (i.e., subroutines, ISR, and receive and transmit queues), from Lab Exercise Nine for character I/O.
- It must use assembly language subroutines from Lab Exercise Five for string and decimal number I/O.
- It must use the assembly language subroutine PutNumHex from Lab Exercise Seven for hexadecimal number output.
- It must have properly commented functions, including a comment header block that includes these components.
  - Description of functionality
  - List of functions called
  - Lists of parameters with descriptions

Lab Procedure

1. Create a new directory and Keil MDK-ARM project with KL05 startup code for this exercise, following the steps below, which differ slightly from previous lab exercises.
   a. When creating the project, name the project and proceed to the target device selection window.
   b. After selecting the KL05 device, click OK to get to the Manage Run-Time Environment window (shown in the following figure).



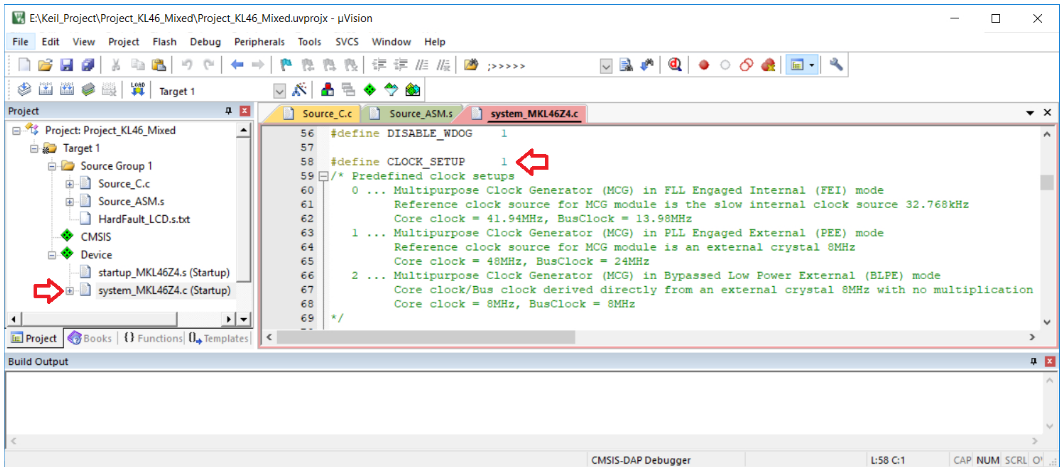
Software Component	Sel.	Variant	Version	Description
CMSIS				Cortex Microcontroller Software Interface Components
CORE	✓		5.0.2	CMSIS-CORE for Cortex-M, SC000, SC300, ARMv8-M
DSP	☐		1.5.2	CMSIS-DSP Library for Cortex-M, SC000, and SC300
RTOS (API)			1.0.0	CMSIS-RTOS API for Cortex-M, SC000, and SC300
RTOS2 (API)			2.1.1	CMSIS-RTOS API for Cortex-M, SC000, and SC300
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
Compiler		ARM Compiler	1.2.0	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Device				Startup, System Setup
Startup	✓		1.0.0	System Startup for NXP MKL46Z4 Devices
File System		MDK-Pro	6.9.8	File Access on various storage devices
Graphics		MDK-Pro	5.36.6	User Interface on graphical LCD displays
Network		MDK-Pro	7.5.0	IPv4/IPv6 Networking using Ethernet or Serial protocols
USB		MDK-Pro	6.11.0	USB Communication with various device classes

Validation Output	Description

Manage Run-Time Environment window

c. As shown in the figure above, under Device select Startup, and under CMSIS select CORE. Then click OK to get to the µVision IDE project window.

d. In the Project pane, expand the project components to find Target1 → Device → system_MKL05Z4.c, and open it. Find the `#define` for `CLOCK_SETUP` and change it from 0 to 1, as shown in the figure below.



```
#define DISABLE_WDOG    1

#define CLOCK_SETUP     1
/* Predefined clock setups
   0 ... Multipurpose Clock Generator (MCG) in FLL Engaged Internal (FEI) mode
         Reference clock source for MCG module is the slow internal clock source 32.768kHz
         Core clock = 41.94MHz, BusClock = 13.98MHz
   1 ... Multipurpose Clock Generator (MCG) in PLL Engaged External (PEE) mode
         Reference clock source for MCG module is an external crystal 8MHz
         Core clock = 48MHz, BusClock = 24MHz
   2 ... Multipurpose Clock Generator (MCG) in Bypassed Low Power External (BLPE) mode
         Core clock/Bus clock derived directly from an external crystal 8MHz with no multiplication
         Core clock = 8MHz, BusClock = 8MHz
*/
```

Clock setup in system_MKL05Z4.c

e. Copy the provided MKL05Z4.s assembly language include file to your project directory.

f. Copy the provided Exercise11.h C header file to your project directory.

