Chapter 8

I/O

I/O: Connecting to Outside World

So far, we've learned how to:

- **compute with values in registers**
- **load data from memory to registers**
- **store data from registers to memory**

Assignment Project Exam Help

But where does data in memory come from?

https://powcoder.com

Add WeChat powcoder

And how does data get out of the system so that humans can use it?

I/O: Connecting to the Outside World

Types of I/O devices characterized by:

- **behavior:** input, output, storage
  - ➢ input: keyboard, motion detector, network interface
  - ➢ output: monitor, printer, network interface
  - ➢ storage: disk, CD-ROM
- **data rate:** how fast can data be transferred?
  - ➢ keyboard: 100 bytes/sec
  - ➢ disk: 30 MB/s
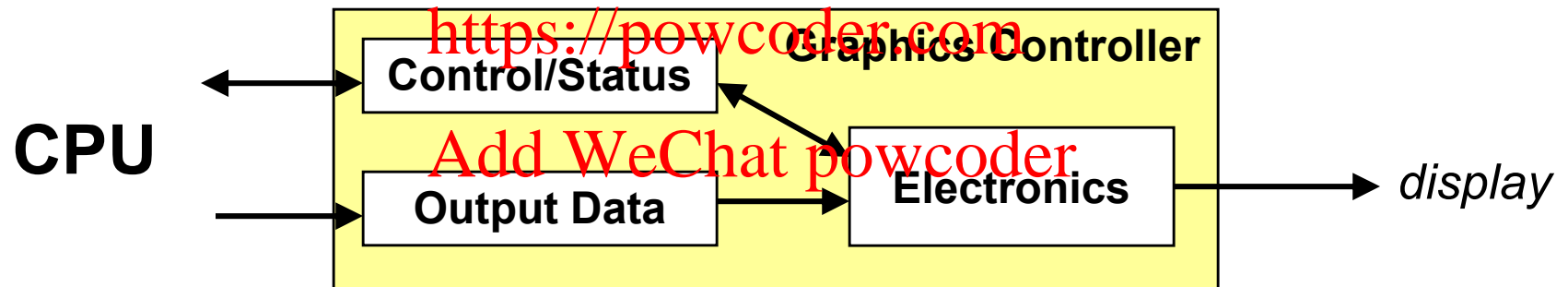 - ➢ network: 1 Mb/s - 1 Gb/s

I/O Controller

Control/Status Registers

- **CPU tells device what to do -- write to control register**
- **CPU checks whether task is done -- read status register**

Data Registers

Assignment Project Exam Help

- **CPU transfers data to/from device**

https://powcoder.com

| CPU | Control/Status | Graphics Controller |
| Output Data | Electronics | → display |

Add WeChat powcoder

Device electronics

- **performs actual operation**
  - ➢ **pixels to screen, bits to/from disk, characters from keyboard**

Programming Interface

How are device registers identified?

- **Memory-mapped** vs. **special instructions**

How is timing of transfer managed?

- **Asynchronous** vs. **synchronous**

Who controls transfer?

- CPU (**polling**) vs. device (**interrupts**)
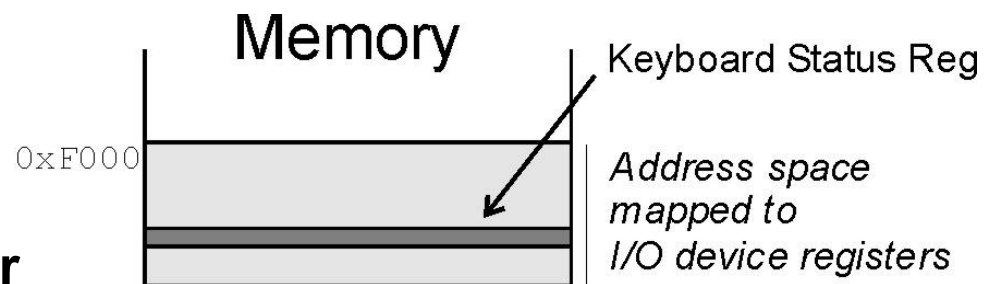
Memory-Mapped vs. I/O Instructions

