Chapter 9 I/O

9.1 Privilege, Priority, and the Memory Address Space

9.1.1 Privilege

The right to do something.

In LC-3, there are 2 levels of privilege, the ***supervisor privilege*** and unprivileged mode.

9.1.2 Priority

The urgency of a program to execute.

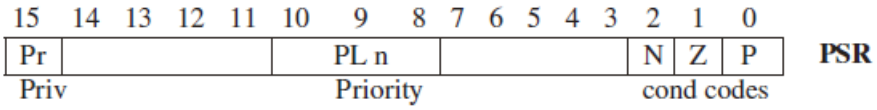
In LC-3, there are 8 levels of priority, from PL0 to PL7. (PL: Priority Level)

The higher the level is, the more urgent the program to execute is.

9.1.3 PSR

To specify or to store the state of a program, we just need two registers in LC-3: PC & PSR. PC states the next instruction that will be executed. PSR states the privilege, priority and condition code of the program.

> *Actually, we also need store the Register Files' values when to store the state of a program. But in LC-3, we make sure that any other program will do save-restore to not destroy the value in R0 - R7, so we do not need store Register Files.*

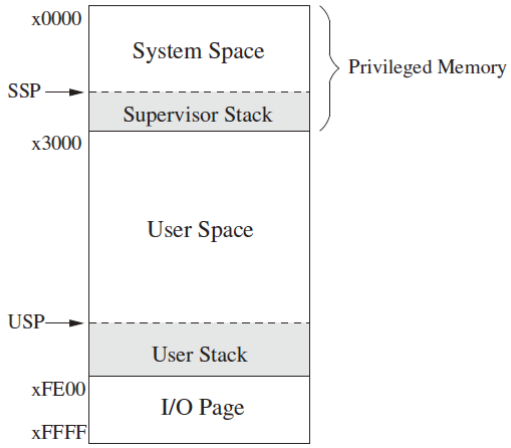


Pr: 1 for unprivileged and 0 for supervisor

PLn: from 000 to 111, indicating PL0 to PL7

NZP: condition code.

9.1.4 Organization of Memory



System Space: contain the various data structures and codes for operating system. **Need supervisor privilege to access.**

User Space: All user program and user data.

I/O Page: Identify registers that take part in I/O functions and some special registers associated with the processor like PSR & MCR.

NOTE: the location in I/O Page **DOES NOT EXIST** in memory physically. They are just mapped into these locations.

> ***SSP & USP:***

Since we just use $R6$ as SP, but there are two stacks actually: **System Stack** and **User Stack** , so we need two locations to store the value of top of the two stacks(the location that last time $R6$ points to recently), which are ***SAVED_SSP*** & ***SAVED_USP***.

Each time the privilege changes, we need change $R6$ from SSP to USP or from USP to SSP. Before we load $R6$ with the value in ***SAVED_SSP*** or ***SAVED_USP***, we need store the value of $R6$ into another one to refresh its value.

9.2 Input/Output

9.2.1 to 9.2.3 is actually 3 pairs of I/O implement strategies.

This part in the 3rd textbook is recommend reading.

9.2.1 Memory-Mapped I/O *VS* Special I/O Instructions

Memory-Mapped I/O:

- Map the registers in I/O device into memory location(I/O Page in LC-3)
- Just normal LD(LDI, LDR, LEA) and ST(STI, STR) instructions.

Special I/O Instructions:

- No mapped addresses
- Need special instructions for I/O operation.(Example: PDP-8)

We take Memory-Mapped I/O strategy in LC-3.

9.2.2 Asynchronous *VS* Synchronous

Asynchronous: Varying rate with the clock in CPU.

Synchronous: Fixed rate with the clock in CPU.

We pay more attention to Asynchronous.

9.2.3 Interrupt-Driven *VS* Polling

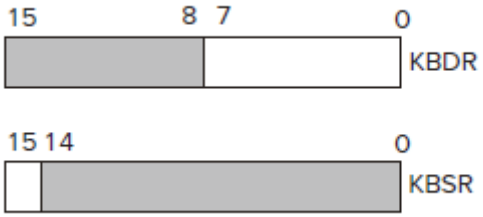
Key Point: who controls the interaction.

