Design Notes

Jeremy Case, Xinhua Fan, Teodor Georgiev, Julie Yu

For Test Program 2:

Test program 2 will take a user input of six sentences, and store that to memory. Afterwards, it will take another user input and look through the original six sentences for this word. If and when it finds the word, it will output the word, its sentence location, and the word number of that sentence that the match was found at.

The program does this through a series of steps, as described below. Keep in mind that the program will not work properly if the user inputs more or less than six sentences, there are grammar errors regarding multiple spaces or periods, or the user input is extremely large (1700+ chars).

First and foremost, once the program is loaded, the program will take the user input and store it to memory. The program knows that this input; however long, will be placed into memory starting at location 200. The second user input (the word to be looked for) will be placed into memory starting at location 1900.

After the second user input, the program loads into memory the number of chars of the second user input, and then starts its main loop. It will start by looking at every single char of the first user input starting at memory location 200. On every loop, the following events occur:

- The program loads from memory a space char. It then compares the current loop's char against this ASCii value. If the current char is a space, the program increments a "word counter" in memory.
- The program then loads from memory a period char. It then compares the current loop's char against this ASCii value. If this

current char is a period, the program increments a "sentence counter" in memory. It then also resets the aforementioned word counter, since we are now located at the first word of another sentence.

- The program then loads from memory a "match counter". It uses this number to know which char from the second user input to use when matching against the current iteration of the loop. I.e., if we have managed to match two chars, let's look at the third char of the user input if the user input "look" – we would using the second "o" to compare.
- If the program finds a match, we increment the match counter into memory.
    - We then compare the match counter against the number of chars contained in the second user input. Unless they are equal, we continue the main loop. If they are equal, we have found the chars we're looking for. Therefore, we break the main loop.
        - When we break the main loop, we load into registers the sentence and word counters, and then output those along with the word. This gives us the word and sentence location of the word to be looked for.
- If we do not find a match, we reset the match counter back to zero, and continue the main loop.
- If we run into a null memory location during the main loop, we simply output that we have not found the word the user wants to look for in the first six sentences. This is equivalent to not finding the user-input-word since we have looped through every char in memory and did not find a match.

For Trap/Machine Fault:

Prerequisite:

We design a table can have a maximum of 16 entries representing 16 routines for user-specified instructions as below:

Table 1

routine ID	user instruction	comment
0	populate MSR with 100	Case: trap code = 0;
1	populate MSR with 101	Cases: 1. trap code = 1; 2. Illegal Memory Address to Reserved Locations
2	populate MSR with 102	Cases: 1. trap code = 2; 2. Illegal TRAP code
3	populate MSR with 103	Cases: 1. trap code = 3; 2. Illegal Operation Code
4	populate MSR with 104	Cases: 1. trap code = 4; 2. Illegal Memory Address beyond 2048 (memory installed)
5	populate MSR with 105	Case: trap code = 5;
6	populate MSR with 106	Case: trap code = 6;
7	populate MSR with 107	Case:

		trap code = 7;
8	populate MSR with 108	Case: trap code = 8;
9	populate MSR with 109	Case: trap code = 9;
10	populate MSR with 110	Case: trap code = 10;
11	populate MSR with 111	Case: trap code = 11;
12	populate MSR with 112	Case: trap code = 12;
13	populate MSR with 113	Case: trap code = 13;
14	populate MSR with 114	Case: trap code = 14;
15	populate MSR with 115	Case: trap code = 15;

In the program, we initialize this table to the memory from location 10 to 25 through Memory(class)'s construction:

Table 2

memory location	memory location's value	comment
10	0	for routine ID = 0
11	1	for routine ID = 1
12	2	for routine ID = 2
13	3	for routine ID = 3

14	4	for routine ID = 4
15	5	for routine ID = 5
16	6	for routine ID = 6
17	7	for routine ID = 7
18	8	for routine ID = 8
19	9	for routine ID = 9
20	10	for routine ID = 10
21	11	for routine ID = 11
22	12	for routine ID = 12
23	13	for routine ID = 13
24	14	for routine ID = 14
25	15	for routine ID = 15

When a Trap instruction or a machine fault occurs, first, our program retrieves the corresponding memory location according to the trap code or the machine fault type as described in table 1 and table 2, and then stores the retrieved memory location to memory location 0 (trap) or 1(machine fault); populates MFR with trap code or the ID of machine fault; saves the current PC to memory location 2 (trap) or 4(machine fault); fetches the address from Location 0 (trap) or 1 (machine fault) into the PC which becomes the next instruction to be executed. After corresponding routine executed, recover PC with the address in memory location 2 (trap) or 4(machine fault).
For example,

1. If trap code is 10 (the PC is 100 before trap), our program will get memory location 20 and store 20 to memory location 0; populate

MFR with 10; save 100 to memory location 2; store 20 to PC (point to the executed routine); execute the routine with ID = 10(populate MSR with 110); lastly, recover PC with 100 from memory location 2.

2. If Illegal Operation Code occurs (the PC is 100 before machine fault occurs), our program will get memory location 13 and store 13 to memory location 1; populate MFR with 3; save 100 to memory location 4; store 13 to PC (point to the executed routine); execute the routine with ID = 3(populate MSR with 103); lastly, recover PC with 100 from memory location 4.