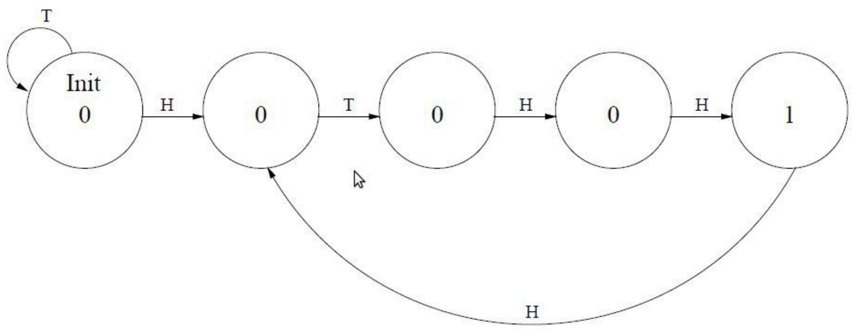
Homework03

1. Shown below is a partially completed state diagram of a finite state machine that takes an input string of H (heads) and T (tails) and produces an output of 1 every time the string HTHH occurs.

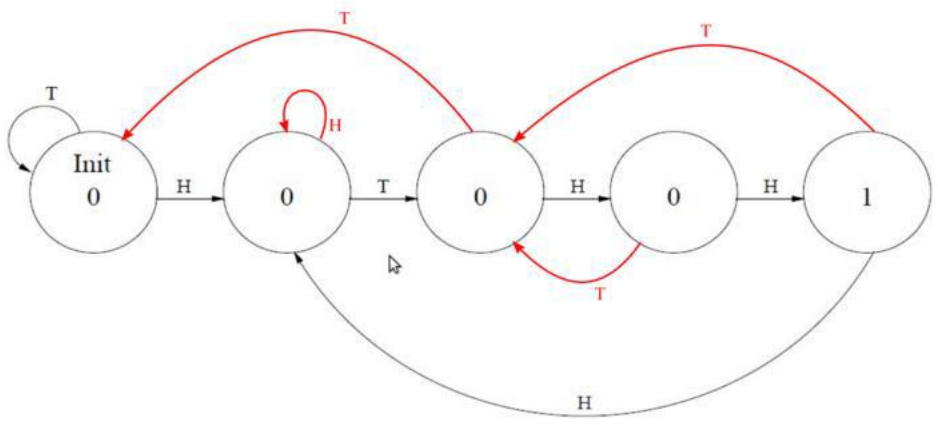


For example,

if the input string is: H H H H H T H H T H H H H H T H H T,
the output would be: 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0.

Note that the 8th coin toss (H) is part of two HTHH sequences.

a. Complete the state diagram of the finite state machine that will do this for any input sequence of any length



b. If this state machine is implemented with a sequential logic circuit how many state variables will be needed?

3bits.

2. If a particular computer has 16 byte addressability and a 7 bit address space, how many bytes of memory does that computer have?

2^7*16=2048(bytes)

3. Using Figure 3.21 on page 69 in the book, the diagram of the 4-entry, 2^2-by-3-bit memory.

   a. To write the second memory location, what must the values of A[1:0] and WE be?

   To read from the second location A[1:0] should be 01, to read from memory the WE bit should be 1.

   b. To change the number of locations in the memory from 4 to 800, how many address lines would be needed? What would the addressability of the memory be after this change was made?

   To address 800 locations you need 10 bits of address line, which means your MAR is 10 bits . However since we did not change the number of bits stored at each location the addressability is still 3 bits.

   c. Suppose the width (in bits) of the program counter is the minimum number of bits needed to address all 800 locations in our memory from part (b). How many additional memory locations could be added to this memory without having to alter the width of the program counter?

   You need 10 bits for part b, which can address 1024 different locations so you could add 224 more locations and not have to increase the width of the program counter.

4. The figure below is a diagram of a 2^2-by-16-bit memory, similar in implementation to the memory of Figure 3.21 in the textbook. Note that in this figure, every memory cell represents 4 bits of storage instead of 1 bit of storage. This can be accomplished by using 4 Gated-D Latches for each memory cell instead of using a single Gated-D Latch. The hex digit inside each memory cell represents what that cell is storing prior to this problem.