Interrupt-Driven: The one who raise an interaction (not only I/O device will cause a interaction) controls the interaction, not CPU. CPU do its own thing **until** the one who raise an interaction **announce** that there is an interaction to handle.

Polling: CPU controls the interaction. CPU **keeps asking** the device if there is an interaction to handle. (example: KBSR&KBDR, DSR&DDR)

We both use Interrupt-Driven & Polling in LC-3, but in different areas.

9.2.4 Input from Keyboard



KBSR: xFE00 KBDR: xFE02

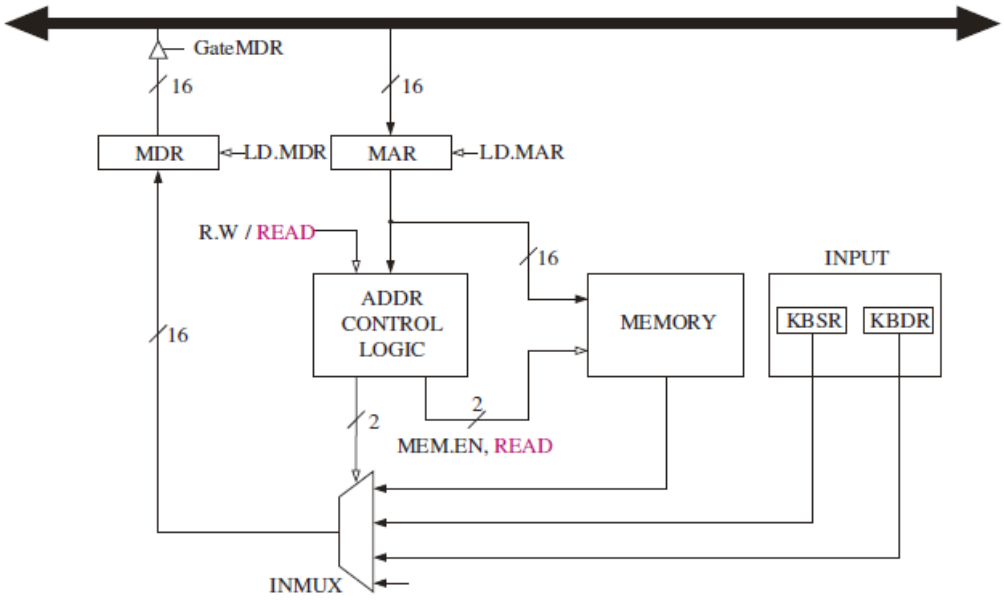
KBSR[15]: ready bit 1 for ready

KBDR[7:0]: ASCII code

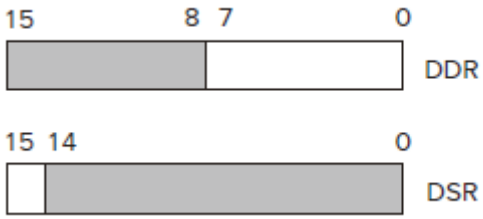
Polling Routine:

```
START   LDI     R1, A       ; Test for
        BRzp    START       ; character input
        LDI     R0, B
        BRnzp   NEXT_TASK   ; Go to the next task
A       .FILL   xFE00       ; Address of KBSR
B       .FILL   xFE02       ; Address of KBDR
```

Data Path for KBSR & KBDR:



