**Main Card Algorithm:**

To understand how to formulate an algorithm that would be provide an accurate result for what is being asked. We had to look at the example provided: https://www.mathsisfun.com/games/mind-reader.html

We can see a pattern in the example that helped us developed techniques to complete this project. For example, we can see a from the first card that a revealing pattern of numbers is displayed showing that the numbers change by a power of 2n, meanwhile collecting data with the flip of each new card. The pattern leads to a binary representation of the player’s number. Each question that is a yes adds 1 bit to the result, and a 0 if to the result if the answer is no. Finally, after the last card the number that the user has in mind should display. Hence, why it is called the Mind Reader Game.

In card 1:

The first row begins in the pattern:

1st row:

1 3 5 7 9 11 13 15

0001 0011 0101 0111 1001 1011 1101 1111

2nd row:

17 19 21 23 25 27 29 31

10001 10011 10101 10111 11001 11101 11111

In the first card we can see that the first line starts adds one 20 and on the next card starts from 21 and the next card from 22. So, each new card starts from a higher power of 2 than the previous card. So the first card would start from 20 and all subsequent cards raise in a power until we get to the last card that starts from 25. Also it would add 1 to each bit while adding 2n to every bit that is divisible by 2n. So, we are adding 1 to the current number to get the next number. If the index of the next number is a multiple of its 2^k, then also add 1+2^k.

K is also the card number, as well as the included bit in each number. As the user progresses through each card the user will select yes or no for if their card is showing on the current card. If the answer is yes, then 2^k will be added to the total with k being the card number. If the answer is no, then nothing is added. For example, if the user is on the 2nd card and answers yes that the number is present then 1+22 is added to the total and the program would move on to card 3. If the answer is no, then ‘0’ would be added to the total and the program would move on to the next card. After the 6th card is completed, the sum total of 2k’s is displayed which would also be the number that the user has in mind. Hence, the input is a binary representation of the number that user and the output is the number the user had in mind.

|  |
| --- |
| **Character representation**    Our bitmap characters are each represented with a 32 bit MIPS word. Each bit is used to decide whether or not a color should be printed in a given memory location. |
| **Character display**    Each time a character is to be printed, the program gets the first bit by using AND with the value 1. If the bit is 1, a given color is written to the next pixel. Otherwise the pixel is incremented. For every fourth pixel, add the width of the board – 3 to the write address to realign the print location. |
| **Character spacing and positions**    The position of a character to print can be given using an X and Y index, similar to using a 2D array. These indices are then passed into a function, which returns the address of the top left corner of the corresponding location in the bitmap. A small x-offset is then added for formatting.  **Memory location = (bitmap board first address) + (4 \* X) + (bitmap width \* Y) + (x offset)** |
| **Clearing the bitmap board**  For each word starting at the initial board address and ending at the last board address, set each pixel color. The color is passed in as a parameter by the user. |
| **Printing the cards**  For each number to be displayed in the card, calculate the number using a given power of 2. Then recursively print each digit. The digits need to be printed in the reverse order they are calculated since higher place value digits are calculated after those of lower place value. Each numerical digit is used to calculate the index of its matching bitmap representation. The representation is loaded, written to the bitmap (see character display above), then a space is printed afterward based on the number of digits printed. This method also accounts for digit count, and therefore printed numbers do not incorrectly wrap when going to a new line. |
| **Printing strings to the bitmap**  For each character (byte) in the string, get the character’s corresponding bitmap representation word. This is done by subtracting ‘a’ or ‘A’ from the character, and checking whether or not either difference corresponds to a valid index in the bitmap character array. If the character does not match (is not a letter), the function then checks whether the character is equal to certain punctuation marks. A match results in the correct bitmap representation being loaded. Otherwise the INVALID\_CHAR placeholder character is loaded. The character is then written to the bitmap, and the index is incremented. |
| **Music implementation**  MIDI in MIPS is used to implement music. MIDI output produces sounds using sound cards through Syscall system services 31 and 33. Only 31 used in this program. Music scales are represented in this way: syscall 31 service parameter 61 represents Do, 63 represents Re, 65 represents Mi, 66 represents Fa and so on. If there is a void note in music, put 0. Music segments are stored in “music.asm” file by array SoundTrack1 and SoundTrack2. Music speed is controlled by Syscall 30, the time delay. The parameter for Syscall 30 is an integer indicating number of milliseconds between two notes. Music volume parameter is specified in syscall 31, from 0 – 127. This program adopts volume 100. In sound tracks, 0 represents the end of the music while 128 represents void note in music. A variable used to detect in advance if there is a void note and one more time delay needs to be added. Loops are introduced to check each note in sound track, do a time delay, and determine if it is a void note or the end of the music. Music in this file is implemented in this way. |
| **Calculating the final answer**  On program init final num value is initialized to 0. The program will ask the user whether or not their number appears in the printed card using the promptbool subroutine. The K bit used to generate the printed card is then added to the final num value based on whether the user answered yes or no.  K bit defined as the exponent in 2^k used to generate the card  **Final num value = Final num value + (user answer bool ^ K)** |
| **Prompting for user input (promptbool subroutine)**  A string is passed into the subroutine. The string is used as the prompt. The function prints the string, then waits for user character input. Once the user inputs a character, the function checks whether it equals ‘y’ or ‘n’ if it equals neither, the prompt is printed again. Otherwise a boolean 1 is returned for ‘y’ or 0 is returned for ‘n’ |