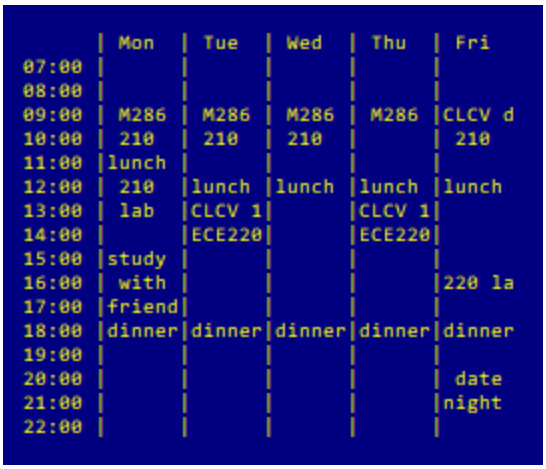
ECE220: Computer Systems & Programming **Fall 2022**
Machine Problem 2 **due: Saturday 8 October, 11:59:59 p.m.**

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.



```
      | Mon   | Tue   | Wed   | Thu   | Fri
07:00 |       |       |       |       |
08:00 |       |       |       |       |
09:00 | M286  | M286  | M286  | M286  |CLCV d
10:00 | 210   | 210   | 210   |       | 210
11:00 |lunch  |       |       |       |
12:00 | 210   |lunch  |lunch  |lunch  |lunch
13:00 | lab   |CLCV 1|       |CLCV 1|
14:00 |       |ECE220|       |ECE220|
15:00 |study  |       |       |       |
16:00 | with  |       |       |       |220 la
17:00 |friend|       |       |       |
18:00 |dinner|dinner|dinner|dinner|dinner
19:00 |       |       |       |       |
20:00 |       |       |       |       | date
21:00 |       |       |       |       |night
22:00 |       |       |       |       |
```

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 x5000 in LC-3 memory. Each event consists of three fields. The first field is a label describing the event (a string, which is a sequence of ASCII characters ending with NUL, x00). The second field is a bit vector of days for the event: Monday is bit 0 (value 1), Tuesday is bit 1 (value 2), and so forth, through Friday (bit 4, value 16). The days on which the event occurs are OR'd together to produce the bit vector. The third field is an hour slot number and should range from 0 to 15, with 0 indicating the 07:00 slot in the schedule, 1 indicating the 08:00 slot in the schedule, and so forth.

The event list ends with an empty string. In other words, when your program finds that the name of the next event has length 0 (not counting the NUL), the program has reached the last event in the list and should proceed with printing the schedule.

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 name of one character followed by a NUL, a bit vector of days, and an hour slot, for a total of four 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 x4000 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 16 one-hour slot arrays (from 07:00 to 22:00), 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 – 7 = 6, then multiply by 5 to get 30, then add 3 for the Thursday slot (fourth day of the scheduled week) to obtain 33, and finally add the 33 (x21) to the start of the schedule at x4000 to obtain x4021. The total schedule requires 80 (16×5) 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 80 entries with NULL (x0000). **You should NOT assume that the memory locations are initialized to x0000.**

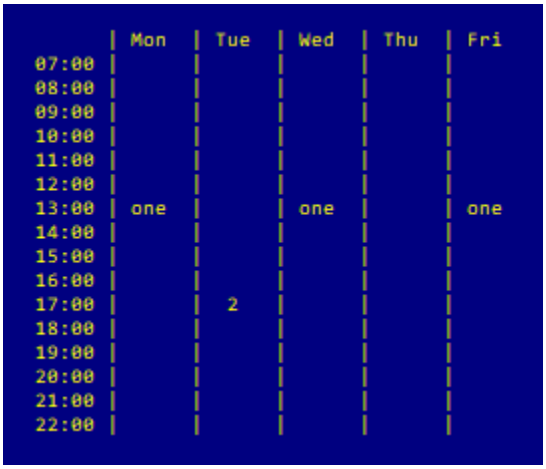
In order to translate the event list to the schedule, your program must walk through the events from the first to the last. For each event, write the address of the first character in that event's label into the correct slot in the schedule for each of the days covered by the event (examine the event's 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. You may further assume that all bit vectors of days are valid combinations representing (possibly empty) subsets of weekdays (Monday through Friday).

