1 Introduction

In this program, we decide to implement the Flappy game with LC-3 assembly language. In the original game, a bird is flying from left to right, but in this game, it may fly from top to bottom. In the game, the bird is represented by 3 continuous letters(for example *aaa*). Without control, it will fall to left, but the user can make it fly to right by 1-9 blocks (chars) by clicking corresponding numbers. By the way, the bird will change its appearance after the user clicked a-z.

In this program, falling to gound (the leftmost side) won't end the game. And flying too high (right) is not allowed,we just put the bird on the rightmost side if it fly too high.

Here is an example:

```
.....aaa............ User Input(you don't need to print)
....aaa.............
...aaa..............
............aaa..... <-9
...........aaa......
.................aaa <-8
................aaa.
...............aaa..
..............aaa...
.............aaa....
............aaa.....
...........aaa......
..........aaa.......
.........aaa........
........ddd......... <-d
.......ddd..........
......ddd...........
.....ddd............
....ddd.............
...ddd..............
..aaa............... <-a
.aaa................
aaa.................
aaa.................
ooo................. <-o
ooo.................
....ooo............. <-4
...ooo..............
```

2 Algorithm Specification

The program can be roughly divided into two part: the interrupt function and the main function

In order to motivate the interrupt, we should initialize the KBSR[14] when initializing the program. So we rewrite the instructions from x0200. In the new instructions, we add a instruction to load x4000 to KBSR, so that the KBSR[14] will be 1. In this way, when we press the keyboard, the KBSR will create a INT signal to interrupt the program and the PC will change to the content of x0180. We also load the address of interrupt function to x0180, which is x2000. When we press the keyboard, the KBSR[15] will be 1, INT signal gengerated, then the PC will come to x2000, where the interrupt function locates.

To handle the problem that the player inputs multiple operation one times. We use a data structure to store the operations inputted. A queue is a good choice. When a character is inputted, we just store it into the queue. When the main function ready to output, we just take out the data in queue and analyse them.

After complex introduction, next is the detail of interrupt function and the main function.

In interrupt function, we initially check the KBSR[15]. when it's 1, we take the KBDR into register and then enqueue it into queue.

```
while KBSR[15] != 1
    loop
R0 ← KBDR
enqueue(R0)

```

In main function, we firstly use a loop to cause the delay. After the loop, the program dequeues the data in the queue with checking whether they are characters or digit. If a data is characters, the name of bird will be changed. If the data is a digit otherwise, we change the ascii code into digit and add it with the height. Finally, before print out, we should check whether the height is overflow or underflow and do some adjustments to it.

```
count ← x6000
R1 ← height
while count > 0
    count--
 c ← dequeue
 while c != -1
    if c is character
        name ← c
    if c is digit
        R1 ←  R1 + c - '0'
 if R1 == height
    R1 ← R1 - 1
 print(name, height)

```

About the function to output according to name an height, it firstly print ‘ · ’ as much as the height. Then, the program output three characters of the name. Finally, we calculate $17 - height$ and output ‘ · ’ as much as it.

```
for i = 1 to height
    R0 ← '·'
    trap x21
R0 ← name
trap x21
trap x21
trap x21
height ← 17 - height
for i = 1 to height
    R0 ← '·'
    trap x21

```

Next is the implement of the data structure, which is a queue supporting the operation of dequeue and enqueue. It's much like the data structure in Lab3, so we don't explain much about it.

```
enqueue: R0 is the element , front_ptr is the point to the front of queue
*(--front_ptr) ← R0

```

```
        dequeue: front_ptr is the point to the front of queue,
            back_ptr is the point to the back of queue
        if front_ptr == back_ptr + 1
            R0 ← -1
        else
            R0 ← *(back_ptr--)

```

3 Q and A

- Q: what you put in each part of your program ?

