Lab4 Report

作者: 罗文杰
专业: 计算机科学与技术
学号: 3210102456

1 Introduction

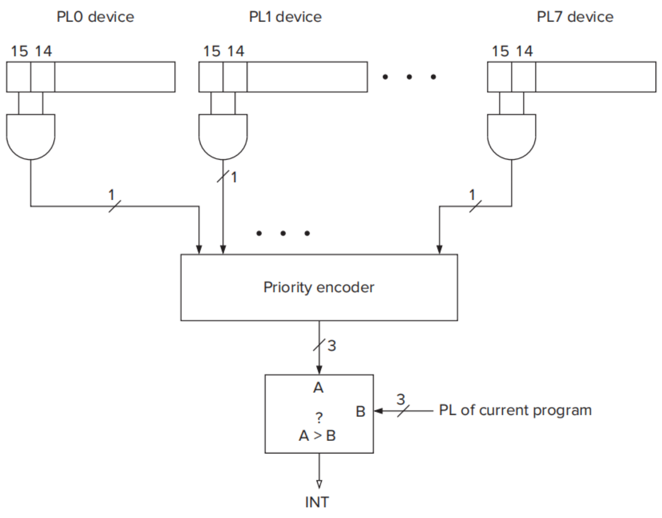
One day, you are playing Flappy and decide to implement it with LC-3 assembly language. In the original game, a bird is flying from left to right, but you may fly from top to bottom. In the game, the bird is represented by 3 continuous letters(for example aaa). Without control, it will fall to left, but the user can make it fly to right by 1-9 blocks (chars) by clicking corresponding numbers.

2 Interrupt

2.1 Causing the Interrupt to Occur

There are two ways to transfer of information between the computer and the I/O. They differ in terms of who leads the interaction, the processor or typist. If processor actively test whether the typist types something in a routine, this type of interaction is called polling. If processor do its own thing until being interrupt by an announcenment from the keyboard, this is called interrupt-driven.

To make interrupt happen, we need a signal called INT. In an instruction cycle, INT play a role between the last state of one instruction cycle and the FETCH of the next instruction. If INT is asserted, the control unit will do two things before back to FETCH phase: (1) save the state of the interrupted program, and (2) load the state of the interrupt service routine. How the INT signal is generated is as follows.



generation of the INT signal

The PL devices shows the status register of the I/O device. KBSR has two bits to decide the generation of interrupt request signal, the IE (interrupt enable) bit and the ready bit. When the two bits are both set, then the priority encode will test whether the PL of interrupt request is higher than the PL of the currently executing program, if so, the INT signal is asserted.

In LC-3 the only one I/O device that can interrupt the processor is keyboard. It interrupts at priority level PL4, so when a program is running at priority level less than 4, the IE bit and ready bit of the KBSR is 1, then the currently executing program is interrupted at the end of the current instruction cycle.

2.2 Handling the Interrupt request

The interrupt service routine is initiated as follows:

1. The processor sets the privilege mode to supervisor mode and the priority level to PL4.
2. If the interrupted process is in User mode, R6 is saved in Saved_USP and R6 is loaded with the SSP from the Saved_SSP.
3. The PSR and PC of the interrupted process are pushed onto the supervisor stack.
4. The PC is loaded with the contents of memory location x0180, the corresponding 16-bit address in the vector table.

Since the PC contains the starting address of the interrupt service routine, the service routine will execute.

The last instruction in every Interrupt service routine is RTI, return from interrupt. The RTI instruction will restore the PC and PSR from the supervisor stack, and if the restored PSR[15]=1, RTI will save R6 into Saved_SSP and load the R6 with Saved_USP.

3 Boot

Based on the above analysis, we find that we need to do a few things if we want the interrupt to be generated: (1) write an interrupt function and put its start address in x180 and (2) rewrite the IE value of KBSR. This all requires proces have Supervisor mode. But our user function is in x3000 and only has User mode, who cann't make it. To cover the problem, we need write system booting code.

This codes are as follows.

```
.ORIG x200
LD  R6, Stack
LD  R0, USR_PTR
ADD R6, R6, #-1
STR R0, R6, #0
LD  R0, USR
ADD R6, R6, #-1
STR R0, R6, #0
LD  R1, INKB    ; x0180
LD  R2, ADDER   ; x2000
STR R2, R1, #0
LD  R1, MASK
STI R1, KBSR
RTI
```

```
Stack   .FILL   x3000
INKB    .FILL   x0180   ; INTV
ADDER   .FILL   x2000
MASK    .FILL   x4000
KBSR    .FILL   xFE00
USR     .FILL   x3000
USR_PTR .FILL   x8002
.END
```

It does three things: (1) put the start address of the interrupt function we wrote in INTV, (2) set the IE value of KBSR and (3) back to user functions.

When the RTI executes, the processor will restore PC and PSR with the contents of supervisor stack (starting from x3000), where the USR address have been pushed, so the processor will execute the user program.

4 Algorithm Explanation

To create a Flappy, the first step is to make it fly.

The Flappy needs two empty memories, one to store its exterior and the other to record its height. In a loop, the program will first output the dots, and when enough dots have been output (up to Flappy's height), the program will place Flappy on the screen and then output the rest of the dots. When a loop ends, the program will make Flappy's height minus 1 and delay for another period of time.

The loop code are as follows.

```
; CHAR is the address storing the exterior of Flappy
; HIGHT is the address storing the hight of Flappy
; R2 is hight of Flappy, R3 is the whole hight of sky
PUT     LD  R3, SKY
        LDI R2, HIGHT
LOOP    ADD R3, R3, #-1
        BRz OVER2
        ADD R2, R2, #-1
        BRz BIRD        ; Flappy is here
        LD  R0, DOTS    ; put the dots
        OUT
        BRnzp   LOOP
BIRD    LDI R0, CHAR
        OUT
        OUT
        OUT             ; put the Flappy
        ADD R3, R3, #-2
        BRnzp   LOOP
```

```
OVER2   LD R0, LINE
        OUT
        LDI R2, HIGHT
        ADD R2, R2, #-1 ; Flappy fly down
        BRnz DOWN
        STI R2, HIGHT
DOWN    JSR DELAY
        BRnzp   PUT
```

When a key is pressed on the keyboard, the processor enters the interrupt function. In the interrupt function, the processor reads the current input and rewrites the value of the address of the stored FLappy property depending on whether it is a number or a letter.

When the processor returns to the user function, Flappy will have a new look or height as the memories storing the characteristic of Flappy are rewriten.

The rewrite codes are as follows.

```
; CHAR1 is the address storing the exterior of Flappy;
; HIGHT1 is the address storing the hight of Flappy
; R2 is hight of Flappy, R3 is the whole hight of sky
        LDI R1, KBDR    ; get the input
        LD  R2, APHU    ; determine if the input is a letter or a number
        ADD R3, R2, R1
        BRp AP
        LD  R0, NNUM
        ADD R1, R0, R1
        LDI R2, HIGHT1
        ADD R2, R1, R2
        LD  R0, MAX
        ADD R0, R2, R0  ; determine if Flappy fly to highest point
        BRp HIGH
        STI R2, HIGHT1
        BRnzp   OVER1
HIGH    LD  R0, MAX
        NOT R0, R0
        ADD R0, R0, #1
        STI R0, HIGHT1  ; Flappy fly to highest point
        BRnzp   OVER1
AP      STI R1, CHAR1
```