18-240: Structure and Design of Digital Systems

HW7 Solutions [4 problems, 64 points]

Covers lectures L14 – L15 *Due: 6 November 2023*

Drill Problems [26 points]

1. [26 points, Lecture 14] For this problem, design a datapath to determine if a string located in a memory bank is a palindrome. A palindrome is a pattern that reads the same when read forward and backward. Famous example palindrome strings include "Able was I ere I saw Elba" and "Ma is as selfless as I am." Spaces and punctuation are normally ignored when viewing a palindrome. However, to keep from complicating your datapath overmuch, every character will count.

Part A: Data [2 points]

- Address of the "front" of the string
- Address of the "back" of the string
- Value read from memory (i.e. the "character" read from the beginning of the string)
- Value read from the "back" of the string can be output from the memory without putting it in a separate register.

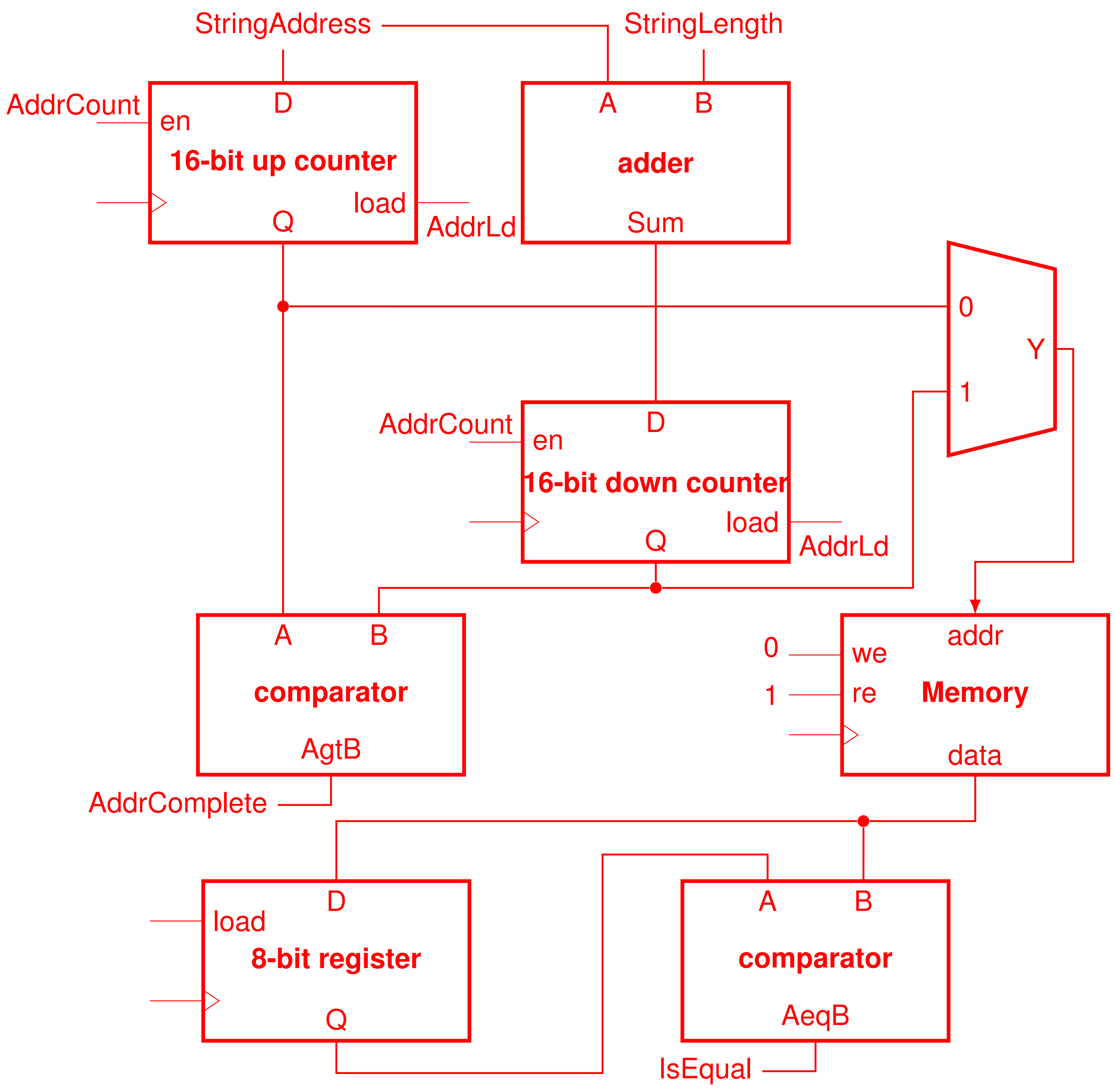
Part B: Transformations [2 points]