Instructions

- **designate opcode(s) for I/O**
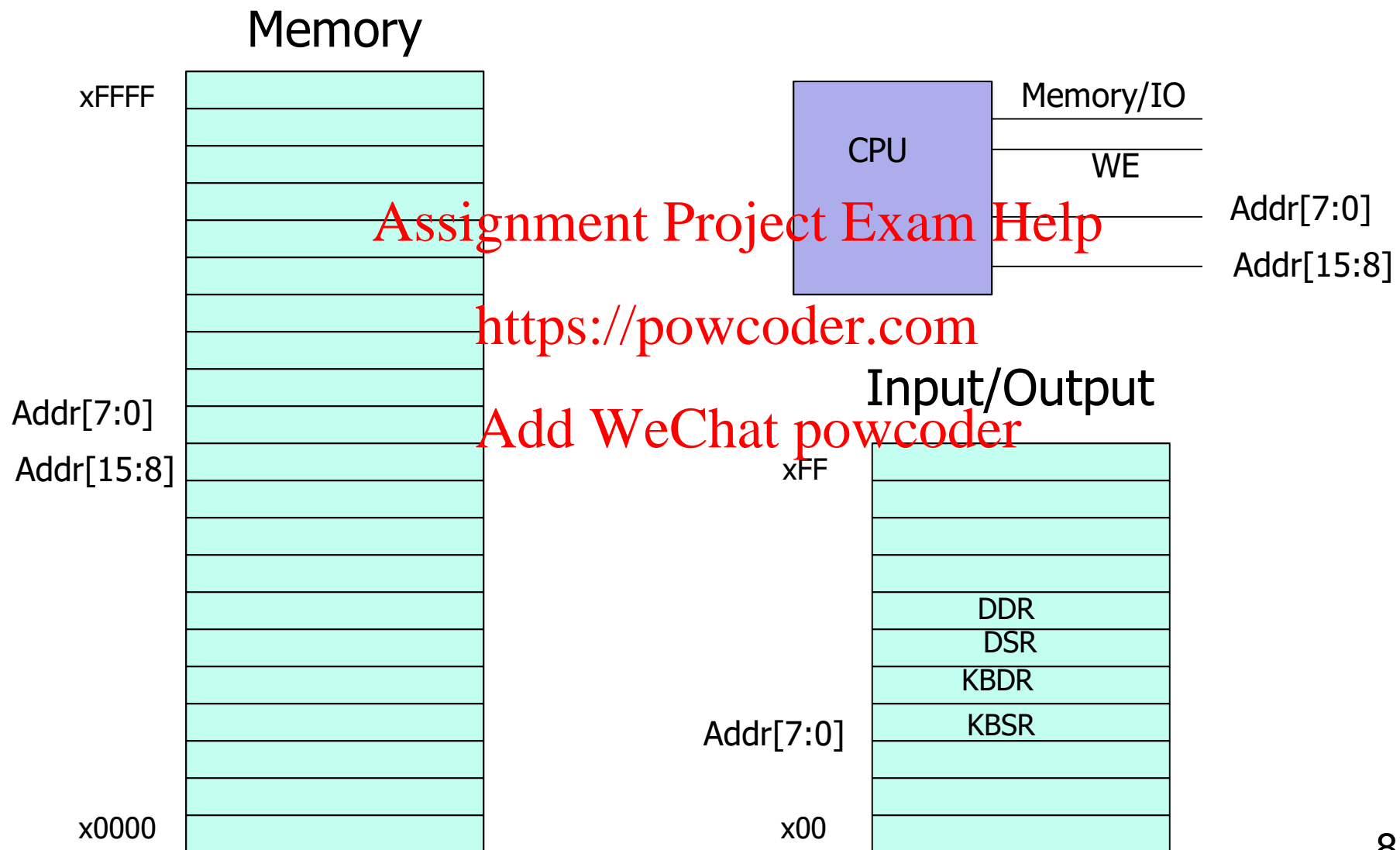- **register and operation encoded in instruction**

```
 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
+--------+--------------------------+------------+
|   IO   |         Device           |     Op     |
+--------+--------------------------+------------+
```

Memory-mapped

- **assign a memory address to each device register**
- **use data movement instructions (LD/ST) for control and data transfer**

Memory

0xF000

Keyboard Status Reg

Address space mapped to I/O device registers

Memory-Mapped vs. I/O Instructions
I/O-mapped

Memory

xFFFF

CPU

Memory/IO

WE

Addr[7:0]

Addr[15:8]

Assignment Project Exam Help

https://powcoder.com

Input/Output

Add WeChat powcoder

Addr[7:0]

Addr[15:8]

xFF

DDR

DSR

KBDR

KBSR

Addr[7:0]

x0000

x00

8-7

Transfer Timing

I/O events generally happen much slower than CPU cycles.

Synchronous Assignment Project Exam Help

- **data supplied at a fixed, predictable rate**
- **CPU reads/writes every X cycles** https://powcoder.com

Add WeChat powcoder

Asynchronous

- **data rate less predictable**
- **CPU must _synchronize_ with device, so that it doesn't miss data or write too quickly**

Transfer Control

Who determines when the next data transfer occurs?

Polling

- **CPU keeps checking status register until**
  new data arrives OR _device ready_ for next data

- **"Are we there yet? Are we there yet? Are we there yet?"**

Interrupts

- **Device sends a special signal to CPU when**
  new data arrives OR _device ready_ for next data

- **CPU can be performing other tasks instead of polling device.**

- **"Wake me when we get there."**

LC-3
Memory-mapped I/O (Table A.3)

Location	I/O Register	Function
xFE00	Keyboard Status Reg (KBSR)	Bit [15] is one when keyboard has received a new character.
xFE02	Keyboard Data Reg (KBDR)	Bits [7:0] contain the last character typed on keyboard.
xFE04	Display Status Register (DSR)	Bit [15] is one when device ready to display another char on screen.
xFE06	Display Data Register (DDR)	Character written to bits [7:0] will be displayed on screen.

