Interrupt

Polling VS Interrupt

The processor control the interaction, specifically by interrogating (usually, again and again) the ready bit until it (the processor) detects that the ready bit is set.

The processor do its own thing until being interrupted by an announcement from the keyboard.

Exception VS Interrupt

异常$Exception$来自于程序内部（包括硬件的损坏）

> 1. privilege mode exception (user program use RTI)
> 2. illegal opcode exception (opcode 1101)
> 3. access control violation exception (user mode access before x3000 or after xfdff)

中断$Interrupt$则来自程序外部，由更高$Priority$的程序打断了正在进行的进程。

- $x0100 - x017F$提供$exception$函数的开始位置
- $x0180 - x01FF$提供$interrupt\ service\ routines$函数的开始位置
- $x0180$**提供键盘中断函数的开始位置**

Two Parts to the Process

There are two parts to interrupt-driven I/O:

1. the mechanism that enables an I/O device to interrupt the processor
2. the mechanism that handles the interrupt request

Condition of Interrupt

Several things must be true for an I/O device to actually interrupt the program that is running:

1. The I/O device **must want** service.（$ReadyBit$，**第**15**位为**1）
2. The device **must have the right** to request the service.（**第14位为1**）
3. The device request **must be more urgent** than what the processor is currently doing.（$Higher\ Priority$）

INT Signal Generation

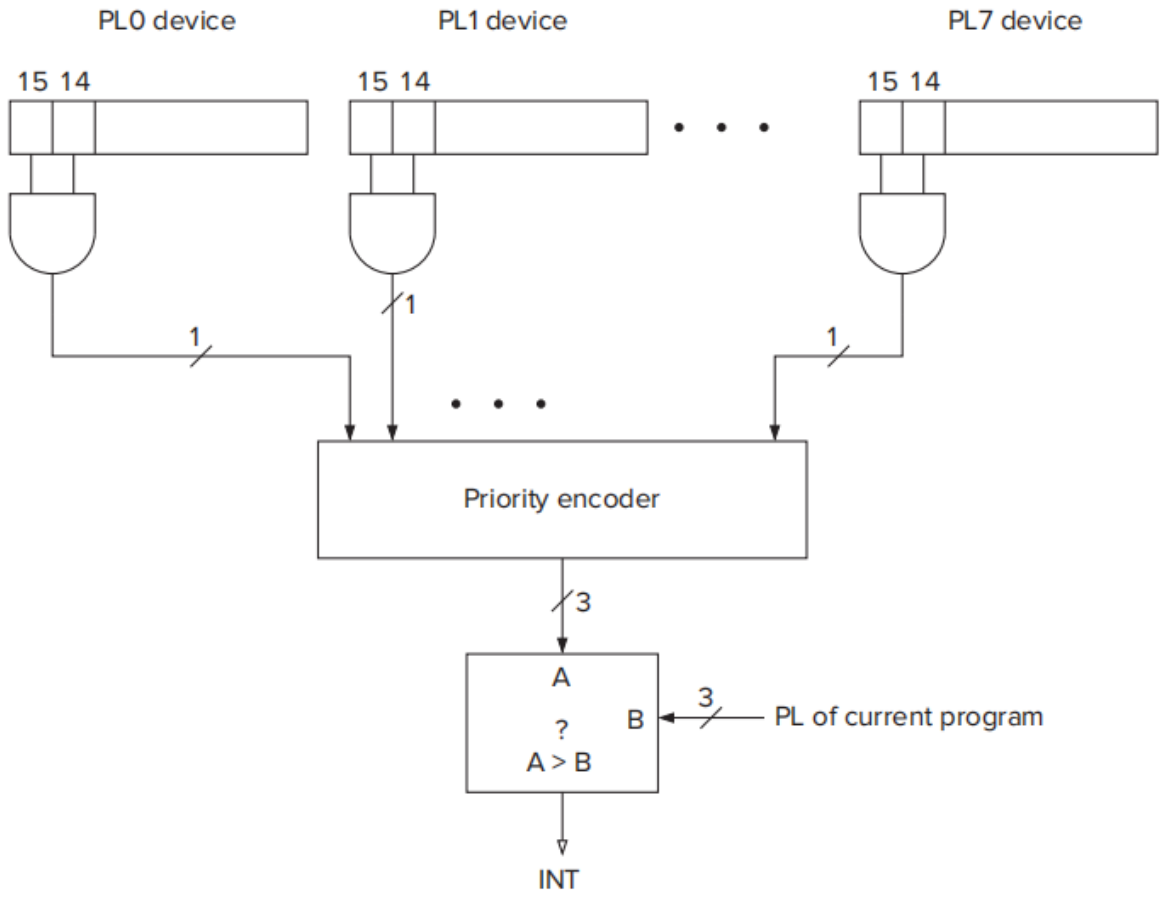

Figure 9.19 Generation of the INT signal.