- front_address $\Leftarrow$ StringAddress
- back_address $\Leftarrow$ StringAddress + StringLength
- front_address $\Leftarrow$ front_address + 1
- back_address $\Leftarrow$ back_address - 1
- value $\Leftarrow$ M[front_address]

Several status points need to be created by the datapath, but aren't really put in registers. Therefore, they sort of fall outside the limits of the question I asked for part B.

- front_address > back_address ?
- value == M[back_address] ?

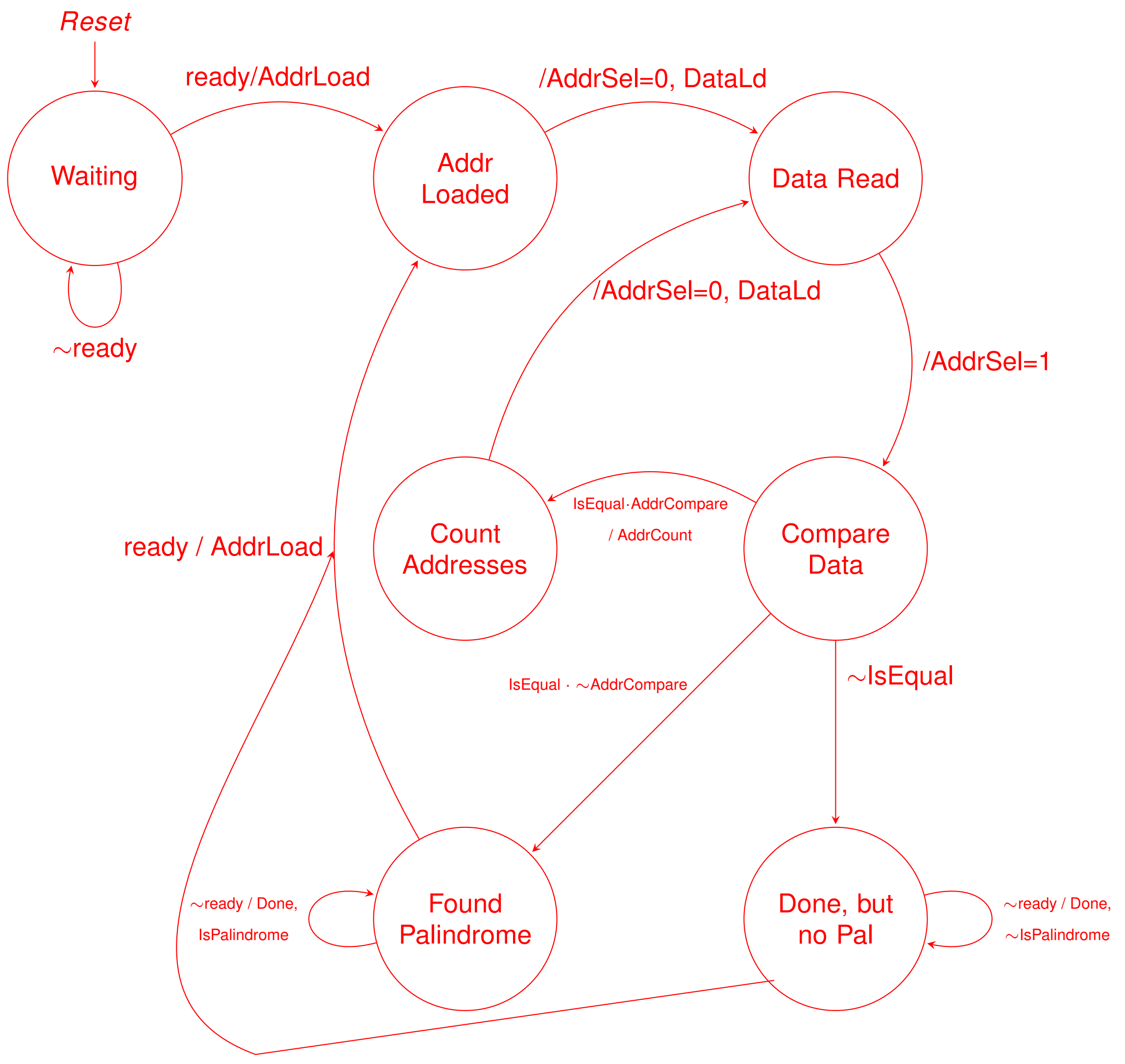
Part C: Datapath [10 points] Note that we only are ever reading from the memory, so we don't need bus drivers to protect any circuitry.