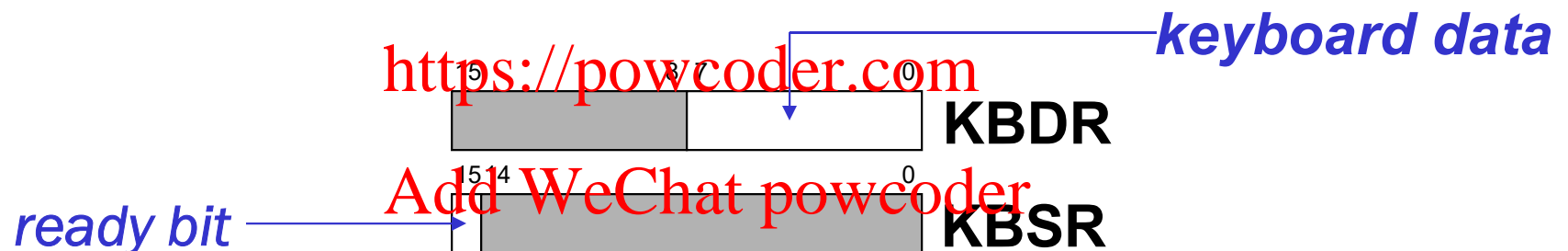## Asynchronous devices
- **synchronized through status registers**

Polling and Interrupts
- **the details of interrupts will be discussed in Chapter 10**

8-10

Input from Keyboard

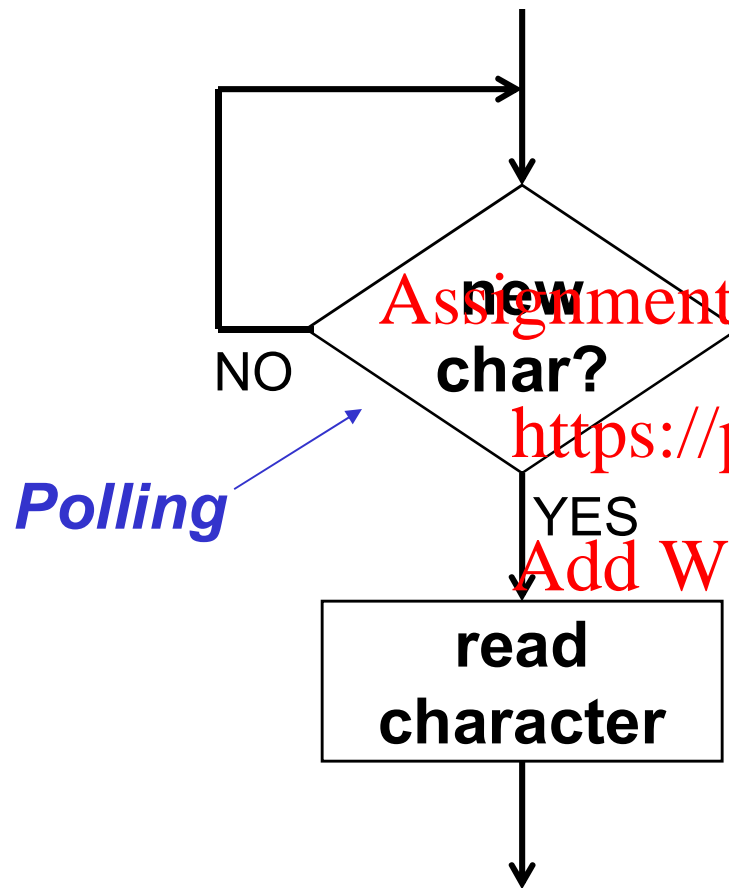## When a character is typed:

- its ASCII code is placed in bits [7:0] of KBDR
  (bits [15:8] are always zero)

