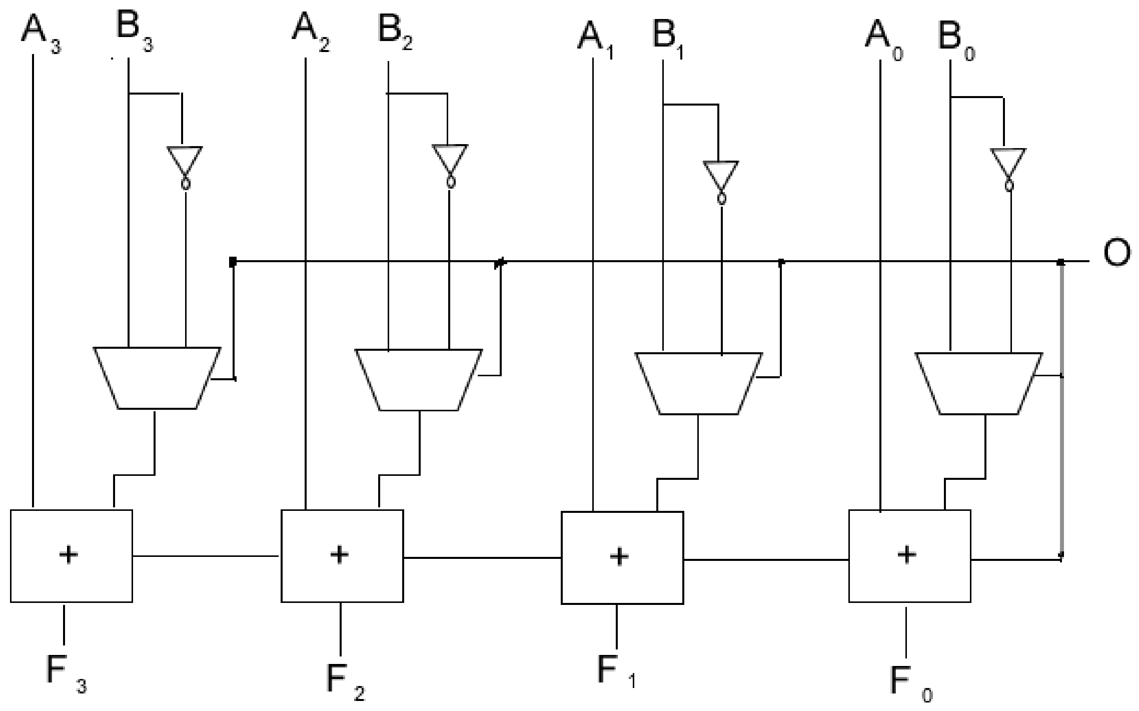
CS 252 Spring 2008 Homework 4

1a.



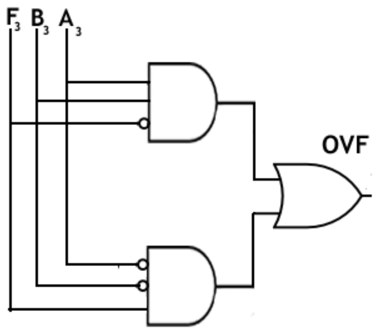
When O is 0, the MUXes choose the B inputs directly so the sum obtained is (A+B). When O is 1, the MUXes choose the negated bits of B (i.e. 1's complement of B). Also the carry-in of the left most adder is set to 1, the net effect of which is that the 2's complement of B is computed, and the result is A+(-B) = A - B.

b. An overflow occurs when the most significant bits of the input are the same, but the MSB of the output differs from the input. When the MSBs of the inputs are different, no overflow can happen. So we can detect overflows by looking at A3,B3 and F3

