Homework 03

和泳毅 PB19010450

1. We want to make a state machine for the scoreboard of the Texas vs. Oklahoma Football game. The following information is required to determine the state of the game:

   1) Score : 0 to 99 points for each team

   2) Down : 1, 2, 3, or 4

   3) Yards to gain : 0 to 99

   4) Quarter : 1, 2, 3, 4

   5) Yardline : any number from Home 0 to Home 49, Visitor 0 to Visitor 49, 50

   6) Possesion : Home, Visitor

   7) Time remaining: any number from 0:00 to 15:00, where m:s (minutes, seconds)

   (a) What is the minimum number of bits that we need to use to store the state required?

   (b) Suppose we make a separate logic circuit for each of the seven elements on the scoreboard, how many bits would it then take to store the state of the scoreboard? (c) Why might the method of part b be a better way to specify the state than the method of part a?

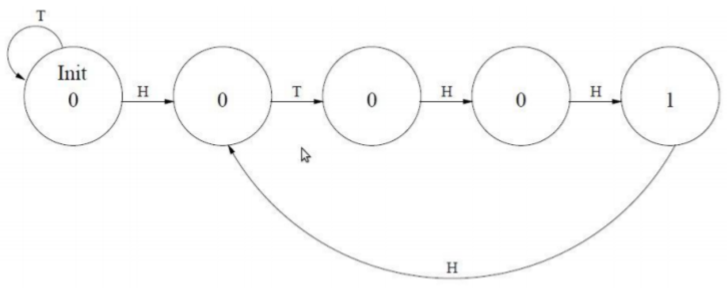
Answer:

$$(100 \times 100) \times 4 \times 100 \times 4 \times 101 \times 2 \times 901 = 2912032000000$$
$$2^{41} < 2912032000000 < 2^{42}$$

So we need **42** bits.

---

2. Shown below is a partially completed state diagram of a finite state machine that takes an input string of H (heads) ant T (tails) and produces an output of 1 every time the string HTHH occurs. Figure 4