- the "ready bit" (KBSR[15]) is set to one

- keyboard is disabled -- any typed characters will be ignored

keyboard data
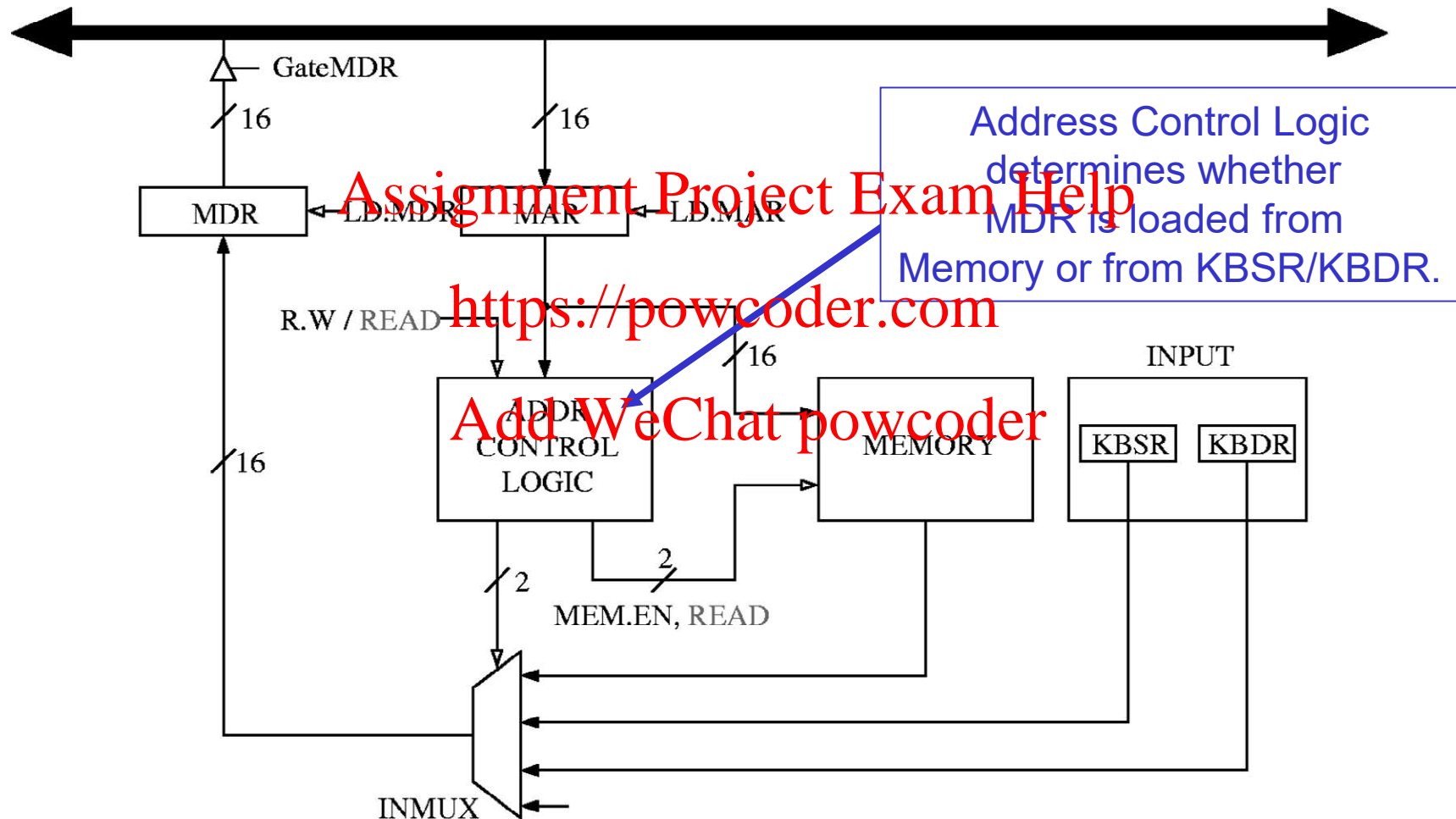
```
 15        8 7        0
┌──────────┬──────────┐
│░░░░░░░░░░│          │  KBDR
└──────────┴──────────┘
 15 14              0
┌──┬─────────────────┐
│  │░░░░░░░░░░░░░░░░░░│  KBSR
└──┴─────────────────┘
```

ready bit

When KBDR is read:

- KBSR[15] is set to zero

- keyboard is enabled

Basic Input Routine

```
POLL    LDI  R0, KBSRPtr
        BRzp POLL
        LDI  R0, KBDRPtr
        ...
KBSRPtr .FILL xFE00
KBDRPtr .FILL xFE02
```
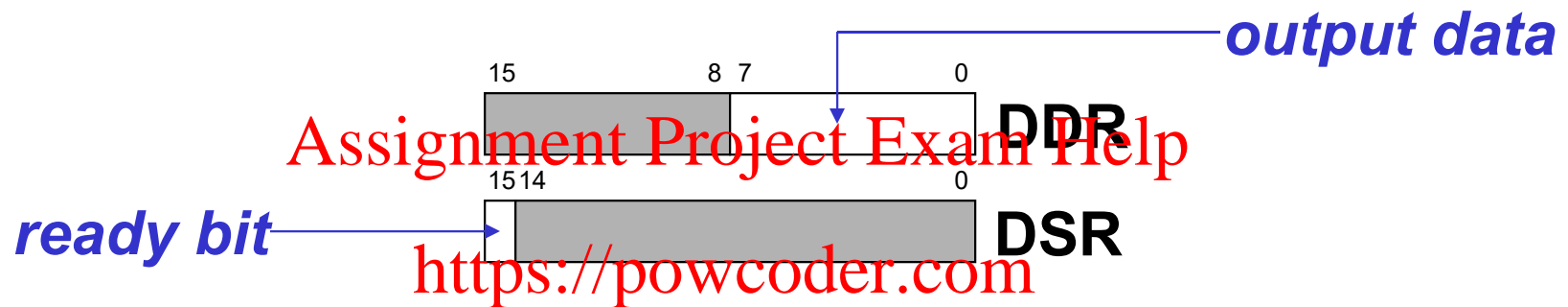
new char?

