ECE220: Computer Systems & Programming **Fall 2020 ZJUI**
Machine Problem 2 **due: Saturday 10 October, 11:59:59 p.m.**

Translating and Printing a Student's Schedule

Your task this week is to write an LC-3 program that translates a student's daily schedule from a list to a two-dimensional matrix of pointers (memory addresses, in this case to names of events), then prints the schedule as shown to the right. Your program must make use of the subroutines that you wrote for MP1 last week. Not counting those subroutines (nor comments, blank lines, and so forth), the program requires about 100 lines of LC-3 assembly code.

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
0600					
0700					
0800					
0900		ECE220		ECE220	213 disc
1000	MATH241	lecture	MATH241	lecture	MATH241
1100	PHYS212	220 lab	PHYS212		
1200	lunch	lunch	lunch	lunch	lunch
1300					
1400				study	
1500			PHYS212	with	
1600	MATH213		disc	friends	
1700	dinner	dinner	dinner	dinner	dinner
1800			MATH213		PHYS212
1900		date			lab
2000		night			

The objective for this week is to give you some experience with understanding and manipulating arrays of data in memory, as well as a bit more experience with formatting output.

The Task

In this program, you must translate events with variable-length into a schedule with fixed-length fields (pointers to the variable-length names of the events). Fixed-length fields simplify random access to data, as accessing field number N simply means multiplying N by the size of the field to find the offset from the start of the data. Before translating, you must first initialize the schedule. After translating, you must print it with day names on the top and hour names on the left, as shown above.

The event list starts at address x4000 in LC-3 memory. Each event consists of three fields. The first field is a bit vector of days for the event: Monday is bit 4 (value 16), Tuesday is bit 3 (value 8), and so forth, through Friday (bit 0, value 1). The days on which the event occurs are OR'd together to produce the bit vector. The second field is a label describing the event (a string, which is a sequence of ASCII characters ending with NUL, x00). The third field is an hour slot number and should range from 0 to 14, with 0 indicating the 06:00 slot in the schedule, 1 indicating the 07:00 slot in the schedule, and so forth.

The event list ends with a bit vector of -1. In other words, when your program finds -1 where the bit vector for the next event should be, the program has reached the last event in the list and should proceed with printing the schedule by calling the subroutine PRINT_SCHEDULE, which you must write.

As the event labels have variable length, the number of memory locations occupied by each of the events also varies. The shortest valid event has a bit vector of days, an empty name string (a NUL), and an hour slot, for a total of three memory locations. There is no upper bound on the length of an event, although of course everything must fit into LC-3 memory.

Variable-length data structures such as the events in the event list are difficult to use, as finding the start of the N^{th} event requires starting at the beginning of the list and walking through all previous events in the list.

To make the information easier to use, your program must translate the event list into a schedule, a two-dimensional array of pointers to strings, starting at address x3800 in LC-3 memory. Each string pointer in the schedule is either a pointer to one of the event labels, or is the special value NULL (x0000, which by convention points to nothing). The array consists of 15 one-hour slot arrays (from 06:00 to 20:00), each of which consists of five memory locations (one for each day, starting with Monday and ending with Friday). Each day within a slot array uses one memory location. So to calculate the address for the Thursday 13:00 slot, first find the hour slot number, 13 – 6 = 7, then multiply by 5 to get 35, then add 3 for the Thursday slot (fourth day of the scheduled week) to obtain 38, and finally add the 38 (x26) to the start of the schedule at x3800 to obtain x3826. The total schedule requires 75 (15×5) memory locations.

Each memory location in the schedule is a pointer (a memory address). If the pointer is NULL (has value x0000), that slot in the schedule is free. Otherwise, the string to which the pointer points describes the event for a particular one-hour block on a particular day. Before your program copies the addresses of event labels into the schedule, it must initialize the schedule by filling all 75 entries with NULL (x0000). **You should NOT assume that the memory locations are initialized to x0000.**