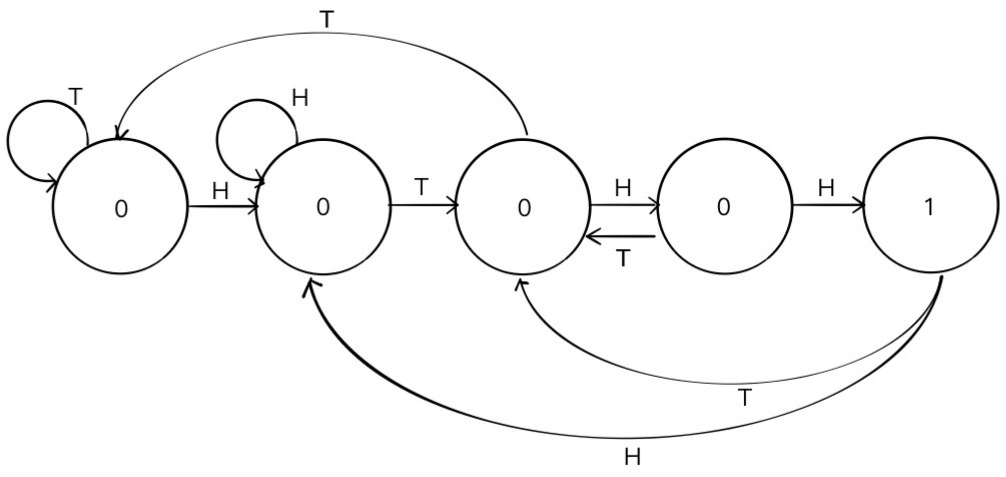
For example,

```
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 8^{th} 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

Answer:



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

Answer:

We need **3** state variables to implement this state machine with a sequential logic circuit .

---

3. (3.37)

   If a particular computer has 8 byte addressability and a 4 bit address space, how many bytes of memory does that computer have?

   Answer:

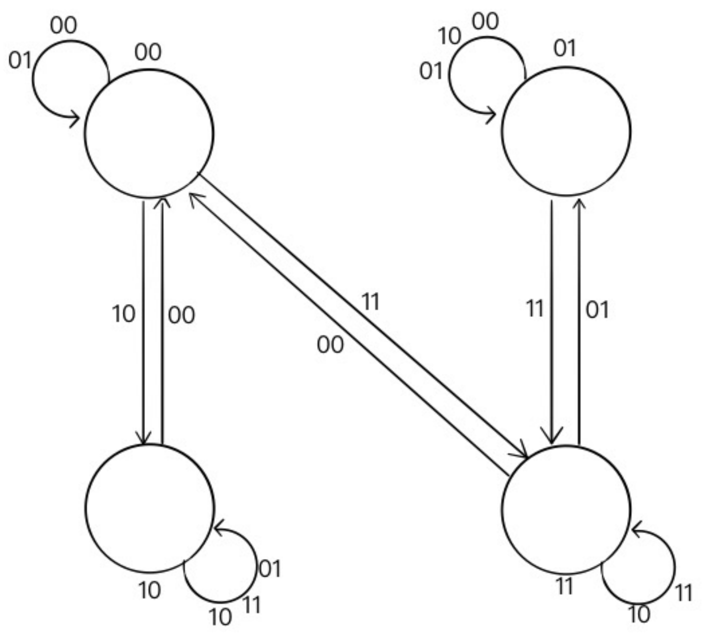
   $8 \times 2^4 = 128 \quad bytes$

---

4. ***Elevator Problem Revisited***

   Recall the elevator controller problem on Problem Set 2. You were asked to design the truth table for an elevator controller such that the option to move up or down by one floor is disabled. If there is a request to move only one floor or to move zero floors, the elevator should remain on the current floor. For this problem, you will design the state machine for the sequential logic circuit for an elevator controller which performs the same operation. You can assume that the building the elevator is in has 4 floors. The input to the state machine is the next requested floor. There will be a state for each floor the elevator could be on. Draw a finite state machine that describes the behavior of the elevator controller. How many bits are needed for the inputs?

   Answer:



00--Floor1 01--Floor2 10--Floor3 11--Floor4

Only **2** bits are enough.

5. (3.33)

   Using Figure 3.21 on page 78 in the book, the diagram of the, 2^2- by-3-bit memory.

   a. To read from the fourth memory location, what must the values of ***A[1:0]*** and ***WE*** be?

   Answer: A=11 WE=0

   b. To change the number of locations in the memory from 4 to 60, how many address lines would be needed? What would the addressability of the memory be after this change was made?

   Answer: **6** address lines would be needed. The addressability is **3**.

   c. Suppose the width (in bits) of the program counter is the minimum number of bits needed to address all 60 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?

   Answer: **4** additional memory locations could be added.

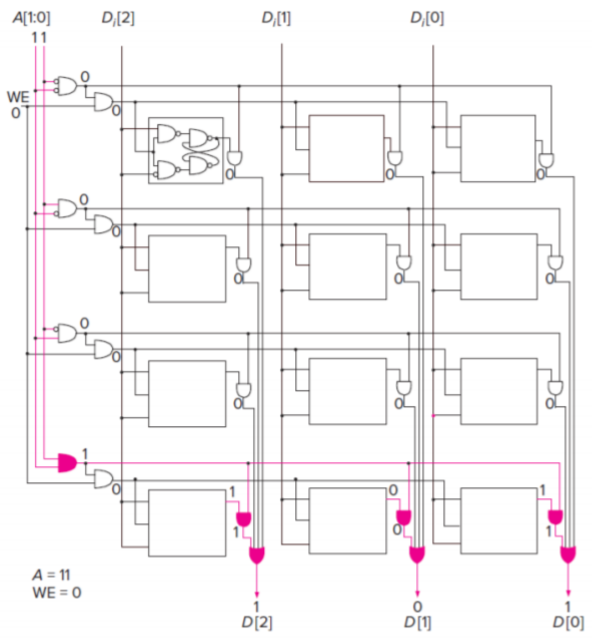


Figure 3.21 Reading location 3 in our 2^2-by-3-bit memory.

6. 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.