A3	B3	F3	OVF
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0

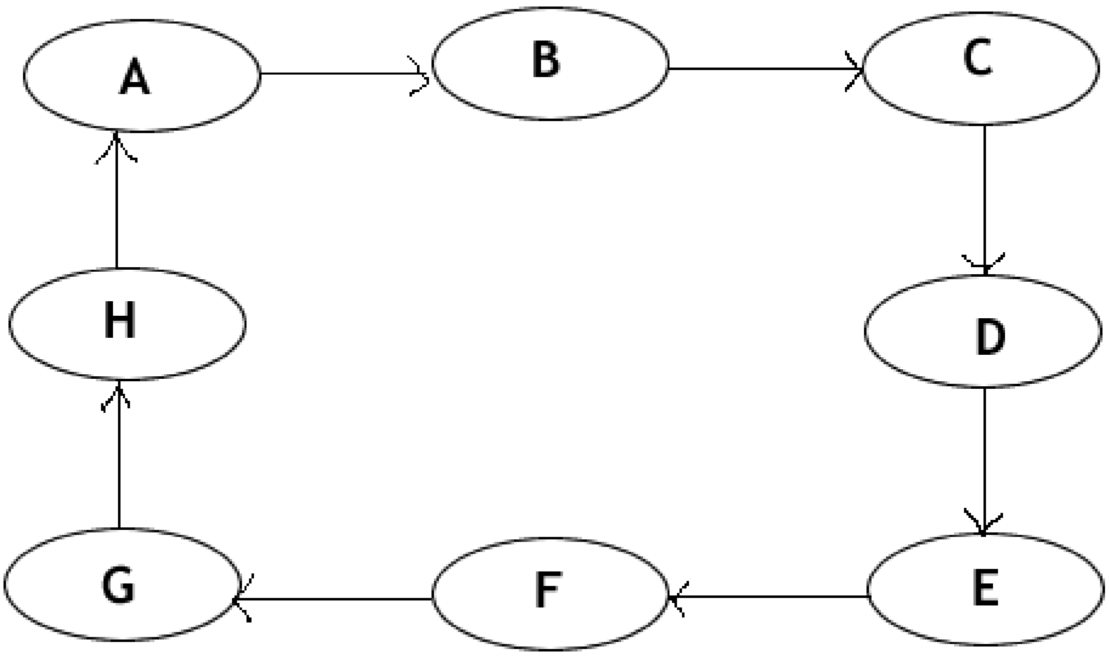
1	0	1	0
1	1	0	1
1	1	1	0

The expression corresponding to OVF is **(A3 and B3 and not F3) OR (not A3 and not B3 and F3)**



2

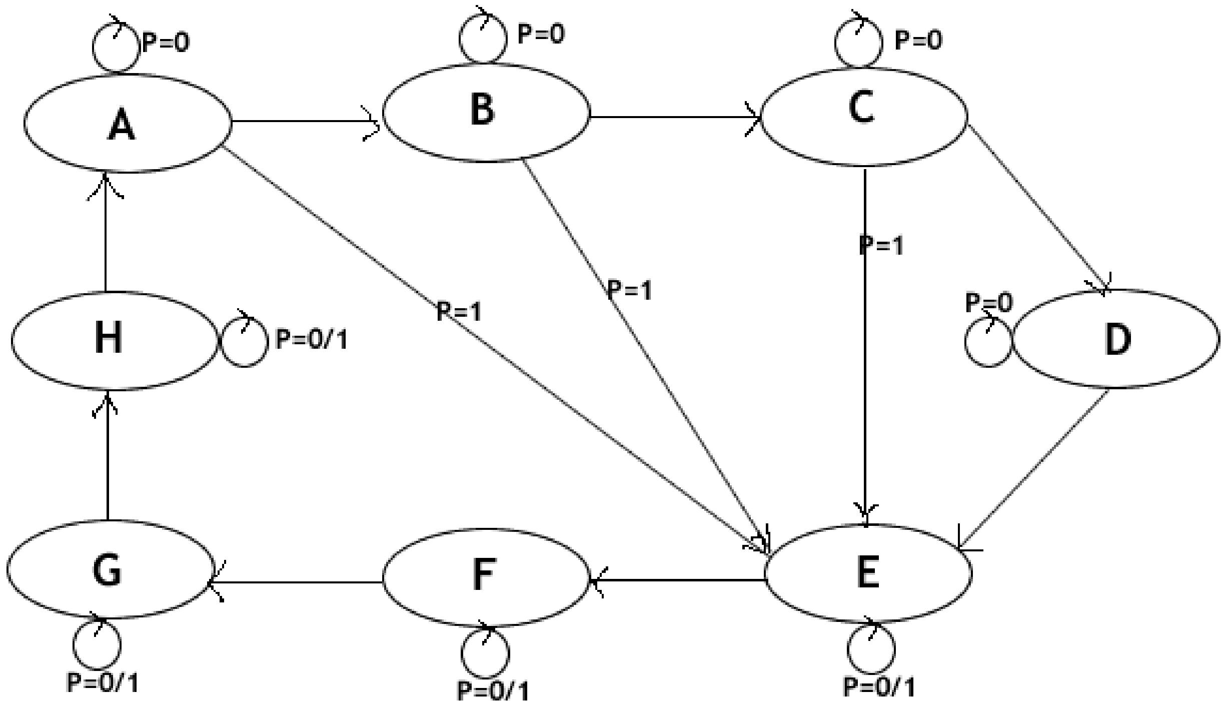
a.