Part D: Control/Status Points [2 points]

- Give the list of control points (i.e. signals from control FSM to datapath).
  AddrCount, AddrLoad, DataLoad, AddrSel
- Give the list of status points (i.e. signals from datapath to control FSM).
  AddrComplete, IsEqual
- For each input from the external world specify a destination as FSM or datapath.
  StringAddress goes to datapath
  StringLength goes to datapath
  Ready goes to FSM
- For each output, does it get generated by your FSM or your datapath?
  Done and IsPalindrome are both generated by the FSM

Part E: Control FSM [10 points]



Non-Drill Problems [38 points]

2. [12 points, Lecture 15] Memory systems are created from several memory (or other) components, each using some subset of an overall address range. In this problem, you will build a memory system.

   This is a fairly straightforward memory decoding problem. The 32-bit address bus needs to be split up and decoded to choose which memory component is which. And, despite using the work *choose* in the previous sentence, we will not be using a multiplexer. Decoders are the name of this game.

   First, we have to examine the range of each of the memory components:

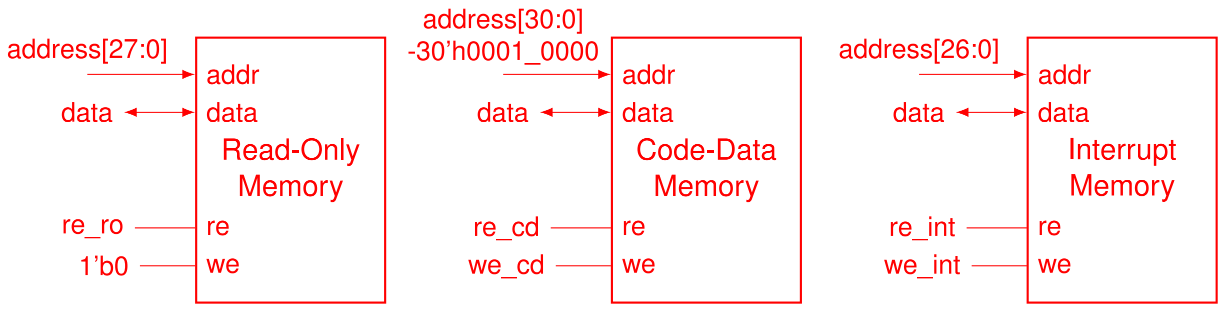
   (a) The Read-Only memory has an address range from **`32'h0000_0000`** to **`32'h0FFF_FFFF`**. That means, it only has 2^{28} words of memory and has only a 28-bit address bus. Those 28-bits will be connected to the least-significant 28-bits from the memory system address bus. And, it will only be enabled when the top 4-bits of the address bits are all zero.