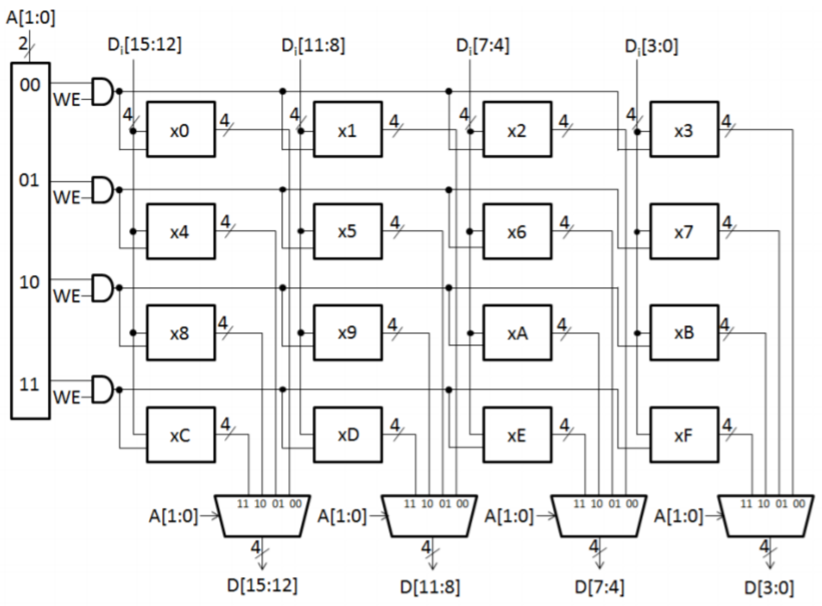


Figure 3: 2^2-by-16 bit memory

a. What is the address space of this memory?

Answer: **2 bits** address space of this memory.

b. What is the addressability of this memory?

Answer: **16 bits** addressability of this memory.

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

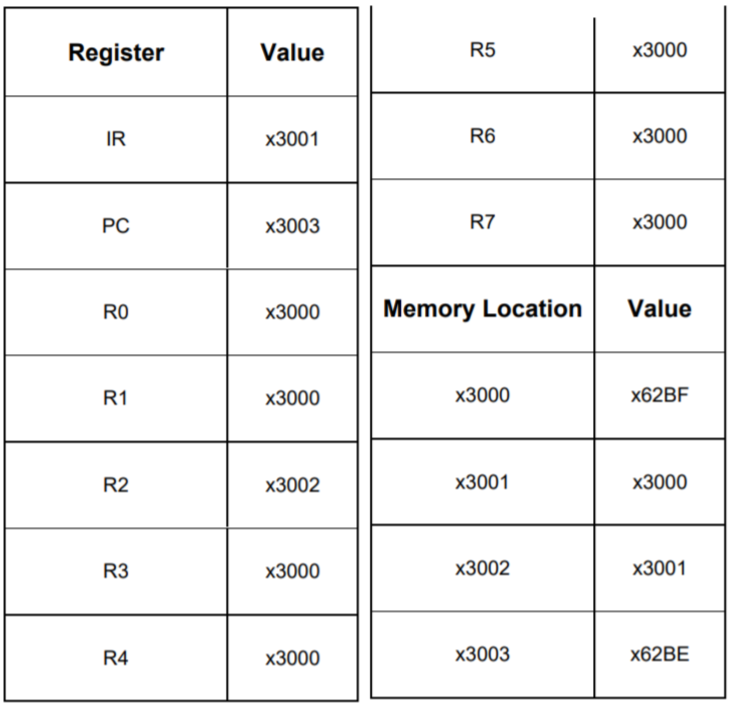
Answer: $2^2 \times 2 = 8 \quad bytes$

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[0:1]	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

---

7. 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: Notice that values of memory locations x3000 and x3003 can be interpreted as LDR instructions.

The instruction from PC: `x62BE` --> `0110 001 010 111110`

0110	**001**	**010**	**111110**
LDR	R1	R2	x3E

Sign-extend bits [5:0] to 16 bits: `111110` --> `1111 1111 1111 1110` --> `xFFFE`

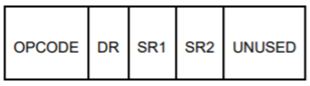
Add the contents of R2 to the sign-extended value contained in IR [5:0]:

`x3002` + `xFFFE` = `x3000`

---

8. (4.8)

Suppose a 32-bit instruction has the following format:



If there are 255 opcodes and 120 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?

Answer:

$2^7 < 255 < 2^8$

So if there are 255 opcodes, **8** bits are required to represent the OPCODE.

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

Answer:

$2^6 < 120 < 2^7$

So if there are 25, **7** bits are required to represent the Destination Register (DR).

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

Answer:

$32 - 8 - 7 - 7 - 7 = 3$

So there are **3** ununsed bits

---

9. ***A State Diagam***

We wish to invent a two-person game, which we will call XandY that can be played on the computer. Your job in this problem is contribute a piece of the solution.

The game is played with the computer and a deck of cards. Each card has on it one of four values (X, Y, Z, and N). Each player in turn gets five attempts to accumulate points. We call each attempt a round. After player A finishes his five rounds, it is player B's turn. Play continues until one of the players accumulates 100 points. Your job today is to ONLY design a finite state machine to keep track of the STATE of the current round. Each round starts in the intial state, where X=0 and Y=0. Cards from the deck are turned over one by one. Each card transitions the round from its current state to its next state, until the round terminates, at which point we'll start a new round in the initial state.

