Assembly Project: Report 2

Wangzheng Jiang 1008109574; Yahya Elgabra 1008553030

December 5, 2022

1 Constants and Variables in Memory

Note: All data stored below are of data type "word".

1. Address of the display: `ADDR_DSPL = 0x10008000`
2. Address of keyboard input: `ADDR_KBRD = 0xffff0000`

3. Color of the side walls (and the top bar): `COLOR_WALLS = 0x00888888`
4. Thickness (in units) of the top bar: `TOP_BAR_THICKNESS = 8`
5. Thickness (in units) of the side walls: `SIDE_WALL_THICKNESS = 2`
6. Gap (in units) between the top bar and the first row of bricks: `TOP_GAP_THICKNESS = 8`

7. Thickness (in units) of a row of bricks: `BRICK_ROW_THICKNESS = 2`
8. The number of rows of bricks: `BRICK_ROW_AMOUNT = 7`
9. An array containing the colors of each row of bricks:
```
BRICK_COLORS = [0x008062e0, 0x007173c6, 0x006283ac, 0x00539492,
                0x0043a577, 0x0034b55d, 0x0025c643]
```

10. The y-coordinate of the paddle (constant): `PADDLE_Y = 61`
11. The x-coordinates of the paddle, these are variable, one stores the x-coordinate of the leftmost pixel of the paddle, one stores the rightmost. This enables us to adjust the length of the paddle, and makes it easier for us to update its position:
`PADDLE_X_LEFT = 26` ; `PADDLE_X_RIGHT = 36`

12. The y-coordinate of the paddle (constant): `PADDLE_2_Y = 58`
13. The x-coordinates of the second paddle, these are variable, one stores the x-coordinate of the leftmost pixel of the paddle, one stores the rightmost. This enables us to adjust the length of the paddle, and makes it easier for us to update its position:
`PADDLE_2_X_LEFT = 26` ; `PADDLE_2_X_RIGHT = 36`

14. The position of the ball (this is a variable): `BALL_X = 31` ; `BALL_Y = 56`

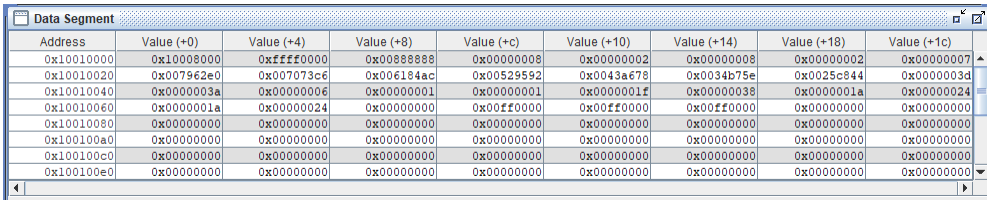
15. The movement vectors for the ball (each cycle the balls position are calculated as such: `BALL_X = BALL_X + VEC_X` ; `BALL_Y = BALL_Y + VEC_Y`). For ease of calculation, `VEC_X` and `VEC_Y` can only be either 1 or -1:
`VEC_X = 1` ; `VEC_Y = 1`

16. The width of a single brick (constant): `BRICK_WIDTH = 6`

17. The players' score, each time a brick is hit the score increments by 1 (variable): `SCORE = 0`

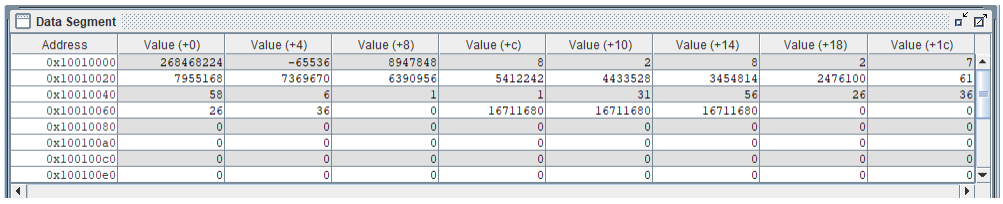
18. The players' lives, each time the players lose, one of the hearts turn black:
`LIVES = [0x00FF0000, 0x00FF0000, 0x00FF0000]`

Data in Memory:



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x10008000	0xffff0000	0x00888888	0x00000008	0x00000002	0x00000008	0x00000002	0x00000007
0x10010020	0x007962e0	0x007073c6	0x006184ac	0x00529592	0x0043a678	0x0034b75e	0x0025c844	0x0000003d
0x10010040	0x0000003a	0x00000006	0x00000001	0x00000001	0x0000001f	0x00000038	0x0000001a	0x00000024
0x10010060	0x0000001a	0x00000024	0x00000000	0x00ff0000	0x00ff0000	0x00ff0000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

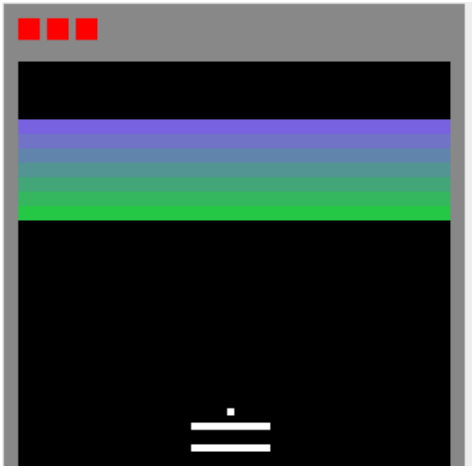
Data in Hexadecimal



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	268468224	-65536	8947848	8	2	8	2	7
0x10010020	7955168	7369670	6390956	5412242	4433528	3454814	2476100	61
0x10010040	58	6	1	1	31	56	26	36
0x10010060	26	36	0	16711680	16711680	16711680	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

Data in Decimal

2 Scene



3 Collision Algorithm

In each game cycle, we check the 4 pixels directly adjacent to the pixel where the ball is at: `(BALL_X + 1, BALL_Y)`, `(BALL_X - 1, BALL_Y)`, `(BALL_X, BALL_Y + 1)`, `(BALL_X, BALL_Y - 1)`. If any of the 4 pixels are not the default black background (0x00000000), then we have "touched" something.

And since we are only moving diagonally, the process is as follows:

If the ball is touching something on it's left or right, i.e. `(BALL_X + 1, BALL_Y)` or `(BALL_X - 1, BALL_Y)`, then we invert the sign of `VEC_X`. (`VEC_X = - VEC_X`) So that when we update the position of the ball using `BALL_X += VEC_X`. It will go the opposite direction than where it was going.

Likewise, if the ball is touching something on it's top or bottom, i.e. `(BALL_X, BALL_Y + 1)` or `(BALL_X, BALL_Y - 1)`, then we invert the sign of `VEC_Y`. (`VEC_Y = - VEC_Y`) So that when we update the position of the ball using `BALL_Y += VEC_Y`. It will go the opposite direction than where it was going.

These 2 cases can have either one applied, or both applied to the directional vectors of the ball.

4 How to Play

Buttons to play:

Player 1: 'a' for moving your pad to the left, 'd' for moving your pad to the right

Player 2: ',' for moving your pad to the left, '.' for moving your pad to the right

'p' to pause, 'p' to unpause and launch

'r' to reset

'q' to quit

Hit bricks with the ball to destroy them, each row of bricks requires 1 extra hit to be destroyed,press 'p' to launch the ball, you can move your paddles and the ball before launching! Make sure not to let the ball reach the bottom of the screen or else you will lose a life! You win once you destroy all the bricks.

5 Specifications

This version of the project uses eMARS 4.7 as it makes the game run smoother and makes things easier to set up