   (b) The Code and Data memory has a 31-bit address bus. It will only be enabled when **`address[31:27]`** is a **`5'b0001_0`**, **`5'b0001_1`** ... **`5'b1000_1`**. Our decoder will be quite useful to watch for all these possibilities. Notice also that we will have to shift the addresses a bit. When the memory system has **`32'h1000_0000`** on the address bus, this component should be accessing the word at address **`31'h0000_0000`** in its internal memory array. Therefore, we will need to subtract **`31'h0001_0000`** from the memory system address bus to feed to this component's address bus. BUT, it turns out you can ignore this subtracter if you're willing to have the internal memory contents not be contiguous. Therefore, this subtracter won't be held against you if you didn't include it.

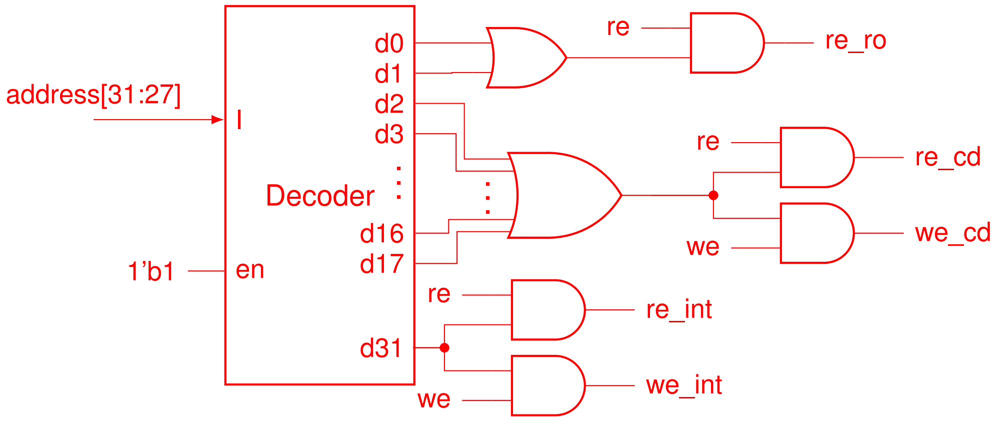
   (c) The Interrupt memory has an 27-bit address bus. It requires all the bits of **`address[31:27]`** to be ones in order to operate.

   The **`Memory`** components we've looked at in class don't have an enable input. They do have read- and write-enables, however. We will ensure that the **`re`** and **`we`** signals for each component only get asserted when the address range matches that we've looked at above.

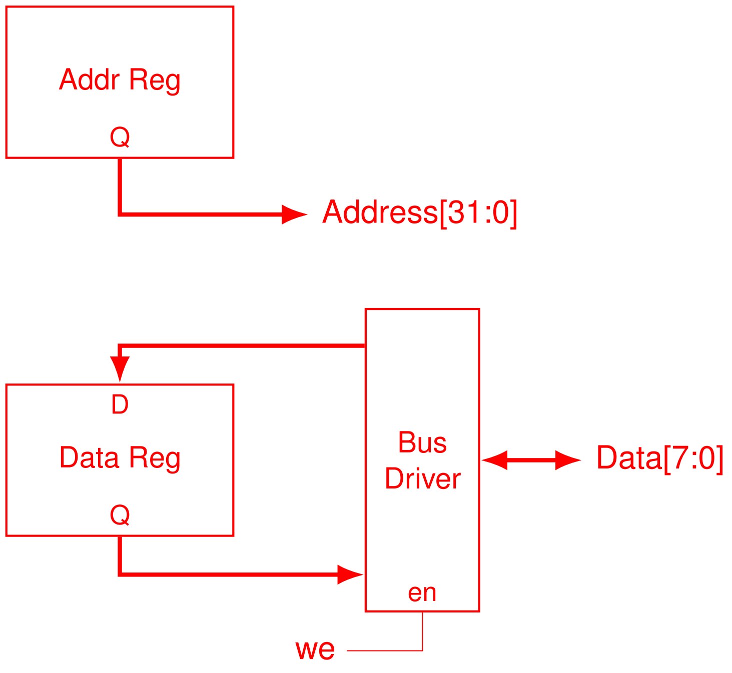
Here's the schematic for the memory components:



Note the read and write enables for each memory component are different. We need to ensure that only the correct component is enabled. For this, we will use a decoder. We will combine the decoder outputs with `re` and `we` to enable each separate component. This gets annoying, with lots of AND gates. For this reason, most memory components – especially those sold in separate chips – have an enable input.



