Interrupt-Driven I/O

1) The enabling mechanism that allows the device to interrupt the processor (Ch. 8)
2) The process that manages the transfer of the I/O data (Ch. 10.2)

I/O Data Transfer

1) Initiate the interrupt
    a) Save the state of the interrupted program
    b) Load the state of the service routine
2) Service the interrupt
3) Return from the interrupt

Program State

1) **PSR** Processor Status Register
    a) **Pr** Privilege: **0** = **supervisor** mode, or **1** = **user** mode
    b) **PL** Priority Level: 0 .. 7
    c) Flags / Condition Codes **N**, **Z**, **P**

Pr					**PL_2**	**PL_1**	**PL_0**						**N**	**Z**	**P**

2) **PC** – Program Counter
3) General Purpose Registers – **R0**, **R1**, .. **R7**
4) Memory allocated to the program

I/O Data Transfer

4) Initiate the interrupt
    a) Save the state of the interrupted program
    //Registers: **Saved.SSP**, **Saved.USP**, Stack Pointer(**R6**)

```
if ( PSR[15] ) //user mode
{     Saved.USP ← R6
      R6 ← Saved.SSP
}
push( PC )
push( PSR )
```

    b) Load the state of the service routine
        Interrupt Vector Table x0100 – x01FF
        I/O Process supplies 8-bit INTV

```
MAR ← x01_ext16 (INTV)
MDR ← memory[MAR]
PC ← MDR
PSR ← x0 0000 ppp 00000 000
```

5) Service the interrupt
        Interrupt Handler Executes

6) Return from the interrupt
        RTI instruction

RTI Execute Phase

```
if  (! PSR[15] ) {
        /*  pop the (supervisor) stack into PSR  */
        pop(PSR)
        /*  pop the (supervisor) stack into PC  */
        pop(PC)
        /*  reset stack pointer if switching back to user mode*/
        if  ( PSR[15] ) {
                Saved.SSP ← R6
                R6 ← Saved.USP
        }
}
else
        Raise a privilege mode exception
```

Stack Operations

push(Register)

```
R6 ← R6 – 1
MAR ← R6
MDR ← Register
memory[MAR] ← MDR
```

pop(Register)

```
MAR ← R6
MDR ← memory[MAR]
Register ← MDR
R6 ← R6 + 1
```