9.2.4 Output to the Monitor



DSR: xFE04 DDR: xFE06

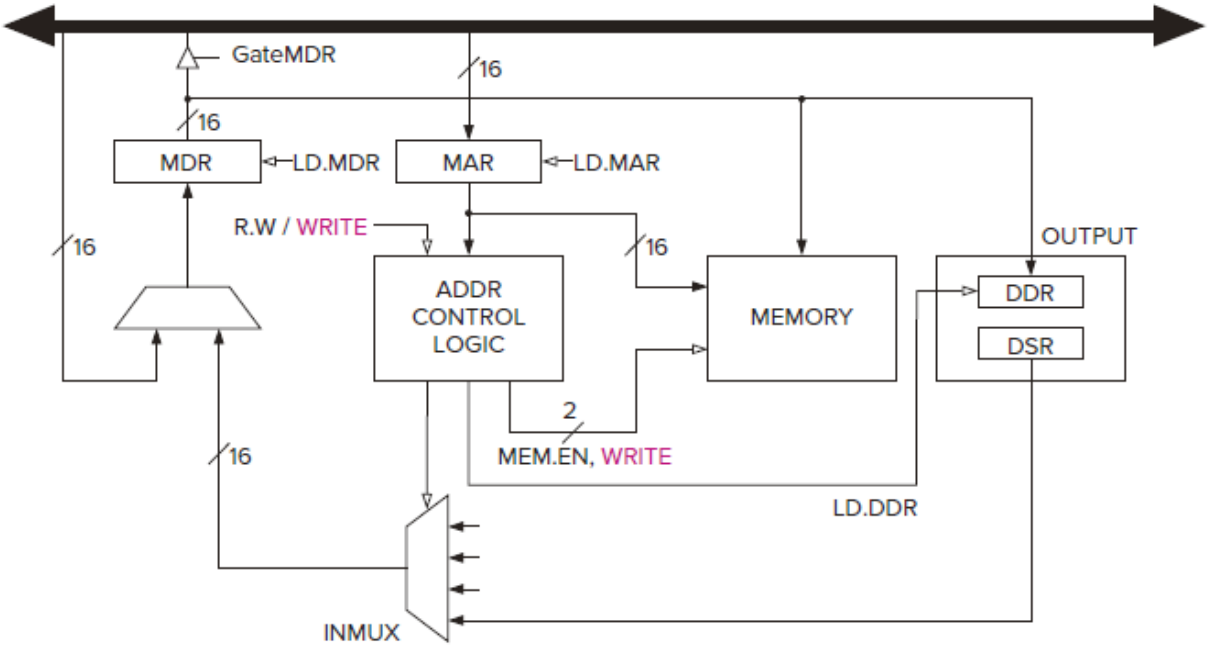
DSR[15]: ready bit 1 for ready

DDR[7:0]: ASCII code

Polling Routine:

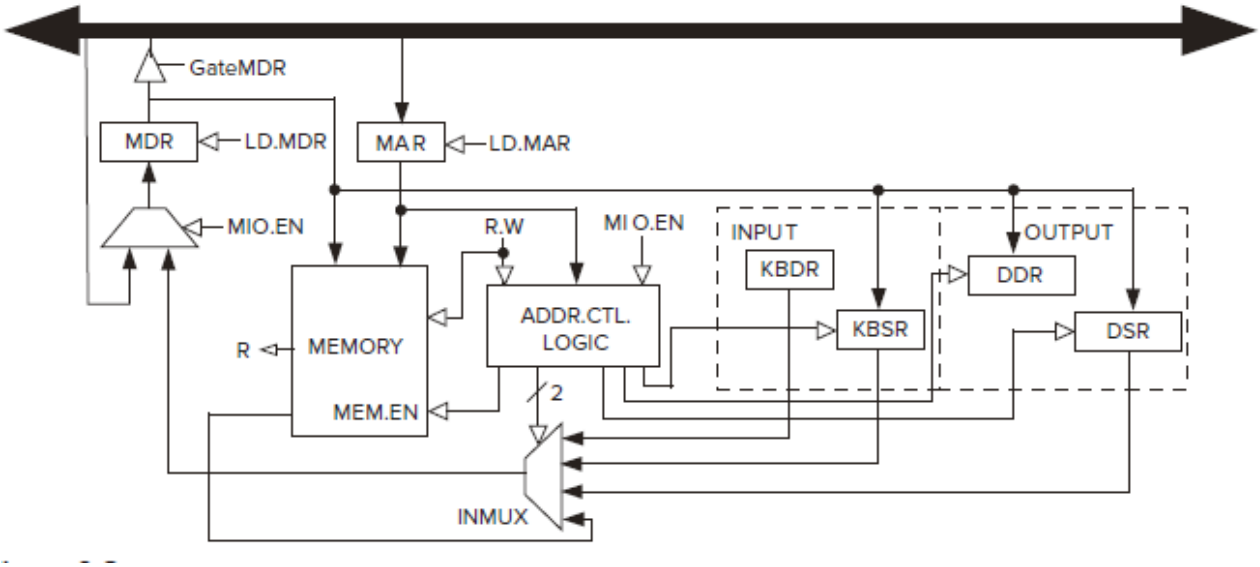
```
START   LDI     R1, A       ; Test for if
        BRzp    START       ; output register is ready
        STI     R0, B
        BRnzp   NEXT_TASK   ; Go to the next task
A       .FILL   xFE04       ; Address of DSR
B       .FILL   xFE06       ; Address of DDR
```