NO

Polling

YES

read character

Simple Implementation: Memory-Mapped Input

Address Control Logic determines whether MDR is loaded from Memory or from KBSR/KBDR.

Output to Monitor

When Monitor is ready to display another character:

- the "ready bit" (DSR[15]) is set to one

output data

```
15        8 7        0
```
DDR

ready bit
```
1514              0
```
DSR

When data is written to Display Data Register:
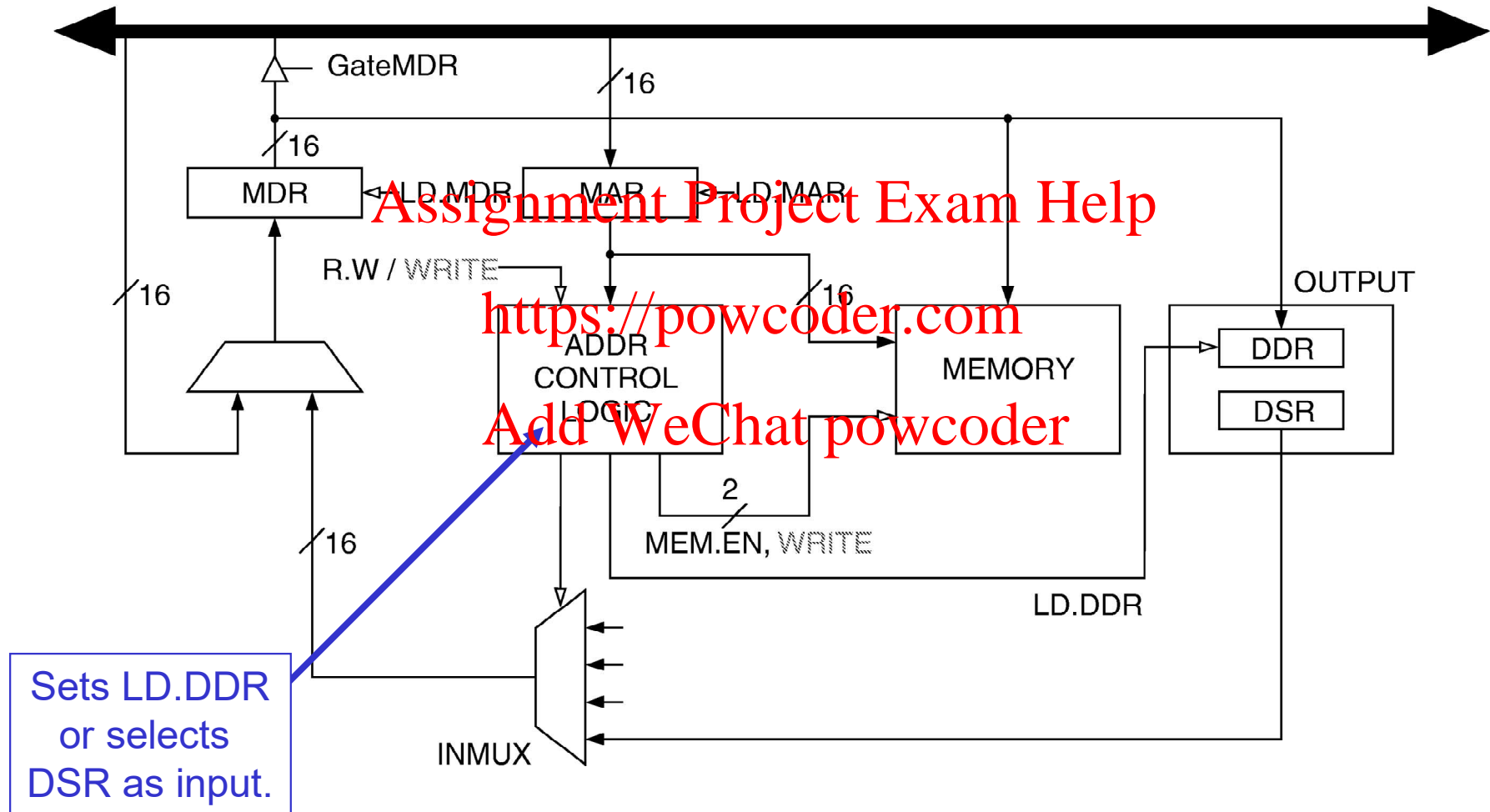
- DSR[15] is set to zero
- character in DDR[7:0] is displayed
- any other character data written to DDR is ignored
  (while DSR[15] is zero)

8-14

Basic Output Routine

NO

screen ready?