2. According to the preceding specification, start with the provided Exercise11_C_Stub.c file to produce a properly commented and properly formatted KL05 C program to control the brightness of the red LED connected to TPM0_CH3 on the KL05Z board based on user input.

3. According to the preceding specification, prepare a properly commented and properly formatted KL05 assembly language module with the serial I/O driver and ROM look-

up tables. You will need to start with the provided ProgramTemplate_C_Start.s, and add your serial driver code from Lab Exercise Nine.

a. Use the assembly language subroutines referenced in the provided C header file as a checklist for the subroutines from previous exercises that need to be included in the assembly language module. There will need to be an EXPORT directive for each of these subroutines in the MyCode AREA. In addition, any subroutines called by these subroutines will also need to be included, but they do not need to be exported.
b. Use the assembly language ROM look-up table entries referenced in the provided C header file as a checklist for the labels needed at the start of each table in the MyConst AREA. There will need also to be an EXPORT directive for each of these labels in that AREA.
c. Before your UART0_ISR label, add the UART0_IRQHandler label and an EXPORT directive for UART0_IRQHandler in the MyCode AREA.

4. Build the program with Keil MDK-ARM to create a listing file and a linker map file.
5. Test your program for proper setting of the LED brightness levels from 1 (dimmest, always off) to 5 (brightest, always on), which must be demonstrated to the instructor.
6. Demonstrate your program for the lab instructor, who will interview you about your program. Be prepared to discuss the points in the "Interview" section that follows.
7. Submit your assembly source file, C source file, listing file, and map file to the myCourses assignments for this exercise.

Interview (each extra credit opportunity also has additional interview questions)

As part of your demonstration, your lab instructor will interview you. Be prepared to show properly documented source code and answer questions about the exercise, in particular with regard to these aspects.

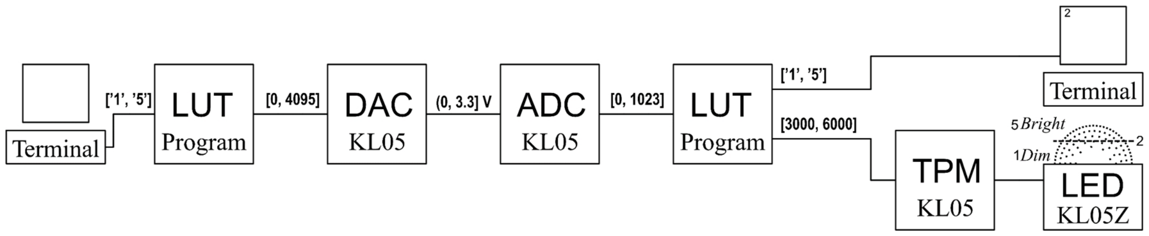
- How did you access the PWM duty table to change the PWM duty cycle?
- Why to the PWM period values decrease as LED brightness increases?
- What are similarities and differences of device programming in C versus assembly language?

Extra Credit Opportunity 1. D/A and A/D Conversions to Compute Brightness Level (20 points)

Application. Explore data transformations between additional data domains by converting the user input to an analog domain and then back to a digital domain to compute the brightness level and PWM duty period. The table below summarizes the transformation of data from user input to LED brightness level. Hardware modules on the KL05 microcontroller perform some of the conversions, and program code performs the other conversions using calculations and/or look-up tables (LUTs). The figure that follows shows a block diagram of the LED brightness control system.

DATA DOMAINS FROM USER INPUT TO LED BRIGHTNESS LEVEL

Source	**Data**	**Range**	**Destination**
User (terminal keyboard)	ASCII	[‘1’, ‘5’]	Program
Program	Integer	[0, 4]	Program LUT for DAC0
Program LUT for DAC	12-Bit Integer	[0, 4095]	DAC0 on KL05
DAC0 on KL05	Voltage	(0, 3.3] V	ADC0 on KL05
ADC0 on KL05	10-Bit Integer	[0, 1023]	Program
Program	ASCII	[‘1’, ‘5’]	User (terminal screen)
Program	Integer	[0, 4]	Program LUT for TPM0
Program LUT for TPM	16-Bit Integer	[49970, 0]	TPM0 on KL05
TPM0 on KL05	PWM	[100%, 0%] of $16^2/_3$ ms	Red LED on KL05Z
Red LED on KL05Z	Brightness	[1, 5]	User (LED brightness)



LED BRIGHTNESS CONTROL SYSTEM

Program Specification. Modify the basic program to perform the following sequence of operations and meet the requirements that follow.