  A: The code of the program is mainly constructed by three parts. First one is from x0200, where we should do some initialization. We initialize the interrupt function address of x0180, and set the KBSR[14] to be 1. Second one is from x2000, where the keyboard interrupt function locates. Here, we read the character in KBDR and store it into queue. Third one is from x3000, which is the main function. We form a loop and the dequeue the item in queue until the queue becomes empty. After getting an item, we check whether it is character or digit, then do operations to name or height. Finally we output a line according to name and height.

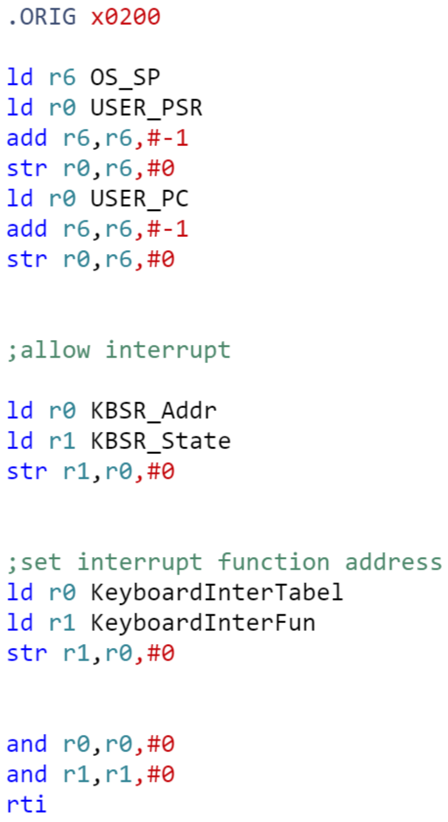
4 essential parts of code

Fig 1 is the implement of code from x0200

Fig 2 is the implement of code from x2000

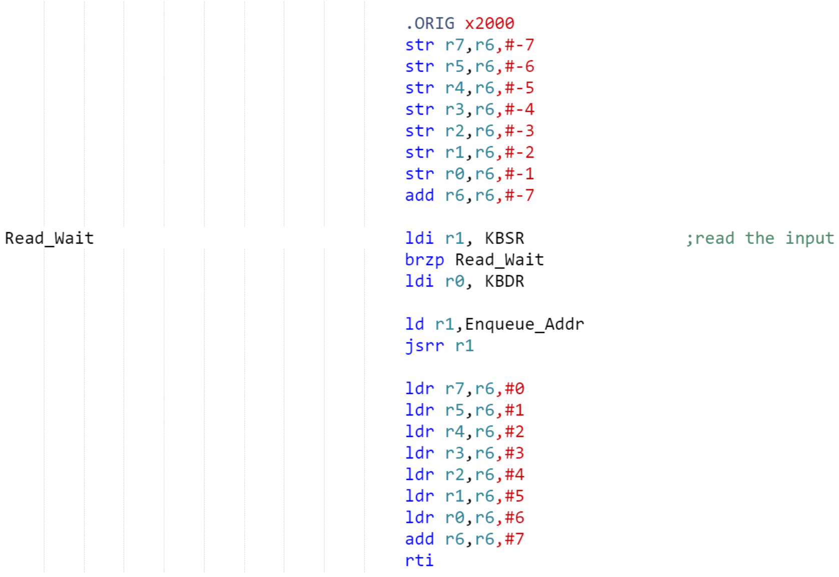
Fig 2 is the main function from x3000

Fig 4 is the implement of getdata function



```
.ORIG x0200

ld r6 OS_SP
ld r0 USER_PSR
add r6,r6,#-1
str r0,r6,#0
ld r0 USER_PC
add r6,r6,#-1
str r0,r6,#0


;allow interrupt

ld r0 KBSR_Addr
ld r1 KBSR_State
str r1,r0,#0


;set interrupt function address
ld r0 KeyboardInterTabel
ld r1 KeyboardInterFun
str r1,r0,#0


and r0,r0,#0
and r1,r1,#0
rti
```