Polling

YES

write character
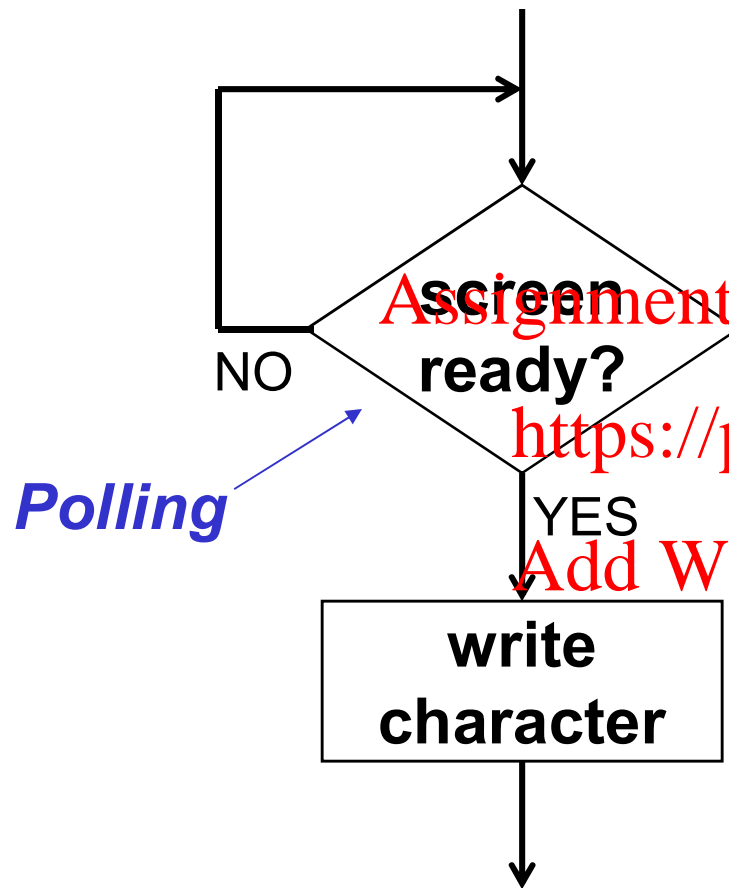
```
POLL    LDI  R1, DSRPtr
        BRzp POLL
        STI  R0, DDRPtr

        ...

DSRPtr .FILL xFE04
DDRPtr .FILL xFE06
```

Simple Implementation: Memory-Mapped Output

Sets LD.DDR or selects DSR as input.

Keyboard Echo Routine

Usually, input character is also printed to screen.

- **User gets feedback on character typed and knows its ok to type the next character.**

```
POLL1    LDI   R0, KBSRPtr
         BRzp  POLL1
         LDI   R0, KBDRPtr
POLL2    LDI   R1, DSRPtr
         BRzp  POLL2
         STI   R0, DDRPtr

         ...

KBSRPtr  .FILL xFE00
KBDRPtr  .FILL xFE02
DSRPtr   .FILL xFE04
DDRPtr   .FILL xFE06
```

new char?

NO

YES

read character

screen ready?

NO

YES

write character

8-17
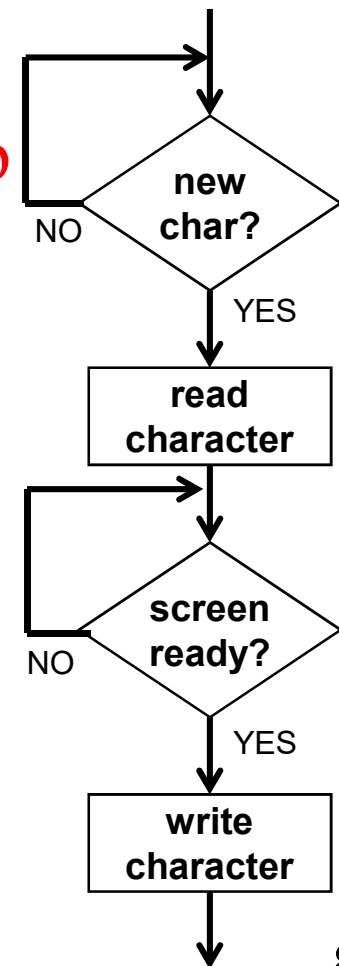
Polling Overhead

Options for polling

- **Don't always need to sit in tight polling loop, doing nothing else.**

- **But must poll device often enough to not miss any potential data transfers.**

Assignment Project Exam Help

Example: mouse https://powcoder.com

- **Must poll mouse at least 30 times per second, to see if (a) change of position and/or (b) button pressed/released.** Add WeChat powcoder

- **How much <u>overhead</u> does this represent?**

  ➤ In other words, what percentage of CPU cycles will be used up checking for mouse events?

Mouse Overhead -- Polling

Assume the following parameters:

- **CPU clock is 1 GHz (1 x 10^9 cycles/sec)**

- **time to service device is 400 cycles, including:**

  ➤ transfer to polling routine, checking status, reading data, updating cursor, returning to user code

How much overhead?

- **time spent polling = 30 x 400 = 12,000 cycles/sec**

- **fraction of CPU cycles = 12 x 10^3 / 1 x 10^9 = 0.00012%**

- **Conclusion: Polling is not a lot of overhead in this case.**

- Not counted -- how to do we know when to poll? Timer interrupt?

Network Overhead -- Polling

What about a network device?