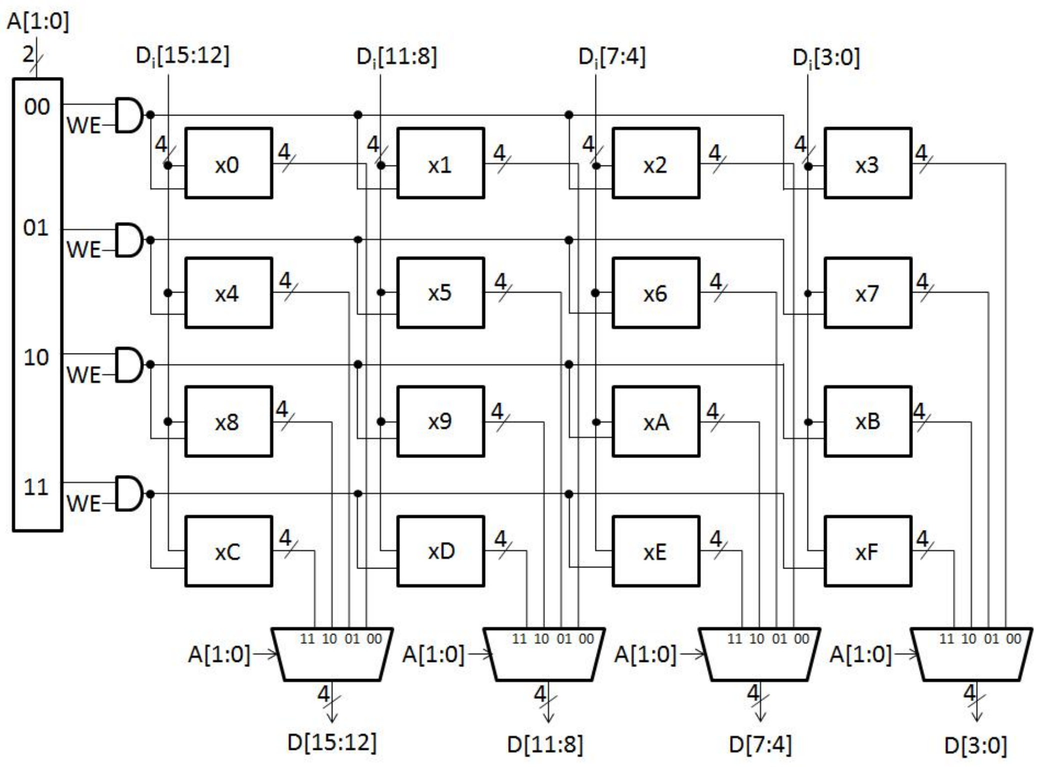


Figure 3: 2^2-by-16 bit memory

a. What is the address space of this memory?

2^2=4 memory locations.

b. What is the addressability of this memory?

16 bits.

c. What is the total size in bytes of this memory?

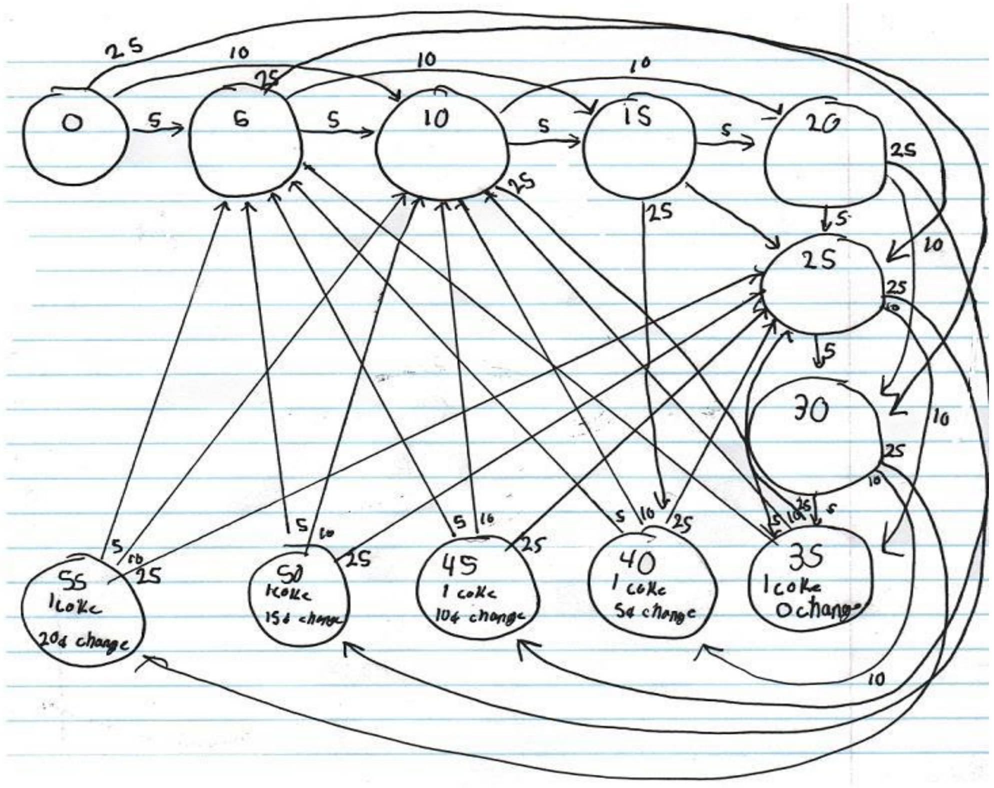
8 bytes.

d. This memory is accessed during four consecutive clock cycles. The following table lists the values of some important variables just before the end of the cycle for each access. Each row in the table corresponds to a memory access. The read/write column indicates the type of access: whether the access is reading memory or writing to memory. Complete the missing entries in the table.