Data Path for DSR & DDR:



Full Data Path:

Chapter 9.2.5 is recommend reading.



9.4 Interrupts and Interrupt-Driven I/O

9.4.1 Interrupt-Driven I/O

1. Force the running program to stop
2. Have the processor execute a program that carries out the needs of the I/O device
3. Have the stopped program resume execution as if nothing had happened

```
.
  Program A is executing instruction n
  Program A is executing instruction n+1
  Program A is executing instruction n+2
1: Interrupt signal is detected
1: Program A is put into suspended animation
1: PC is loaded with the starting address of Program B
2: Program B starts satisying I/O device's needs
2: Program B continues satisfying I/O device's needs
2: Program B continues satisfying I/O device's needs
2: Program B finishes satisfying I/O device's needs
3: Program A is brought back to life

  Program A is executing instruction n+3
  Program A is executing instruction n+4
.
```

Interrupt-Driven I/O is much more efficient than polling.

9.4.2 Two Parts to the Process

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

9.4.3 PART I: Causing the Interrupt to Occur

3 things that must be true to interrupt the running program:

1. The I/O device **must want** service

   -- ***ready bit***

2. The device **must have the right** to request the service

   -- KBSR[14] is IE bit, that is, Interrupt Enable, specifying if it is allowed to cause interrupt.

   Only $ready\space bit\space \&\space IE==1$, can I/O device cause interrupt.

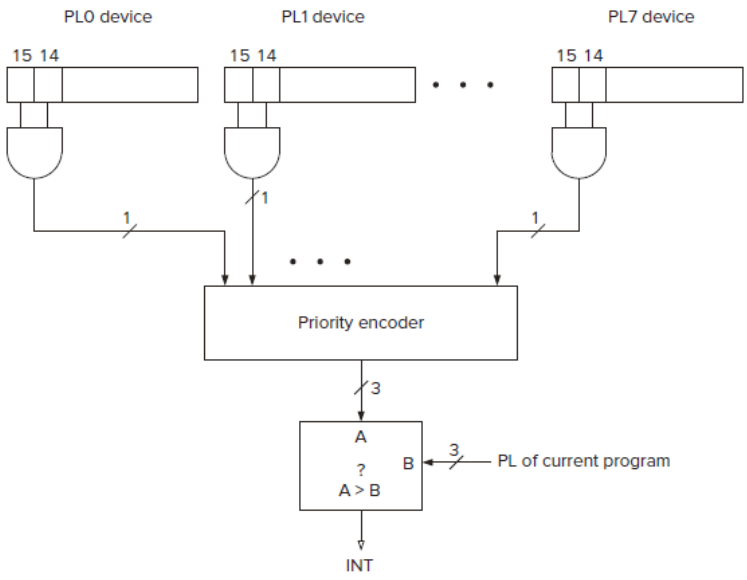
3. The device request **must be more urgent** than running program.

   The priority should be higher. To complete this, we have **INT signal**.