The transistions are as follows:

X: The number of X's is incremented, producing a new state for the round.

Y: The number of Y's is incremented, producing a new state for the round.

Z: If the number of X's is less than 2, the number of X's is incremented, producing a new state for the round. If the number of X's is 2, the state of the current round does not change.

N: Other information on the card gives the number of points accumulated. N also terminates the current round.

Important rule: If the number of X's or Y's reaches a count of 3, the current round is terminated and another round is started. When a round starts, its state is X=0, Y=0.

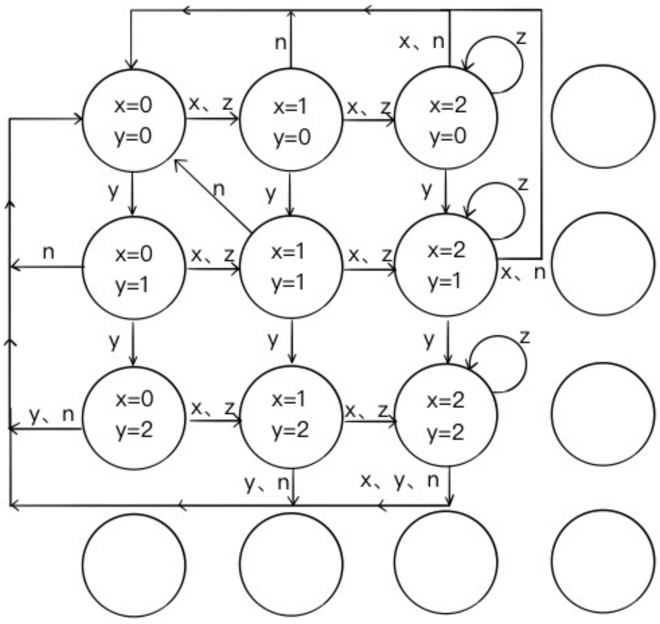
Hint: Since the number of X's and Y's specify the state of the current round, how many possible states are needed to describe the state of the current round.

Hint: A state cannot have X=3, because then the round would be finished, and we would have started a *new* current round.

On the diagram below, label each state. For each state draw an arrow showing the transition to the next state that would occur for each of the four inputs. (We have provided sixteen states. You will not need all of them. Use only as many as you need).

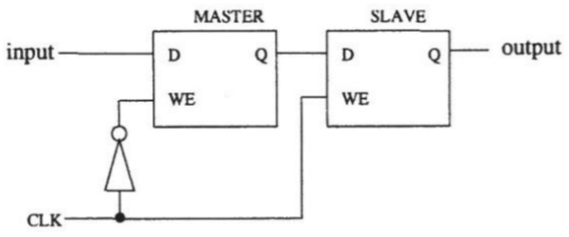
Note, we did not specify outputs for these states. Therefore, your state machine will not include outputs. It will only include states and transistions represented by inputs.

Answer:

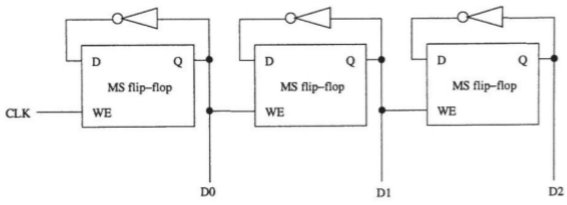


10. ***Trying Out Flip-Flops***

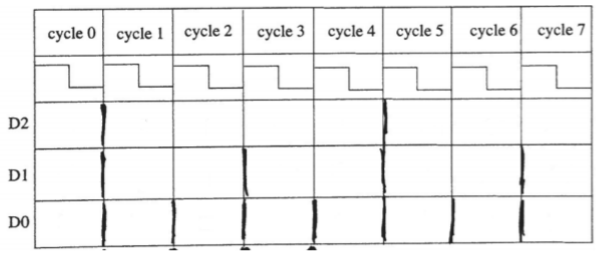
The Master-Slave flipflop we introduced in class is shown below.



Note that the input value is visible at the output after the clock transitions from 0 to 1. Shown below is a circuit constructed with three of these flipflops.



Your job: Fill in the entries for D2, D1, D0 for each of clock cycles shown: (In Cycle 0, all three flip-flops hold the value 0)



	cycle 0	**cycle 1**	**cycle 2**	**cycle 3**	**cycle 4**	**cycle 5**	**cycle 6**	**cycle 7**
D2	0	1	1	1	1	0	0	0
D1	0	1	1	0	0	1	1	0
D0	0	1	0	1	0	1	0	1

In 10 words or less, what is this circuit doing?

Answer:

It's doing auto-decrement operating.

---

11. 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
```

