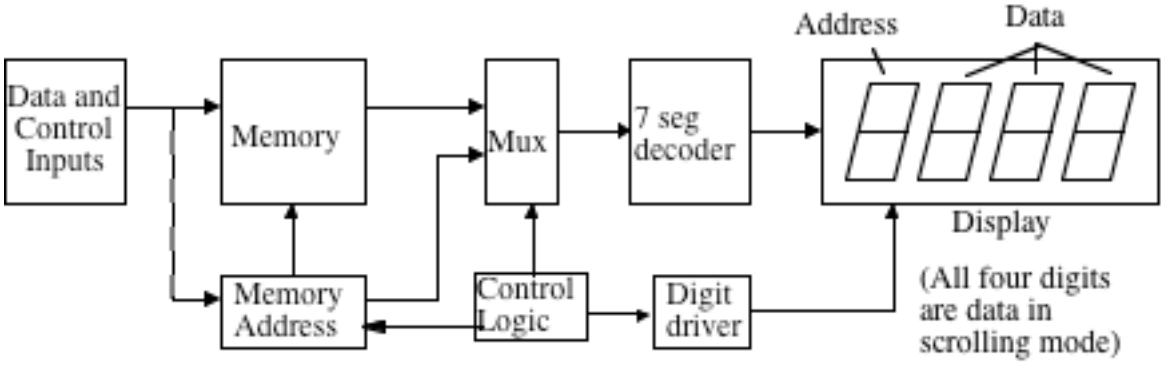
EE 241 Digital Design Laboratory Assignment #8
Scrolling Display Apr 27 for operation

In this exercise, memory and other logic is added to your multiplexed display. You can use a device programmer to insert data into any of the addresses in a 4 bit wide memory, and see the (code for) the contents of that address. (A PAL is to be used for the 7 segment decoding, allowing the use of 4 or more bits, and consequently 16 or more characters in the character set. (In an emergency, you could use a 74LS47 for just numbers.) An inspection could be included to show the address currently accessed, and the values of the data at that address and following. The idea is that this allows you to see where each character is in the address space of the memory. The following block diagram illustrates the concept. The controller controls operation in either the normal scrolling mode or in the inspection mode. You may want to eventually to go to more than 4 characters displayed at a time.



The data path includes a data input device (DIP switches, or if you want to get fancy, a keyboard) which can be used to select from among several messages. The decoder and display are inherited from previous labs. You will no longer use the earlier multiplexing circuitry for choosing the character in memory. Instead, you will normally be successively displaying 3 or 4 digits of data from the read only memory device. The "memory address" register shown above gives the starting point for the data displayed. The multiplex is two way, to choose from either the memory address (used in inspection mode only) or (normally) the memory.

The report for this lab will be a demonstration, but you may optionally submit a formal or informal report instead (for more credit). If you do not have it working by the due date, you must submit an informal report by exam time whether the circuit is working or not. The informal report must include the schematic, a description of how the circuit operates, and conclusion. (For credit for working correctly, you must demonstrate it.) You are to include reasons for the data path circuit not operating properly, and its current state of operation.

Scrolling Display Strategy:

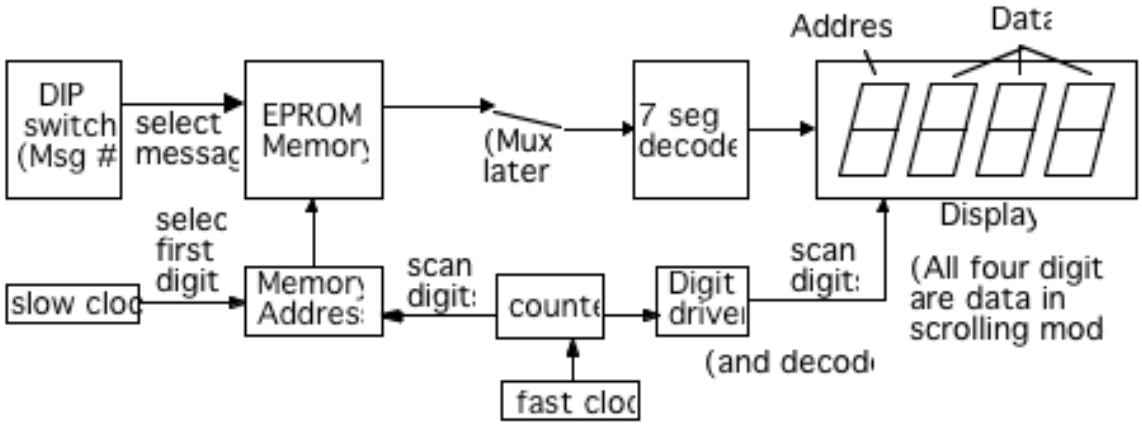
In order to get scrolling, you will need a control unit that goes through a sequence of states (possibly many states) to move the user's view either one character or 4 characters at a time through the memory. This can be done using a counter for the memory address. But, does the counter need to also count while the display goes from one digit to the next? Here are some possibilities. These are not exhaustive; you may come up with something better.