E1. Write C function `Init_DAC0()` to initialize the KL05 DAC0 for continuous (non-buffer) 12-bit conversion of `DAC0_DAT0` to an analog value of range (0, 3.3] V. Follow the class notes on the DAC posted on myCourses. You should use the #define names in the provided Exercise11_C_Stub.c file. Call `Init_DAC0()` from `main()` to initialize the KL05 DAC0.

E2. Write C function `Init_and_Cal_ADC0()` to initialize and calibrate the KL05 ADC0 for polled conversion of single-ended channel 5 (AD5), which is connected to DAC0 output, to an unsigned 10-bit value right-justified in ADC0_RA, (`ADC0->R[0]` in C). Follow the class notes on the ADC posted on myCourses. You should use the `#define` names in the provided Exercise11_C_Stub.c file. Call `Init_and_Cal_ADC0()` from `main()` to initialize and calibrate the KL05 ADC0.

E3. Perform steps 1–2 of the basic program specification on p. 2 to initialize UART0 and TPM0.

E4. Perform steps 3–5 of the basic program specification on p. 2 to prompt the user and get input, which would produce the output shown below on the terminal screen for user input of "1."

```
Type a number from 1 to 5:  1

```

E5. Scale the value input by the user to the full range of DAC0 12-bit conversion using an assembly language ROM look-up table. One way is to convert the ASCII character to a binary value, subtract one from the value, and use the value as an index for accessing the look-up table of DAC0 digital values shown in the myCourses notes on the DAC0.

E6. Write the value to `DAC0_DAT0` to get an analog value for the ADC0.

E7. Output `Original digital value:` followed by the hex representation of the DAC0 digital value, and then advance the cursor to the beginning of the next line, as shown below.

```
Type a number from 1 to 5:  1
   Original digital value:  0x00000199

```

E8. Get a digital value from ADC0 that is the 10-bit analog-to-digital conversion of the voltage output from DAC0.

E9. Output `New digital value:` followed by the hex representation of the 10-bit ADC0 digital value, and then advance the cursor to the beginning of the next line, as shown below.

```
Type a number from 1 to 5:  1
   Original digital value:  0x00000199
        New digital value:  0x00000066

```

E10. Using an assembly language ROM look-up table, scale the 10-bit digital value to a TPM PWM duty count value for a duty cycle between 100% and 0%. One way is first to scale the 10-bit digital value to an index between 0 and 4 (inclusive, which corresponds to LED brightness levels between 1 and 5, respectively) and then to use the index to access the look-up table of PWM duty period values shown in the myCourses notes on the TPM.

E11. Write the duty count value to `TPM0_C3V`, (`TPM0->CONTROLS[3].CnV` in C), to set the LED brightness level requested by the user.

E12. Output `LED brightness:` followed by the LED brightness level, (which is one more than the index calculated in step E10), and then advance the cursor to the beginning of the second line down, as shown below.

```
Type a number from 1 to 5:  1
   Original digital value:  0x00000199
        New digital value:  0x00000066
           LED brightness:  1


```

E13. Repeat steps E4–EE13.

In addition, the program must meet the requirements given for the basic program, which start on p. 2.

Lab Procedure. Follow the lab procedure specified for the basic program, starting on p. 3, modifying procedure step 5 as follows.

E5. Test your program for proper DAC0 and ADC0 functionality and for proper setting of the LED brightness levels from 1 (dimmest, always off) to 5 (brightest, always on), which must be demonstrated to the instructor.

Interview. As part of your demonstration, your lab instructor will interview you. Be prepared to show properly documented source code and answer questions about the exercise, in particular with regard to these aspects.

- How did you generate a different analog voltage for different user inputs, (i.e., “1” through “5”)?
- In hexadecimal representation, what range of digital values can be output to DAC0, (in general terms, not specifically values from the ROM look-up table)?
- In hexadecimal representation, what range of digital values would you expect to input from ADC0 as initialized in this program, (in general terms, not specifically related to the DAC0 outputs used)?
- Do the values you observe from reading ADC0 match what you would expect from the presentation in the lecture notes? If so, explain how they match; if not, explain how they are different.