One final detail is to figure out how the user's registers drive the address and data busses. This is fairly easy, but don't forget that we need a bus driver on the data bus:



3. [20 points, Lecture 15] Design a datapath to calculate a final grade for some non-240 course. The grade is calculated from a weighted average, based on individual homework, lab, exam and class participation grades.

Let's follow the design steps for a hardware thread.

(a) **Data Elements:** We will need to keep track of the sum of each type of score. Therefore, we need **`sum_hw`**, **`sum_lab`**, **`sum_exam`**, **`sum_cp`** registers. It might be necessary to have a register keep track of the total overall score, so let's tentatively include a **`sum_scores`** register as well.

(b) **Transformations:** We will need to detect score input type, so some form of $type =? = homework$.

We also need to be able to update each score: $sum\_hw \Leftarrow sum\_hw + score$, etc.

We need to be able to calculate an overall sum: $sum\_scores \Leftarrow \frac{1}{4}sum\_hw + \frac{1}{4}sum\_lab + \frac{3}{8}sum\_exam + \frac{1}{8}sum\_cp$

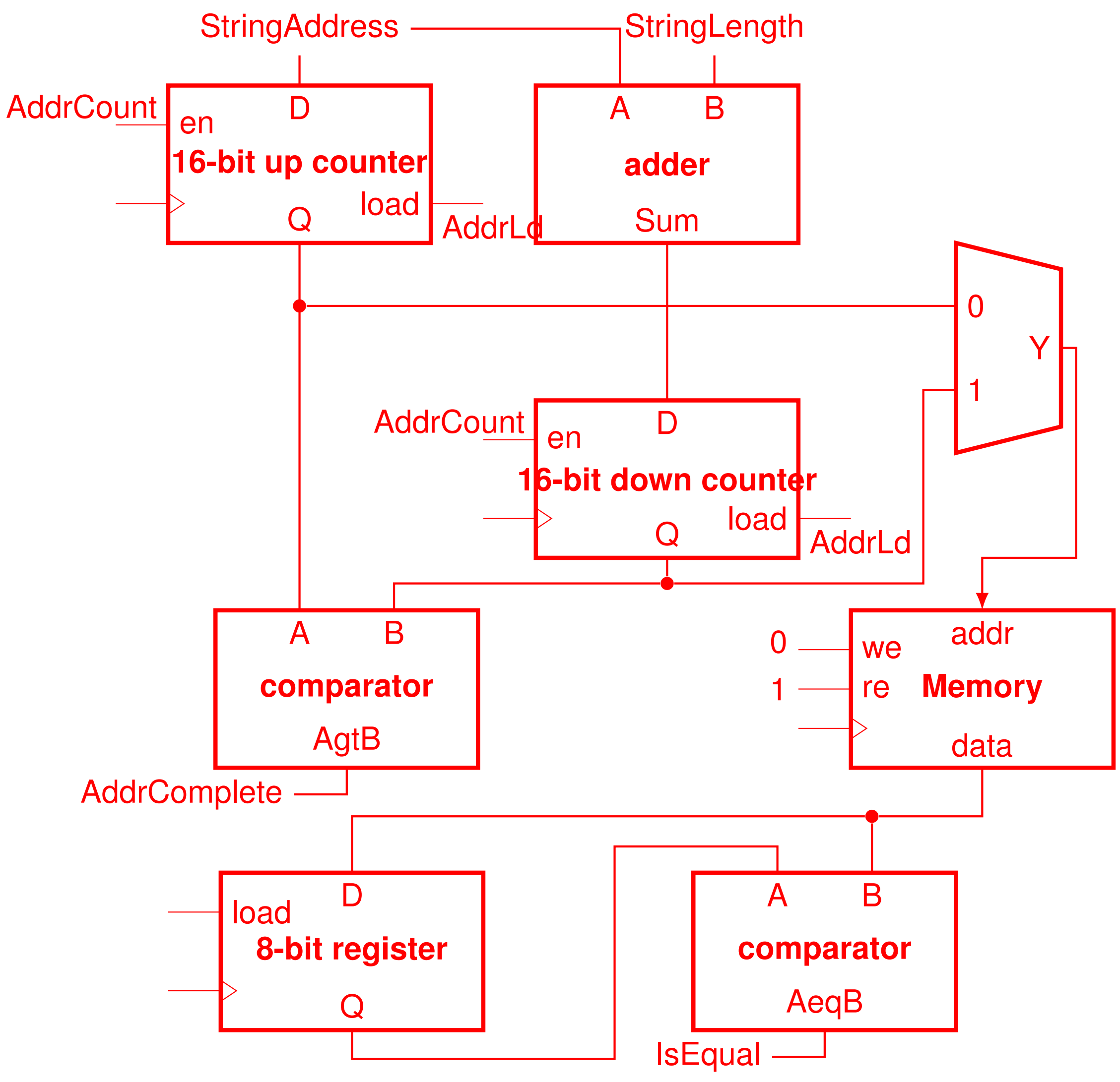
And, we need to test and compare to get the final scores: $sum\_score \geq 90$, $90 > sum\_score \geq 80$, etc.