- 100 Mb/s Ethernet, each "packet" is 40B

- Have to poll often enough to not drop data.

- 1 GHz processor, 400 cycles to poll + read data

How much overhead?

- $(320 \text{ bits} / 10^8 \text{ bits/sec}) \times (10^9 \text{ cycles/sec})$
  = 1 packet every 3200 cycles = 312,500 packets/sec

- fraction of CPU cycles
  = $(400 * 312{,}500 \text{ cycles/sec}) / (10^9 \text{ cycles/sec})$ = **12.5%**

- This seems a little much -- will interrupts help?

Interrupt-Driven I/O

External device can:

(1) Force currently executing program to stop;

(2) Have the processor satisfy the device's needs; and

(3) Resume the stopped program as if nothing happened.

Why?

- **Polling consumes a lot of cycles, especially for <u>rare</u> events – these cycles can be used for more computation.**

- **Example: Process previous input while collecting current input. (See Example 8.1 in text - page 221.)**

Network Overhead -- Interrupts

Why interrupts?

- **Network is not "active" 100% of the time.
  Many times, we'll poll and find out there's no data to get.**
- **Interrupt will happen ONLY when there's actually a packet to be transferred.**
- **Let's assume network is active 10% of the time.**

Overhead?

- **312,500 packets/sec x 10% = 31,250 packets/sec**
- **fraction of CPU cycles
  = (400 x 31,250 cycles/sec) / (10^9 cycles/sec) = 1.25%**

Priority

Every instruction executes at a stated level of urgency.

LC-3: 8 priority levels (PL0-PL7)

- **Example:**
  - ➢ **Payroll program runs at PL0.**
  - ➢ **Nuclear power correction program runs at PL6.**
- **It's OK for PL6 device to interrupt PL0 program, but not the other way around.**

Priority encoder selects highest-priority device, compares to current processor priority level, and generates interrupt signal if appropriate.

Interrupt-Driven I/O

To implement an interrupt mechanism, we need:

- A way for the I/O device to **signal** the CPU that an interesting event has occurred.
- A way for the CPU to **test** whether the **interrupt signal is set** and whether its **priority is higher** than the current program.
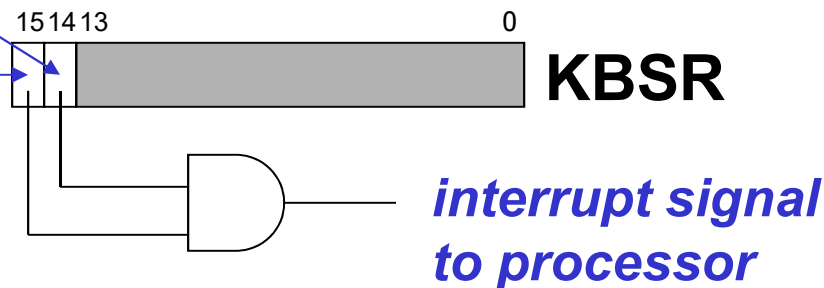
Generating Signal

- Software sets "interrupt enable" bit in device register.
- When ready bit is set and IE bit is set, interrupt is signaled.
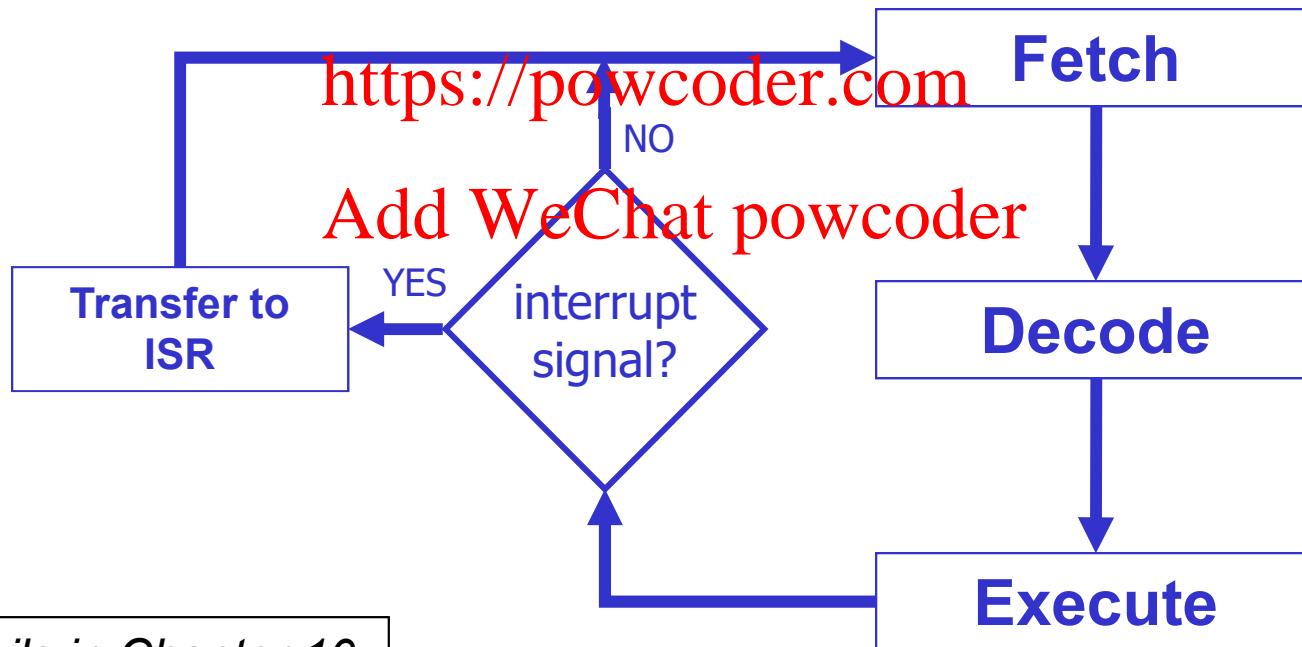
interrupt enable bit