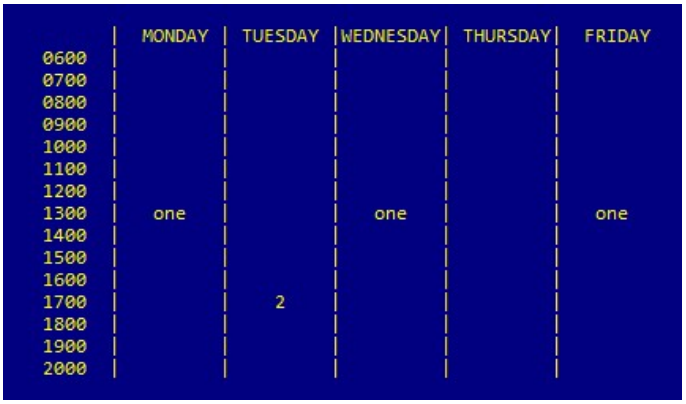
In order to translate the event list to the schedule, your program must walk through the events from the first to the last. For each event, write the address of the first character in that event's label into the correct slot in the schedule for each of the days covered by the event (examine the event's bit vector to determine which days should be included). If a memory location in the schedule is already occupied by another event (the pointer is not NULL), your program must detect the conflict, print an error message, and terminate. Otherwise, the event should be added to all appropriate locations in the schedule by replacing NULL with a pointer to the event's label.

You may assume that all event labels are valid ASCII strings. Note that an empty string (a single NUL) **is a valid ASCII string**. You may further assume that all bit vectors of days are valid combinations representing (possibly empty) subsets of weekdays (Monday through Friday), and that all larger bits (bits 15 through 5) are 0, except for the event termination marker at the end of the event list.

Some events may have bad slot numbers (values not in the range 0 to 14). In such a case, your program must detect the problem, print an error message, and terminate.

Here is an example (provided to you as **`simple.asm`**) of how an event list might appear in memory and how the schedule produced by your program should appear after translation and printing. The schedule here consists of only two events starting at x4000 and x4006. The -1 at x400A marks the end of the list.

address	contents	Meaning
x4000	x0015	M (16) | W (4) | F (1)
x4001	x006F	'o'
x4002	x006E	'n'
x4003	x0065	'e'
x4004	x0000	NUL
x4005	x0007	slot #7, 13:00
x4006	x0008	Tue (8)
x4007	x0032	'2'
x4008	x0000	NUL
x4009	x000B	slot #11, 17:00
x400A	xFFFF	-1 (ends list)



Specifics

- Your code must be written in LC-3 assembly language and must be contained in a single file called **`mp2.asm`** in the **`mp/mp2`** subdirectory of your repository. We **will not grade** any other files.
- Your program must start at x3000. You must write a subroutine called PRINT_SCHEDULE to print the schedule. This subroutine takes no inputs (it uses the schedule from memory) and generates no outputs. For simplicity, all registers are caller-saved, and the subroutine prints the schedule from memory to the display. Your subroutine must be able to execute without execution of your main program (for our testing purposes).
- The schedule is an array of 15 arrays of 5 pointers to strings. Each pointer is the starting address of an event label, or NULL (x0000) if that time on that day is free in the schedule.
- Your program must first initialize all 75 memory locations in the schedule (memory locations x3800 through x384A) to NULL pointers (x0000) before translating events from the list into the schedule.
- Your code must not access the contents of memory locations other than those required for this MP or declared within your program (using .FILL or .BLKW).
- Your program must translate the event list starting at x4000 in memory into the schedule at x3800.
    - Each event in the list consists of a bit vector of days of the week (Monday = 16, Tuesday = 8, Wednesday = 4, Thursday = 2, Friday = 1), a NUL-terminated sequence of ASCII characters, and an hour slot (0 = 06:00, ..., 14 = 20:00).
    - A bit vector of -1 ends the event list (the final entry is not considered an event, and no string nor hour slot number are included after -1 bit vector).
    - You may assume that all events' bit vectors have 0 bits in the high 11 bits.
    - You may **not** assume that event bit vectors are non-zero.
    - You may **not** assume that event strings are non-empty.
    - If the slot number for an event is not valid (not in the range 0 to 14), your program must print the label of the invalid event followed by the string, " has an invalid slot number.\n", then terminate without processing further events nor printing the schedule. Note the leading space in the suffix string provided. Your program's output must match exactly.
    - If an event conflicts with a previous event, your program must print the label of the invalid event followed by the string, " conflicts with an earlier event.\n", then terminate without processing further events nor printing the schedule. Note the leading space in the suffix string provided. Your program's output must match exactly.
- After translating the event list successfully, your program must print the schedule by calling the PRINT_SCHEDULE subroutine, which must work as follows:
    - The appearance of the schedule printed by your program must match the figures in this document exactly.
    - The first line printed should provide the days of the week in capital letters ("MONDAY", "TUESDAY", and so forth) printed using **`PRINT_CENTERED`** and separated by the vertical line character (ASCII x7C).
    - Each subsequent line should begin with an hour slot number (printed used **`PRINT_SLOT`**) followed by the events for that hour on Monday, Tuesday, and so forth. If no event is scheduled in that slot (a NULL pointer in the schedule), pass an empty string to **`PRINT_CENTERED`** (do **not** pass a NULL pointer). If an event is scheduled, print the name of the event with **`PRINT_CENTERED`**. Separate the days of the week with the vertical line character (ASCII x7C). End each line with a line feed character (ASCII x0A).