1. Use a 74193 up/down counter for the address. (I won't detail this approach; it's more complex. If you want to try it, ask me about it.)

2. Use an adder (7483 or 74283) in the data path. (DO THIS!) The address register can stay constant (except for scrolling) and the display counter offset is added to give the actual memory address. So, as the display counter goes 0-1-2-3-0-1-2-3-0-1 etc, if the address register is set to "3", the sum to the memory address pins would be 3-4-5-6-3-4-5-6-3-4 etc. This makes the data path more complex than the previous method, but allows the job to de done without needing up/down counters, and the control logic will be simpler. When scrolling, the address register is simply counted up occasionally, at perhaps 1 Hz or so. When doing inspection, the address register would be constant, and the addresses would need to be sent as a-0-1-2-a-0-1-2 etc. This presents a problem, since in this mode you really need the digit -1 to be added to the address. This is easily fixed by imagining that the address register actually contains the data address -1, and in normal (display, or scrolling) mode simply add the 1 back using the "carry in" of the added. In entry mode, carry in would be 0. This alternative is probably easier.

Steps toward the design: #1, EPROM stored messages

Rather than trying to do all of this at once, we will do it in discrete steps. You can demonstrate each step for partial credit toward the final goal. (Thus, it won't be catastrophic if you do not manage to get all of the controller capabilities working in the last step.) Step #1 is to do it for a set of messages stored in the EPROM with no inspection mode. The figure below illustrates.



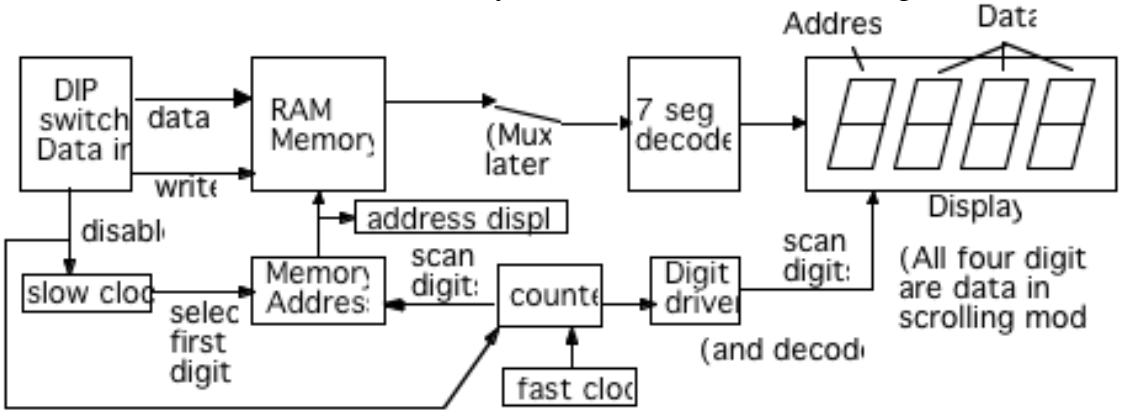
Here, the low 4 address wires to the EPROM select which digit of the selected 16 character message is sent to the display. (Additional address wires can be used for message selects.) The next few (depending on how many different messages you store in the EPROM) select which 16 character message you display. The slow clock and scan clock both affect the number for the digits to be displayed. You can have the slow clock run an independent counter, and that 4 bit value is added to a 2 bit value from the scan counter. The 7483 or 74283 is a 4 bit adder that can do the arithmetic. Or, you could use a 4 bit up-down counter and manipulate it so that it scans up and down to cover 4 adjacent characters, getting an occasional extra count up to scroll. I think the adder is easier. If you build and demo this, you are about 1/2 of the way to the final possible goal. If you get this working with no problems, perfect report, neat and color coded, etc. you can get a 90. The rest of the exercise beyond this can take you over 100.

#2 Add inspection mode

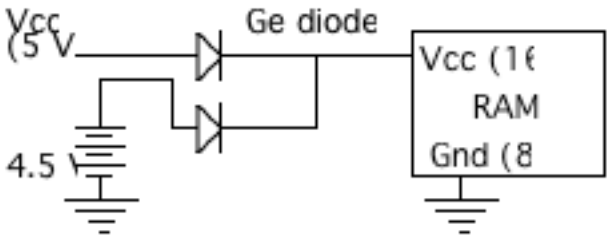
In this step, you add the multiplex that selects between the memory and the address register (as shown in the initial diagram). The control mechanisms are then added to choose mode and correctly manipulate the timing depending on what mode you are in. You will want to use a debounced SPDP pushbutton as the "slow clock" input in inspection mode to, at the pace you want to go, scroll through the various addresses. You can use one of the DIP switches as the select to choose between normal and inspection mode. (Other switches can select which message.) If you have this working, all other aspects being great, then you can earn a 100.

#3 RAM stored messages, without inspection mode working (yet) This is optional, only for the ambitious. Don't try it unless you have demonstrated the ROM approach first, then do for extra credit later.