Some events may have bad slot numbers (values not in the range 0 to 15). 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 x5000 and x5006. The NUL at x500A marks the end of the list.

address	contents	Meaning
x5000	x006F	'o'
x5001	x006E	'n'
x5002	x0065	'e'
x5003	x0000	NUL
x5004	x0015	Mon (1) | Wed (4) | Fri (16)
x5005	x0006	slot #6, 13:00
x5006	x0032	'2'
x5007	x0000	NUL
x5008	x0002	Tue (2)
x5009	x000A	slot #10, 17:00
x500A	x0000	NUL (empty string ends list)



Specifics

- Your code must be written in LC-3 assembly language and must be contained in a single file called **`mp2.asm`**. We **will not grade** files with any other name.
- Your program must start at x3000.
- Your program must initialize the schedule in memory locations x4000 through x404F to contain all NULL pointers (x0000) initially.
- The schedule is an array of 16 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 80 memory locations in the schedule to NULL before translating events from the list into the schedule.
- Your program must translate the event list starting at x5000 in memory into the schedule at x4000.
    - Each event in the list consists of a NUL-terminated sequence of ASCII characters, a bit vector of days of the week (Monday = 1, Tuesday = 2, Wednesday = 4, Thursday = 8, Friday = 16), and an hour slot (0 = 07:00, … , 15 = 22:00).
    - An empty string ends the event list (the final entry is not considered an event, and no bit vector of days nor hour slot number are included after the empty string).
    - If the slot number for an event is not valid (not in the range 0 to 15), 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, your program must print the schedule.
    - The appearance of the schedule printed by your program must match the figures in this document exactly.
    - The first line printed should provide three-letter prefixes of days of the week ("Mon", "Tue", 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), print an empty string with PRINT_CENTERED. 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.

Coding Style

In general, being able to write readable code is a skill that's just as important as being able to write working code. People and industry teams have their own preferences and rules when it comes to coding style. In this class, we won't nitpick over small things such as spaces, blank lines, or camel case, nor will we enforce any rigorous coding guidelines. However, we still do have a basic standard that we expect you to adhere to and will be enforced through grading. Our expectations for the rest of the semester (not just this MP!) are outlined below:

1. Give meaningful and descriptive (but not too long) names to your variables, labels, constants, functions, and files. Be consistent in your naming conventions.
2. Do NOT use magic numbers (any number that appears in your code without a comment or meaningful symbolic name). -1, 0, and 1 are usually OK when used in obvious ways.
3. Keep programs and functions relatively short. Don't write spaghetti code that jumps back and forth everywhere.
4. Use comments to explain the interfaces to all functions or subroutines, lengthy segments of code, and any non-obvious line of code. However, do NOT overdo it. Too many comments is just as bad as too little. Use comments to explain why, not what.

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`**; you must assemble it yourself). To test, first load the fake schedule into the simulator, then load your program, then execute.
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 80 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. You can start your testing with the sample schedules that we have provided (**`simple.asm`**, **`sched1.asm`**, and **`sched2.asm`**), but be sure to also create schedules with conflicts and bad slot values and test with those.

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)
- **-10% PENALTY** - VSCode extension reports any errors or warnings; note that even a single warning incurs the full 10% penalty

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. Similarly, if your PRINT_SLOT and/or PRINT_CENTERED subroutines are not working, you will receive few or no points for printing the schedule.

Sharing MP1 Solution

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 noon on Tuesday 4 October. 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 BOTH students receiving 0 for both MP's.
- 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.