State Machine

- 注意$PC-1$（因为中断是还没有执行这条指令的时候，在$Fetch$阶段就被打断了，$Exception$则是要么直接停机，要么重新执行一次失败的指令）
- 注意PSR和PC总是存在SSP下的

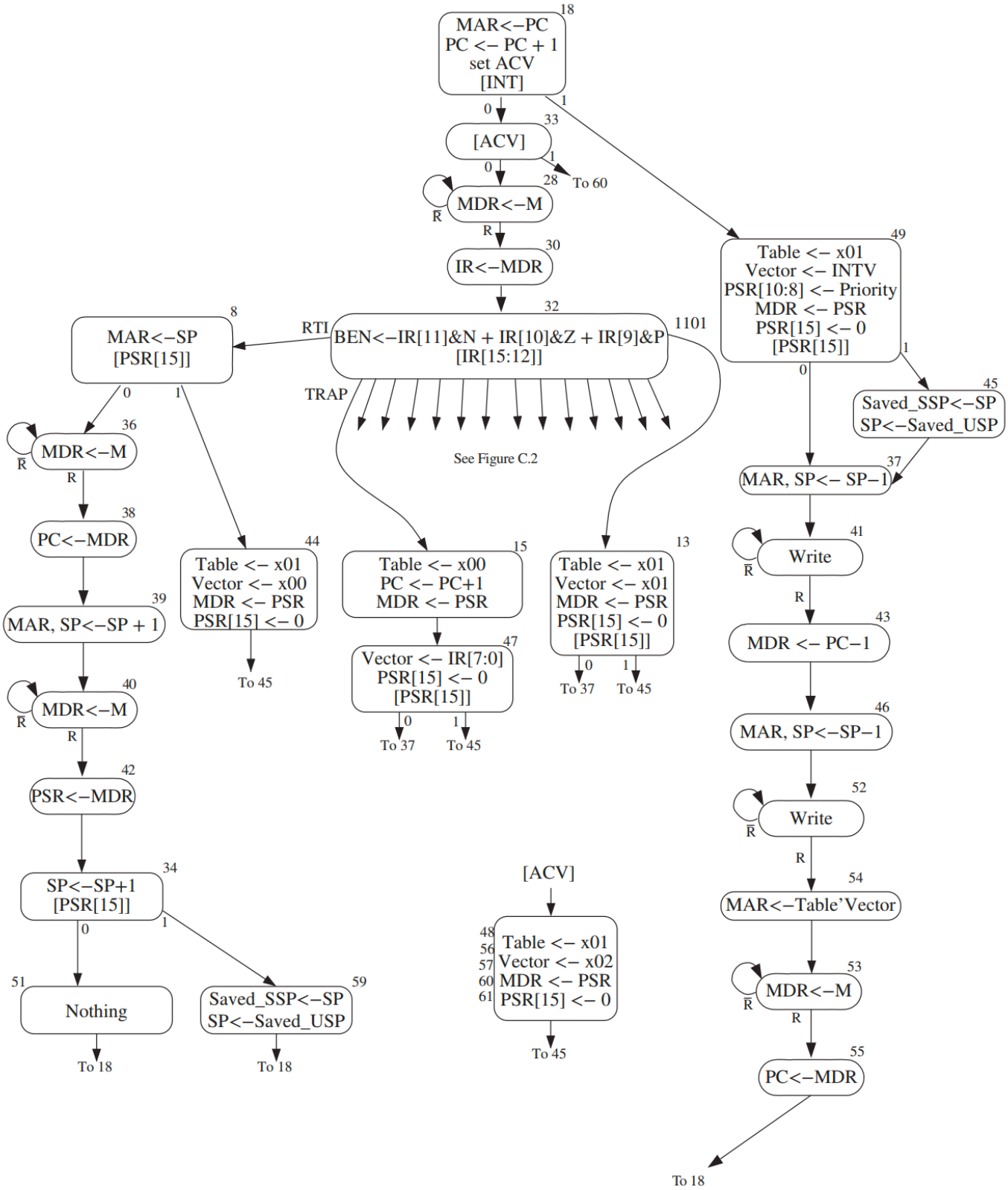

Figure C.7 LC-3 state machine showing interrupt control.

> state 45 is wrong
>
> Can you find all three kinds of exceptions of LC3 in the following graph ?
> When enter interrupt, is PSR pushed into system stack before or after PC ? (before)
> In state 49, why is Priority put into PSR[10:8] without checking whether larger than current? (INT信号产生的时候就检查了)
> In state 43, why should we push PC-1 into stack, but not PC ? (中断的指令还没做，所以返回地址要把

> PC++的效果抵消)

Tip

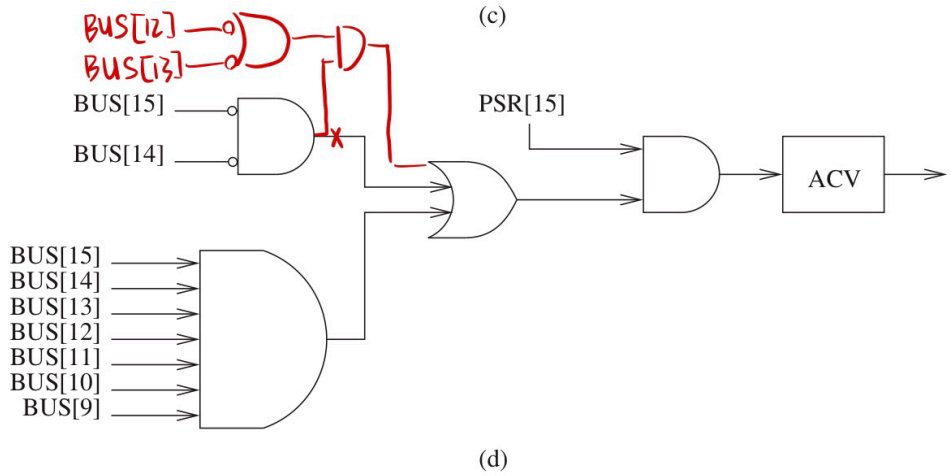

Figure C.6 Additional logic required to provide control signals.