- Your code must be well-commented and must include a table describing how registers are used within each part of the code: initializing the schedule, translating the event list to the schedule, and printing the schedule. Follow the style of examples provided to you in class and in the textbook.
- Do not leave any additional code in your program when you submit it for grading.

Testing

We suggest that you adopt the following strategy when developing your program:

1. Begin by writing the code that prints the schedule to the display. Your code must make use of **`PRINT_CENTERED`** and **`PRINT_SLOT`**. Note that you can produce the empty slot label for the first line of the schedule (the days of the week) by calling **`PRINT_CENTERED`** with an empty string. To test this part of your code, we have provided a fake schedule for you (called **`fake.asm`**). To test, assemble and load the fake schedule into the simulator, load your program, then **`continue`** execution
2. Once your schedule printing code works, you can add code to clear the memory for the schedule. If you test with **`fake.asm`**, you should then see an empty schedule. Be sure to write exactly 75 zeroes into memory—you can use the simulator's dump command to check that you have not overwritten memory after the schedule; the strings in **`fake.asm`** should be untouched.
3. Finally, write the code to translate the event list into the schedule. We have provided a few sample schedules (**`broken.asm`**, **`simple.asm`**, **`sched1.asm`**, and **`sched2.asm`**) and scripts, but be sure to create and test some of your own as well.

Remember that **testing your program is your responsibility**. The strategy here is intended to make the process simpler for you, but does not guarantee that your program contains no errors.

Grading Rubric

Functionality (50%)

- 5% - program initializes the schedule correctly
- 15% - program correctly translates valid events from the list into the schedule and stops at the end of the event list
- 5% - program handles schedule conflicts correctly (including error message output)
- 5% - program handles bad slot numbers correctly (including error message output)
- 5% - program prints schedule header correctly (the line with days of the week)
- 15% - program prints schedule correctly

Style (20%)

- 10% - A doubly-nested loop (hours for the outer loop, and days for the inner loop) is used for printing the schedule.
- 10% - program uses **`PRINT_SLOT`** and **`PRINT_CENTERED`** appropriately in order to print the schedule (the only output directly from the main program should be vertical lines, line feeds, and error messages), and does not pass invalid input to either subroutine (such as NULL pointers to **`PRINT_CENTERED`**)

Comments, Clarity, and Write-up (30%)

- 5% - a paragraph appears at the top of the program explaining what it does (this is given to you; you just need to document your work)
- 15% - each of the three parts of the code (initialization, translation, and printing) has a register table (comments) explaining how registers are used in that part of the code
- 10% - code is clear and well-commented

Note that some categories in the rubric may depend on other categories and/or criteria. For example, if your code does not assemble, you will receive no functionality points. Note also that the remaining LC-3 MP (MP3) will build on these subroutines, so you may have difficulty testing those MPs if your code does not work properly for this MP. You will also be penalized heavily if your code executes data or modifies itself (do not write self-modifying code).

Sharing MP1 Solutions

To help you in testing your code, you may make use of another student's MP1 solution as part of your MP2, provided that you **strictly obey the following**:

- You may not obtain another student's MP1 solution until **after class on Tuesday 29 September**. Violation of this rule is an academic integrity violation, will result in BOTH students receiving 0 for MP1, and may have additional consequences.
- You must clearly mark the other student's code in your own MP2, and must include the students name in comments indicating that you are using their code. **Failure to mark their code appropriately will result in your receiving a 0 for MP2.**
- As you should know already, you may not share any additional code beyond the solution to MP1.

Please also note that if the other student's code has bugs that lead to your introducing bugs into your MP2 code, you may lose points as a result. We in no way guarantee the accuracy of any student's MP1 solution.