This step is almost the same as step #1 above, but we substitute a 16 character RAM (74189) for the EPROM. Instead of the message being pre-stored (requiring an EPROM programmer) the message is put in with DIP switches one character at a time. (You could also include both RAM and EPROM, selected by a DIP switch, to include both possibilities!)

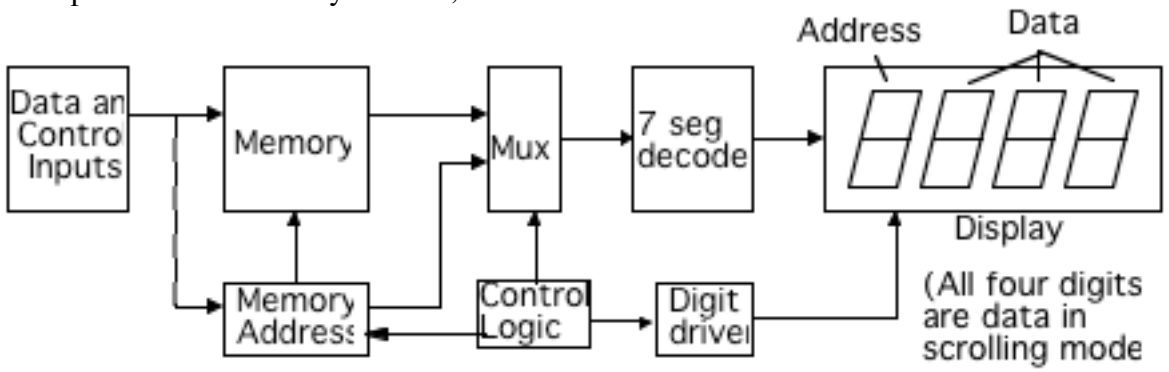


In order to write data into RAM, you need to either be in inspection mode (if you have that working) or otherwise stabilize the address so that the RAM address (from the Memory Address register) is a constant. We don't want it to change while a write is occurring. This is what the "disable" switch (one of the DIP switches, or a SPST pushbutton) does. (You'd really like to use clever timing to do this instead.) We also need an indicator of what the current address is so that we can "bump" it up or down by enabling the counters (at least, the slow counter) for a while; that's why we really would prefer a working inspection mode. We want to write into a particular one of the RAM addresses. Once the display (and RAM) is not scanning and we have the correct address on the Address, we can use a switch (or better, a pushbutton) to assert the "Write" signal to the RAM, which causes the data value on the DIP switches to be written to the RAM. We can use a battery back-up to keep the RAM alive when the circuit is not plugged in with a circuit as shown below. (This might even work with Si diodes, but the RAM voltage may be too low to reliably retain the data. The TTL RAM we are using is not particularly low power. If you are serious about this, use CMOS RAM parts instead. I have 6116's, which have the additional advantage of being 8 bits wide and 2K bytes in size, so you can have a much bigger character set and message.) If you do that, you may be able to use a big capacitor instead of a battery. You can get a Farad for $5. It's cool to own your own Farad. You can sneer at all the smaller capacitors.)



#4 RAM with working inspection mode::

This step uses the display mode multiplex to get a working inspection mode so that when you are entering data, you can see the address of the character being modified, and also see three characters (the one being changed and the next two) at the same time. Thus, the device works like a normal editor. The display always shows four characters, so the "write" signal must be carefully controlled so that it only goes low when the correct digit is being accessed. While entering data, we do not want to be scrolling. Instead, we would like to manually step it to the next character, possibly even as part of the "enter" function. There are lots of possibilities here, described earlier. This is ultimately what we would like to see. This is full credit with bonus: a 115 if report etc. is fine. Maybe more; dunno.



#5 Extras way beyond anything you have time to do:

1) More than 4 digits. Especially if you come up with a strategy for making them bright (higher peak current) but protecting against burn-out if the clock stops. In fact, doing that kind of brightness enhancement is worth something even if you stay at 4 digits.
2) Use BCD or hexadecimal for the address in inspection mode rather than the normal character set (a 7447, for example, could be a selected alternate seven segment decoder)
3) Use of an Ascii character set (but only if you are using an 8 bit wide RAM: 6116 or two 7489's)
4). RS232 control from computer: Use a UART for an input device, and enter data using the "Hyperterminal" program from a PC. This is buckets of useful functionality. If you want to go farther, have it display the current message contents. You need RAM working to do this; you may not need inspection mode. Another option is to use a small keyboard; I have some.
5) Dot matrix rather than seven segment for your display. This lets you scroll by column instead of whole characters. You will have to come up with a way of building a good sized display. You need at least 5 x 7 (35 LED's) for each digit. You could mount them on a separate board. Or buy 5x7 displays. (I have some tiny ones.) You will need brightness enhancement / overcurrent protection if you do this. Use an EPROM for your character generator.
6) If you can think of something else that would be neat, like being able to display GPS position, telephone in your message, or whatever else, I'd be anxious to see it!