For initialization, each register will also need to be able to be cleared: *sum_hw* $\Leftarrow$ 0, etc.

(c) **Datapath design:** Based on the transformations from the previous step, it seems we will need the following combinational circuitry:

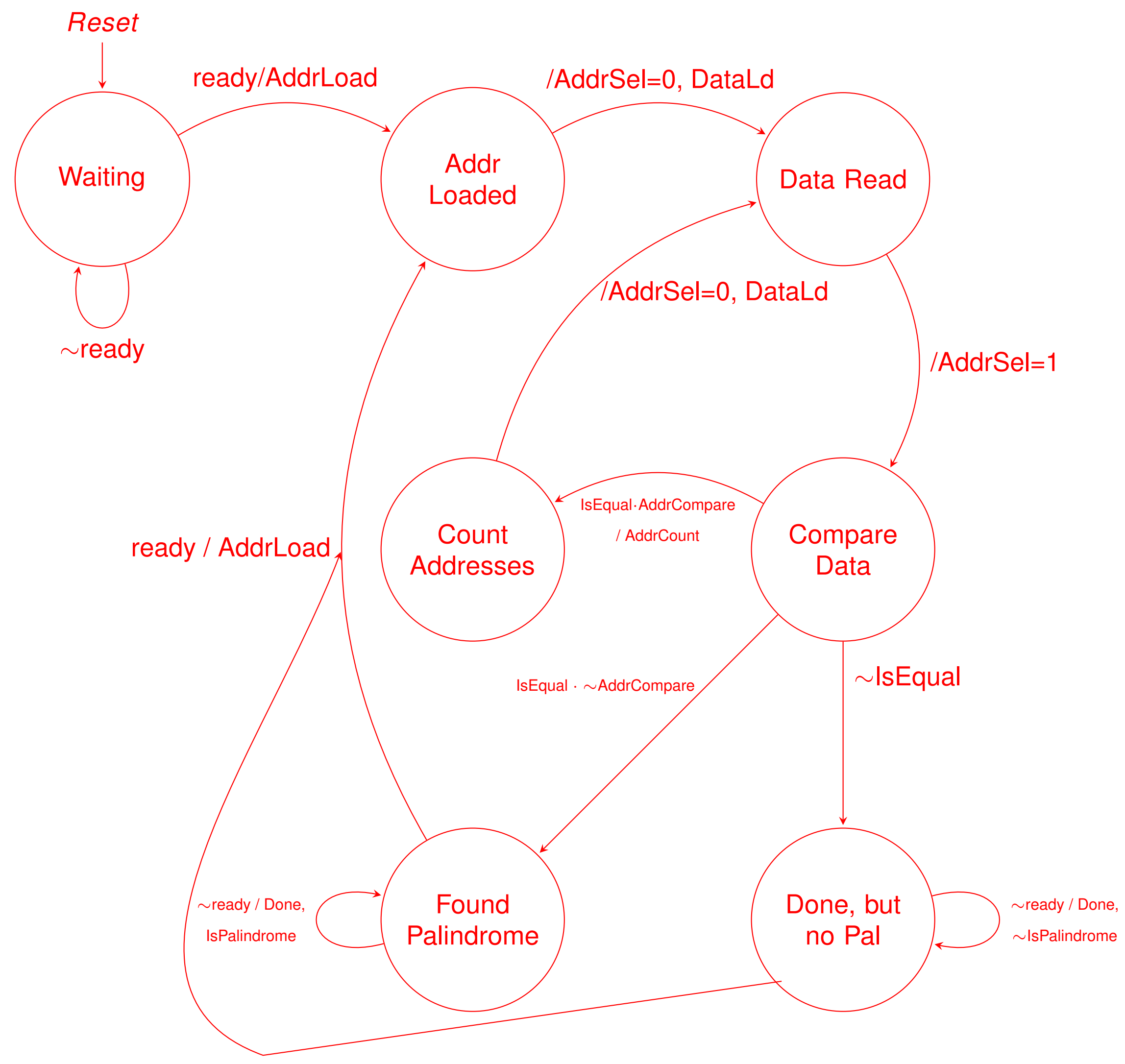
- A decoder to detect the score type.
- Lots of adders
- Some shifters to do the bottom part of the fractions (divide by 4 and 8).
- Some comparators to determine the grade outputs.