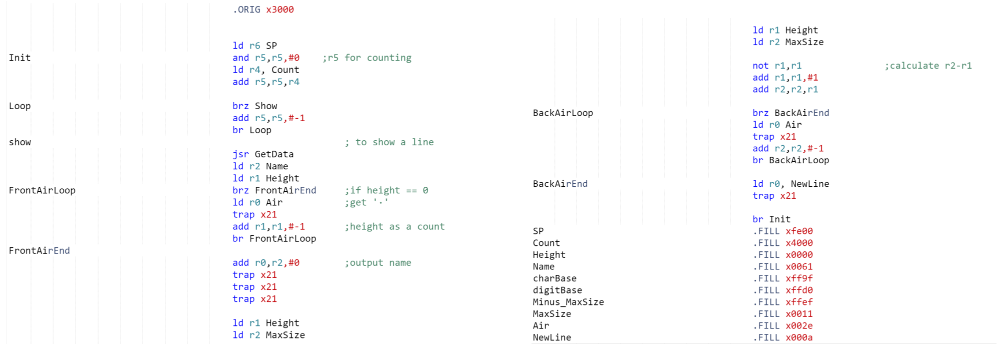
Figure 1: Fig 1



```
                .ORIG x2000
                str r7,r6,#-7
                str r5,r6,#-6
                str r4,r6,#-5
                str r3,r6,#-4
                str r2,r6,#-3
                str r1,r6,#-2
                str r0,r6,#-1
                add r6,r6,#-7

Read_Wait       ldi r1, KBSR              ;read the input
                brzp Read_Wait
                ldi r0, KBDR

                ld r1,Enqueue_Addr
                jsrr r1

                ldr r7,r6,#0
                ldr r5,r6,#1
                ldr r4,r6,#2
                ldr r3,r6,#3
                ldr r2,r6,#4
                ldr r1,r6,#5
                ldr r0,r6,#6
                add r6,r6,#7
                rti
```

Figure 2: Fig 2

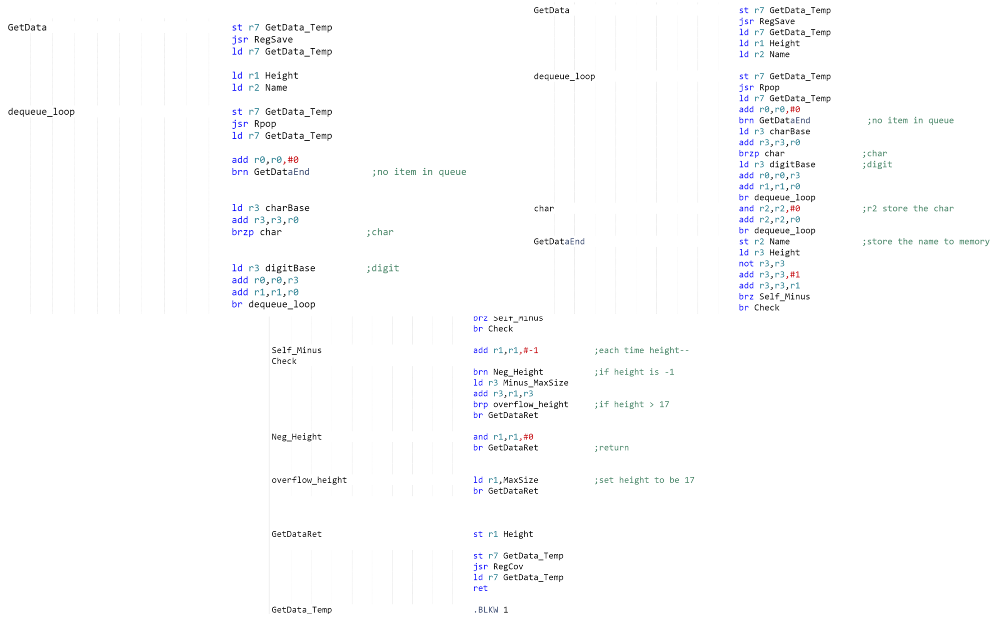
Figure 3: Fig 3



```
                                        .ORIG x3000

                                        ld r6 SP
Init                                    and r5,r5,#0    ;r5 for counting
                                        ld r4, Count
                                        add r5,r5,r4

Loop                                    brz Show
                                        add r5,r5,#-1
                                        br Loop
show                                                        ; to show a line
                                        jsr GetData
                                        ld r2 Name
                                        ld r1 Height
FrontAirLoop                            brz FrontAirEnd     ;if height == 0
                                        ld r0 Air           ;get '.'
                                        trap x21
                                        add r1,r1,#-1       ;height as a count
                                        br FrontAirLoop
FrontAirEnd
                                        add r0,r2,#0        ;output name
                                        trap x21
                                        trap x21
                                        trap x21

                                        ld r1 Height
                                        ld r2 MaxSize
```

```
                                        ld r1 Height
                                        ld r2 MaxSize

                                        not r1,r1               ;calculate r2-r1
                                        add r1,r1,#1
                                        add r2,r2,r1

BackAirLoop                             brz BackAirEnd
                                        ld r0 Air
                                        trap x21
                                        add r2,r2,#-1
                                        br BackAirLoop

BackAirEnd                              ld r0, NewLine
                                        trap x21

                                        br Init
SP                                      .FILL xfe00
Count                                   .FILL x4000
Height                                  .FILL x0000
Name                                    .FILL x0061
charBase                                .FILL xff9f
digitBase                               .FILL xffd0
Minus_MaxSize                           .FILL xffef
MaxSize                                 .FILL x0011
Air                                     .FILL x002e
NewLine                                 .FILL x000a
```

