Assembly Project: Report 1

Wangzheng Jiang 1008109574; Yahya Elgabra 1008553030

November 29, 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 left-most 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 position of the ball (this is a variable): `BALL_X = 31` ; `BALL_Y = 58`
13. 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`

Data in Memory:

Data Segment

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	0x008062e0	0x007173c6	0x006283ac	0x00539492	0x0043a577	0x0034b55d	0x0025c643	0x0000003d
0x10010040	0x0000001a	0x00000024	0x0000001f	0x0000003a	0x00000001	0x00000001	0x00000006	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	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
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

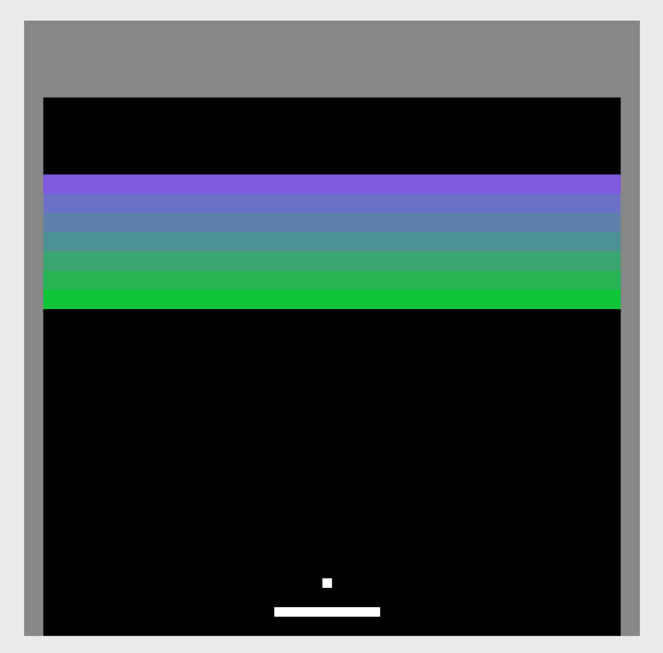
Data in Hexadecimal

Data Segment

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	8413920	7435206	6456236	5477522	4433271	3454301	2475587	61
0x10010040	26	36	31	58	1	1	6	0
0x10010060	0	0	0	0	0	0	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
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0
0x10010140	0	0	0	0	0	0	0	0
0x10010160	0	0	0	0	0	0	0	0
0x10010180	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.