   NOTE: The priority of a interrupt is decided by I/O device, the programmer cannot change the priority in LC-3.

INT signal:

The interrupt only occurs when INT = 1.



NOTE: The test for INT signal only happens before the FETCH phrase of next instruction. This is for convenience.

9.4.4 The progress of Interrupt

There are 3 phrases:

1. Initial the Interrupt
    - Save the State of the Interrupted Program
        - Push PC & PSR of the running program into System Stack

          Assume the Interrupt routine always save the registers it will use
        - Change $R6$ if privilege change happens
    - Load the State of the Interrupt Service Routine
        - PC:

          Vectored Interrupts, use **Interrupt Vector Table** to store & find the start address of appropriate interrupt routine.

          Interrupt Vector Table: x0100 to x01FF

          Base + Offset -- Base: x0100, Offset: INTV(interrupt vector, 8bits)

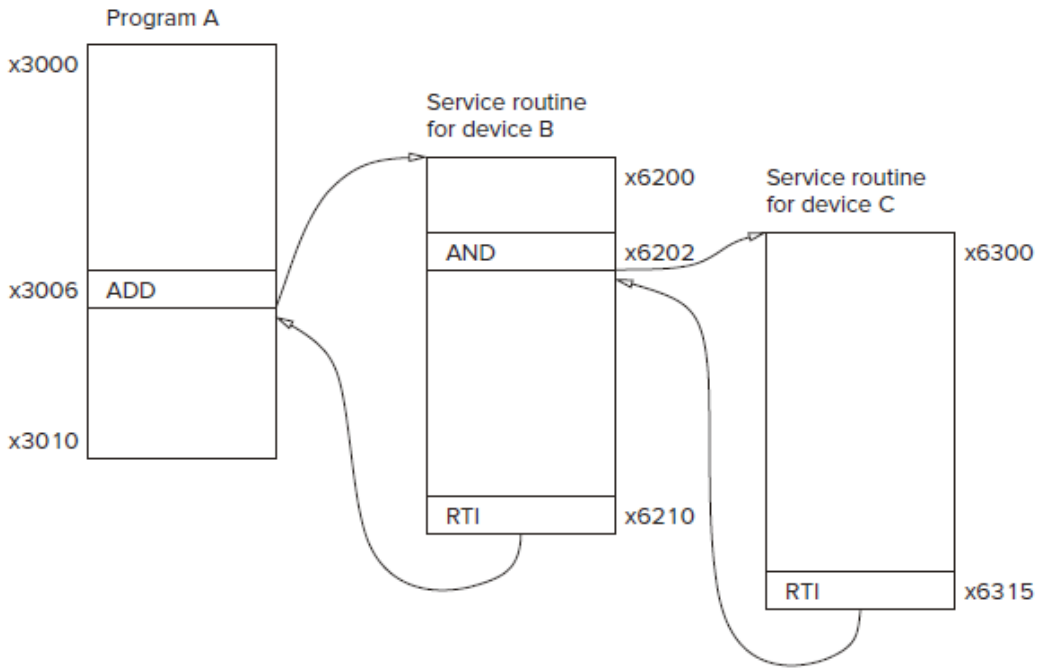
          Set PC to the start address of appropriate interrupt routine.
        - PSR:
            - PSR[2:0]: meaningless information since no instruction is executed. We usually set it to 010.
            - PSR[15]: 0 since interrupt routine is executed under supervisor privilege mode.
            - PSR[10:8]: set according to the priority level associated with interrupt request.
2. Service the Interrupt

   Just execute the Interrupt routine.
3. Return from the interrupt

   Use RTI instruction, it does
    - POP PC and PSR from the stack and reload them.

NOTE:

- Interrupt can happens when a interrupt routine is running (See example in 9.4.6)



- Not just I/O devices can cause interrupt. (See 9.4.7 for example)

9.5 Polling Revisited, Now That We Know About Interrupts

Chapter 9.5 in the 3rd textbook is recommend reading. It specifies the **improved version of polling program** due to the disturb of interrupt.