The output for each of the states is:

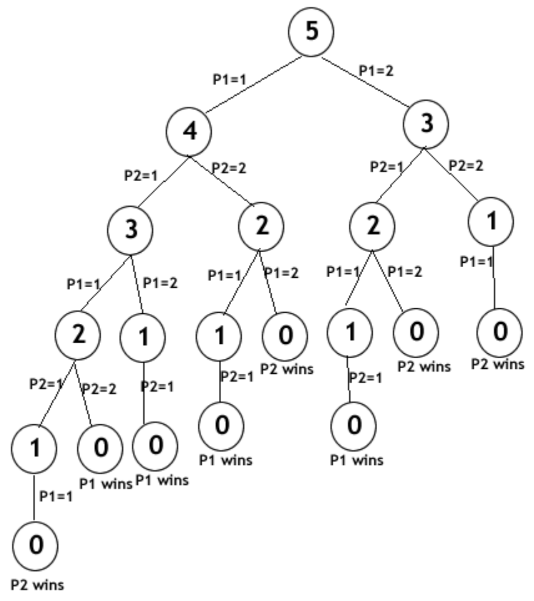
States	Output
A B C D	00110
E	01010
F G	10001
H	10010

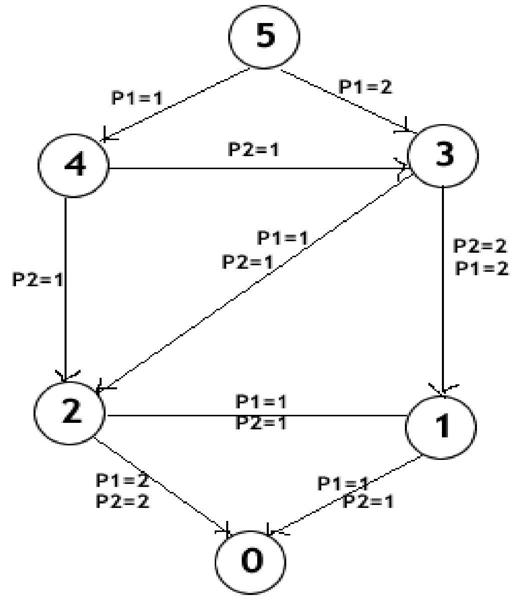
b.



We add a new input **P** corresponding to the Pedestrian button. If vehicle light is green, (states A ,B ,C ,D) the state changes to amber (state E) if the button is pressed. In all other cases the button is ignored.

3.





Each state corresponds to the number of balls left in the pile. The game is over when all the balls are gone, i.e. state is ‘0’. Each transition is labeled by player’s choice: ‘P1=1’ means Player 1 picked 1 ball from the pile.

4. The idea of universality still holds. Consider two machines A and B with A’s word length being greater than B’s. A can do all the operations that B can do, by ignoring the extra bits. What about B? Well, B can simulate A’s operation by carrying out multiple operations which achieve the same result. Taking addition for example, B can add numbers of size ‘a’ (A’s word length) by carrying out two or more additions, taking ‘b’(B’s word length) bits at a time, making sure that the carry is propagated correctly between additions. Similarly all operations that A can do, B can simulate.

5.

1. Type1 = 100%, Type2 = 0%, Type3 = 0%

   “Average” number of cycles per instruction

   = (100 * 1 + 0 * 20 + 0 * 500)/ 100 = 1

   Number of instructions per second = (2 Billion) / 1 = 2 Billion
2. Type1 = 95%, Type2 = 4%, Type3 = 1%

   “Average” number of cycles per instruction

   = (95 * 1 + 4 * 20 + 1 * 500)/ 100 = 6.75

Number of instructions per second = (2 Billion) / 6.75 = 0.296 Billion

3. Type1 = 95%, Type2 = 1%, Type3 = 4%

   "Average" number of cycles per instruction

   = (95 * 1 + 1 * 20 + 4 * 500)/ 100 = 21.15

   Number of instructions per second = (2 Billion) / 21.15 = 0.094 Billion

4. Type1 = 70%, Type2 = 20%, Type3 =10%

   "Average" number of cycles per instruction

   = (70 * 1 + 20 * 20 + 10 * 500)/ 100 = 54.7

   Number of instructions per second = (2 Billion) / 54.7 = 0.037 Billion

6.

- Fetch – The next instruction is loaded from memory into the Instruction Register of the Control Unit
- Decode – This phase evaluates the instruction in order to figure out what the microarchitecture is being asked to do.
- Evaluate Address – This phase computes the address of the memory location that is needed to process the instruction.
- Fetch Operands – This phase obtains the source operands needed to process the instruction.
- Execute – This is when the actual execution of the instruction happens.
- Store result – The result is written to its destination.