WE	A[1:0]	Di[15:0]	D[15:0]	Read/Write

0	01	xFADE	x4567	Read
1	10	xDEAD	xDEAD	Write
0	00	xBEEF	x0123	Read
1	11	xFEED	xFEED	Write

5. The USTC west campus sells sodas for 35 cents. Suppose they install a soda controller that only takes the following three inputs: nickel, dime, and quarter. After you put in each coin, you push a pushbutton to register the coin. If at least 35 cents has been put in the controller, it will output a soda and proper change (if applicable). Draw a finite state machine that describes the behavior of the soda controller. Each state will represent how much money has been put in (Hint: There will be seven of those states). Once enough money has been put in it, the controller will go to a final state where the person will receive a soda and proper change (Hint: There are five such final states). From the final state, the next coin that is put in will start the process again, contributing to the next purchase.



6. Suppose that an instruction cycle of the LC-3 has just finished and another one is about to begin. The following table describes the values in select LC-3 registers and memory locations:

Register	**Value**
IR	x3001
PC	x3003
R0	x3000
R1	x3000
R2	x3002
R3	x3000
R4	x3000
R5	x3000
R6	x3000
R7	x3000

Memory Location	**Value**
x3000	x62BF
x3001	x3000
x3002	x3001
x3003	x62BE

For each phase of the new instruction cycle, specify the values that `PC`, `IR`, `MAR`, `MDR`, `R1`, and `R2` will have *at the end* of the phase in the following table:

	PC	IR	MAR	MDR	R0	R1	R2	R3	R4	R5	R6	R7
Fetch	x3004	x62BE	X3003	X62BE	X3000	X3000	X3002	X3000	X3000	X3000	X3000	X3000
Decode	x3004	x62BE	X3003	X62BE	X3000	X3000	X3002	X3000	X3000	X3000	X3000	X3000
Evaluate Address	x3004	x62BE	X3003	X62BE	X3000	X3000	X3002	X3000	X3000	X3000	X3000	X3000
Fetch Operands	x3004	x62BE	X3000	X62BF	X3000	X3000	X3002	X3000	X3000	X3000	X3000	X3000
Execute	x3004	x62BE	X3000	X62BF	X3000	X3000	X3002	X3000	X3000	X3000	X3000	X3000
Store Result	x3004	x62BE	X3000	X62BF	X3000	X62BF	X3002	X3000	X3000	X3000	X3000	X3000

Hint: Example 4.2 on page 104 illustrates the LDR instruction of the LC-3. Notice that values of memory locations x3000, and 3003 can be interpreted as LDR instructions.

7. Suppose a 32-bit instruction has the following format:

OPCODE	DR	SR1	SR2	UNUSED

If there are 1511 opcodes and 40 registers, and every register is available as a source or destination for every opcode,

a. What is the minimum number of bits required to represent the OPCODE?

1511 opcode, 11 bits are required to represent the OPCODE

b. What is the minimum number of bits required to represent the Destination Register (DR)?

40 registers, 6 bits to represent the DR

c. What is the maximum number of UNUSED bits in the instruction encoding?

3 registers and 1opcode, 3*6 + 11 = 29 bits.So there are 3 unnsed bits

8. Write a program in LC-3 machine language that loads a 1 into R0 if the value in R1 is identical to the value in R3, and loads a 2 into R0 if the values in R1 and R3 are different.

```
0101000000100000 ; R0 <-0
1001011010111111 ; R3 <-NOT(R2)
0001011011100001 ; R3 <-R3 + 1; in effect, makes R3 <--R2
0001001001000011 ; R1 <-R1 + R3; in effect, makes R1 <-R1 -R2
0000101000000001 ; branch to the last instruction if negative or positive
0001000000100001 ; R0 <-R0 + 1 (makes R0 1 instead of 0)
1111000000100101 ; HALT
```

9. What does the following program do (in 20 words or fewer):

```
0101 100 100 1 00000
1001 000 001 111111
0001 000 000 1 00001
0001 000 000 000 010
0000 100 000000001
0001 100 100 1 00001
1111 0000 0010 0101
```

Makes R4 <-1 if R2 >= R1; else, makes R4 <-0.