Answer:

```
0101 100 100 1 00000    R1 <- R1 AND (00000)    [R1=0000 0000 0000 0000]
1001 000 001 111111     R0 <- NOT R1            [R0=1111 1111 1111 1111]
0001 000 000 1 00001    R0 <- R0 + 1            [R0=0000 0000 0000 0000]
0001 000 000 000 010    R0 <- R0 + R2           [R0=R2]
0000 100 000000001      N = 1
0001 100 100 1 00001    R4 <- R4 + (00001)
1111 0000 0010 0101     HALT THE PROGRAM
```

IF R2>=R1 DO R4 <- 1, ELSE DO R4 <- 0.

---

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

    0101 000 000 1 00000

    0101 101 001 1 00001

    0000 101 000000001

    0001 000 000 1 00001

    1111 0000 0010 0101

Answer:

```
0101 000 000 1 00000    R0 <- R0 AND (00000)    [R0=0000 0000 0000 0000]
0101 101 001 1 00001    R5 <- R1 AND (00001)    [R5=0000 0000 0000 000?]
0000 101 000000001      N = 1, P = 1
0001 000 000 1 00001    R0 <- R0 + 1
1111 0000 0010 0101     HALT THE PROGRAM
```

IF R1 is odd DO R0 <- 0, ELSE IF R1 is even DO R0 <- 1.