Extra Credit Opportunity 2. Flicker Fusion Investigation (20 points)

Background. Since voltage supplied by a PWM signal results in the LED being off for a certain amount of time within each pulse period, below a certain frequency the LED will appear to flicker. The frequency below which this occurs varies somewhat from person to person and also depends on the intensity of the LED and physiological factors of the vision system, such as the area of the retina illuminated by the LED [1]. This frequency is the critical flicker fusion frequency (f_{CFF}).

For the original CRT-based televisions, 60 Hz was a standard refresh rate, and it was also the minimum refresh rate in CRT-based computer monitors. In historical accounts regarding the choice of 50 Hz or 60 Hz for the AC power frequency, there have been stories that 60 Hz was chosen in the US over 50 Hz because 60 Hz "was felt to be less likely to produce annoying light flicker" [2]. Since lights powered by AC at 60 Hz actually turn off twice per period, their on/off cycle frequency is actually 120 Hz, and we might thus infer that these early power engineers determined $100 \text{ Hz} < f_{\text{CFF}} < 120 \text{ Hz}$. However studies have indicated that for most people $60 \leq f_{\text{CFF}} < 90$ Hz [1].

If the viewer and LEDs are not stationary, flicker and other physiological effects can be perceived at higher frequencies [1]. When vehicle manufacturers began to use LEDs rather than incandescent bulbs in tail lights, many people found that they could still see flicker in the LEDs powered by PWM even though they were operating at frequencies between 200 and 500 Hz. This phenomenon is caused by rapid eye movement, called "saccade." Thus, f_{CFF} is higher for LEDs viewed in motion than for LEDs viewed from a static position. Frequencies above 3 kHz have essentially no human biological effects [1].

References:

[1] B. Lehman and A. J. Wilkins, "Designing to Mitigate the Effects of Flicker in LED Lighting," *IEEE Power Electronics Magazine*, vol. 1, no. 3, pp. 18-26, Sept. 2014.

[2] E. L. Owen, "The Origins of 60-Hz as a Power Frequency," *IEEE Industry Applications Magazine*, vol. 3, no. 6, pp. 8-14, Nov.-Dec. 1997.

Investigation. As initialized in this lab assignment, the TPM produces a 60-Hz PWM signal to control the brightness of the red LED.

1. Maintaining a fixed gaze on the red LED and keeping the board stationary, check for any detectable flicker of the red LED at each brightness level, and describe any flicker observed in the table below. (As a reference, compare with the flicker-free, always on green LED.)
2. Maintaining a fixed gaze on the red LED, check for any detectable LED flicker of the red LED at each brightness level while you move the board fairly rapidly from side to side (parallel to the short side of the board, and perpendicular to the long side of the board). (As a reference, compare with the flicker-free, always on green LED.) Again, describe any flicker effects observed in the table below.
3. Repeat steps 1 and 2 for each of the PWM frequencies listed in the table below. To change the PWM frequency, edit the `#define PWM_FREQ` in the C source file, edit the `PWM_FREQ` `EQU`ate in the assembly source file, rebuild the project, and run it.

FREQUENCY, INTENSITY, AND MOTION FLICKER EFFECTS WITH PWM LED CONTROL

f_{PWM} (Hz)	Brightness Level	Stationary Flicker Effects	Moving Flicker Effects
60	1		
60	2		
60	3		
60	4		
60	5		
120	1		
120	2		
120	3		
120	4		
120	5		
3000	1		
3000	2		
3000	3		
3000	4		
3000	5		

Interview. As part of your demonstration, your lab instructor will interview you. Be prepared to show properly documented source code and answer questions about the exercise, in particular with regard to these aspects.

- At each of the three different PWM frequencies, (60 Hz, 120 Hz, and 3000 Hz), did you observe any flickering of the LED when looking directly at the KL05Z board with no motion?
- Which (if any) of the PWM frequencies, (60 Hz, 120 Hz, and 3000 Hz), produce visible flicker effects, (e.g., light arrays) when the board is moved fairly rapidly from side to side (parallel to the short side of the board, and perpendicular to the long side of the board).
- Did any of the PWM frequencies produce a different result when the LED was supposed to be off, (i.e., level 1)?
- Did any of the PWM frequencies produce a different result when the LED was supposed to be brightest, (i.e., always on, level 5)?

Submission

myCourses Assignments
Separate assignment for each item below
• Assembly source code (.s)
• C source code (.c)
• Listing (.lst)
• Map (.map)

As specified in "Laboratory and Report Guidelines," late submissions are penalized per day late, where "day late" is calculated by rounding submitted time to the nearest greater or equal integer number of days late, (i.e., ceiling function).

Note: To receive grading credit for this lab exercise you must earn points for *both* the demonstration *and* the report.