Figure 4: Fig 4



```
GetData                                 st r7 GetData_Temp
                                        jsr RegSave
                                        ld r7 GetData_Temp

                                        ld r1 Height
                                        ld r2 Name

dequeue_loop                            st r7 GetData_Temp
                                        jsr Rpop
                                        ld r7 GetData_Temp

                                        add r0,r0,#0
                                        brn GetDataEnd          ;no item in queue


                                        ld r3 charBase
                                        add r3,r3,r0
                                        brzp char               ;char


                                        ld r3 digitBase         ;digit
                                        add r0,r0,r3
                                        add r1,r1,r0
                                        br dequeue_loop
```

```
GetData                                 st r7 GetData_Temp
                                        jsr RegSave
                                        ld r7 GetData_Temp
                                        ld r1 Height
                                        ld r2 Name

dequeue_loop                            st r7 GetData_Temp
                                        jsr Rpop
                                        ld r7 GetData_Temp
                                        add r0,r0,#0
                                        brn GetDataEnd           ;no item in queue
                                        ld r3 charBase
                                        add r3,r3,r0
                                        brzp char               ;char
                                        ld r3 digitBase         ;digit
                                        add r0,r0,r3
                                        add r1,r1,r0
                                        br dequeue_loop
char                                    and r2,r2,#0            ;r2 store the char
                                        add r2,r2,r0
                                        br dequeue_loop
GetDataEnd                              st r2 Name              ;store the name to memory
                                        ld r3 Height
                                        not r3,r3
                                        add r3,r3,#1
                                        add r3,r3,r1
                                        brz Self_Minus
                                        br Check
```

```
                                        brz Self_Minus
                                        br Check

Self_Minus                              add r1,r1,#-1           ;each time height--
Check
                                        brn Neg_Height          ;if height is -1
                                        ld r3 Minus_MaxSize
                                        add r3,r1,r3
                                        brp overflow_height     ;if height > 17
                                        br GetDataRet

Neg_Height                              and r1,r1,#0
                                        br GetDataRet           ;return


overflow_height                         ld r1,MaxSize           ;set height to be 17
                                        br GetDataRet



GetDataRet                              st r1 Height

                                        st r7 GetData_Temp
                                        jsr RegCov
                                        ld r7 GetData_Temp
                                        ret

GetData_Temp                            .BLKW 1
```