Here is my schematic diagram:



(d) **Control and Status points:** listed here

(e) **Control FSM:** Here's my FSM.



(f) **SystemVerilog Implementation:** You can find **`hw7prob3.sv`** on Canvas.

4. [6 points, Lecture 15] Lecture #15, contained a block diagram of a 1Gb DDR SDRAM chip. It is organized as 256M x 4-bits, with a multiplexed address bus.

(a) A chip that can address 256M (that is $256 * 2^2 0$) different values must have $\log^2 256 * 2^2 0$ or 28 address lines. But, since the address bus is multiplexed, there only need to be 14 address lines, since half will be sent one one clock edge and the other half following.

(b) Each of the SDRAM chips has 1Gbit. 16GBytes is $16 * 8$ times the size, so I will need 128 of those chips.

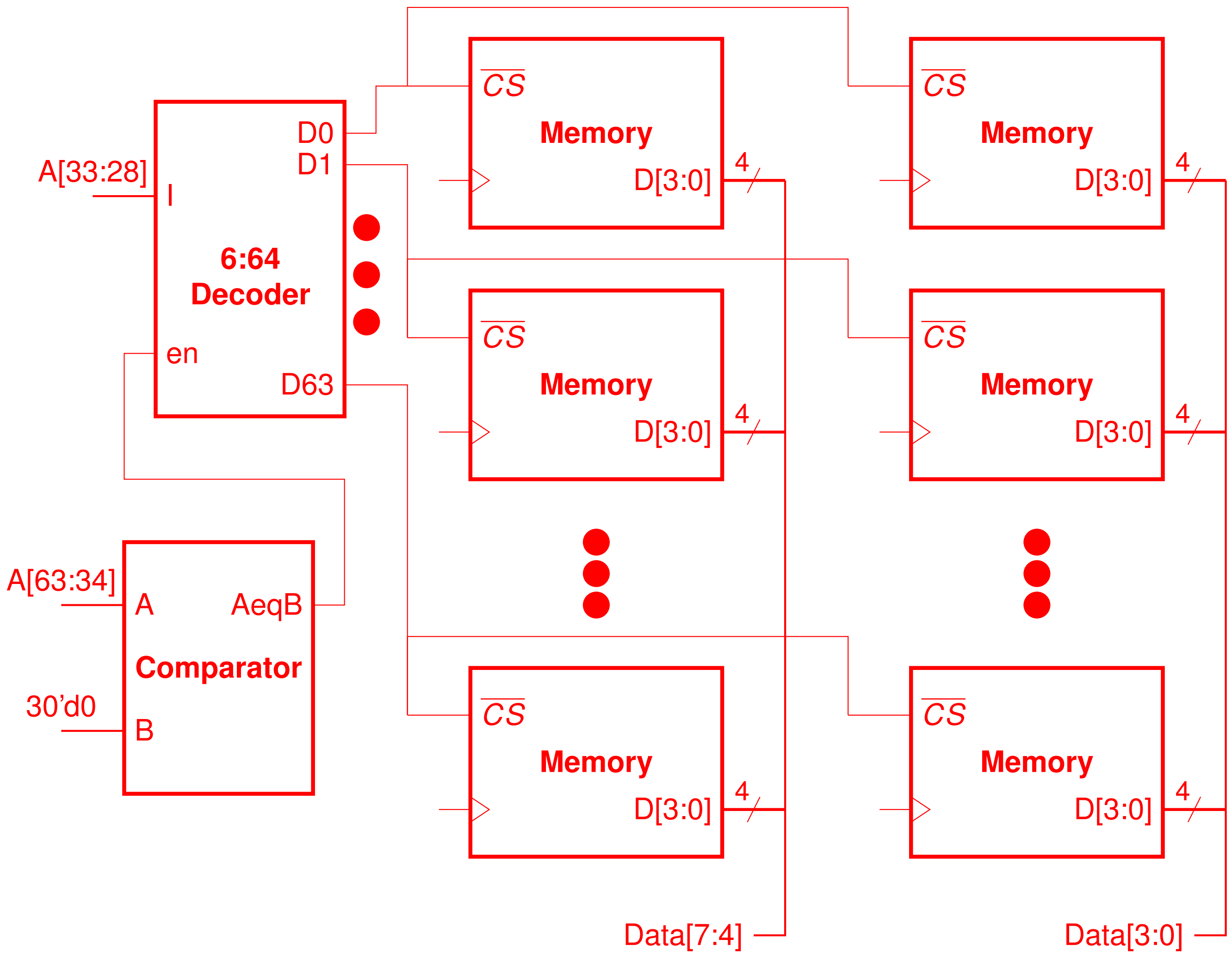
(c) I am assuming that my 16GBytes occurs at the bottom range of the 2^64 address range (that is, physical addresses 0 through `64'h0000_0003_FFFF_FFFF`). Two chips are needed to provide 8-bits of data bus, and in doing so provide 256Mx8 bits of memory. The address range for the first two chips is then 0 through `64'0000_0000_0FFF_FFFF`. So, those two chips should have their chip select ($\overline{CS}$) active whenever address bits `[63:28]` are all zeros. Whew, $\frac{1}{64}$th of the way done.

The next two chips will provide 8-bits of data bus for the next 256M of memory, from `64'h0000_0000_1000_0000` through `64'h0000_0000_1FFF_FFFF`. We will activate their chip selects when `Address[63:28] == 36'b1`.

Now, we can see the structure of these 128 memory chips. There will be 64 pairs, where one of each pair is connected to `Data[7:4]` and the other connected to `Data[3:0]`. Both will be selected together, thus their ($\overline{CS}$) inputs are connected.

I used a 6:64 Decoder with one-cold outputs. It is enabled when `Address[63:34] == 1'b0` (that's the comparator). Each output is then connected to the chip select of one pair of memory chips.

Other control lines like $\overline{WE}$, $\overline{RAS}$, and $\overline{CAS}$ would be connected to all of the chips, though I haven't shown it in this diagram for simplicity sake.



A[33:28]
I
6:64
Decoder
en
D0
D1
D63
A[63:34]
A
AeqB
Comparator
30'd0
B
$\overline{CS}$
Memory
D[3:0]
4
Data[7:4]
Data[3:0]