ready bit

15 14 13   0

KBSR

*interrupt signal
to processor*

Testing for Interrupt Signal

CPU looks at signal between EXECUTE and FETCH phases.
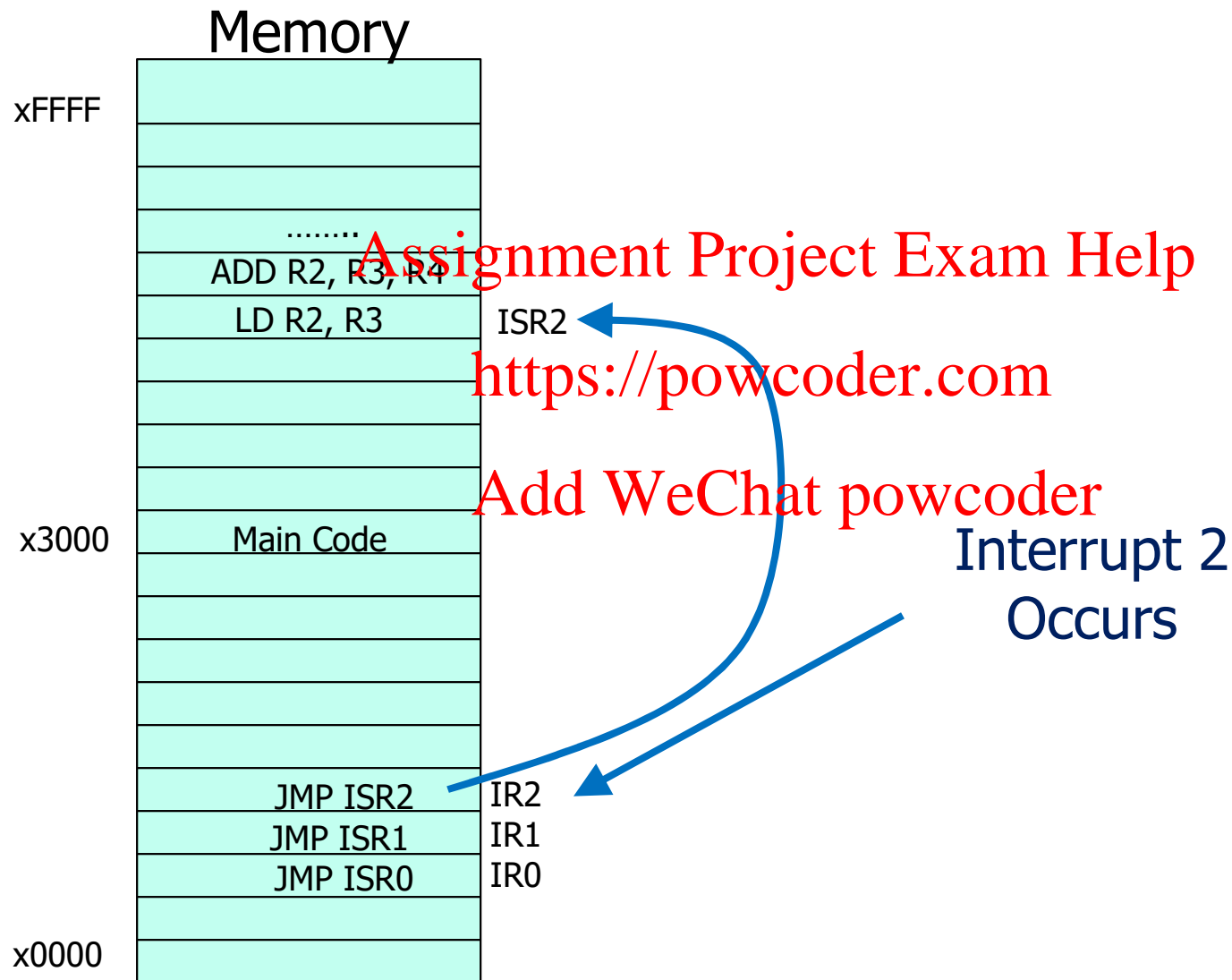
If not set, continues with next instruction.

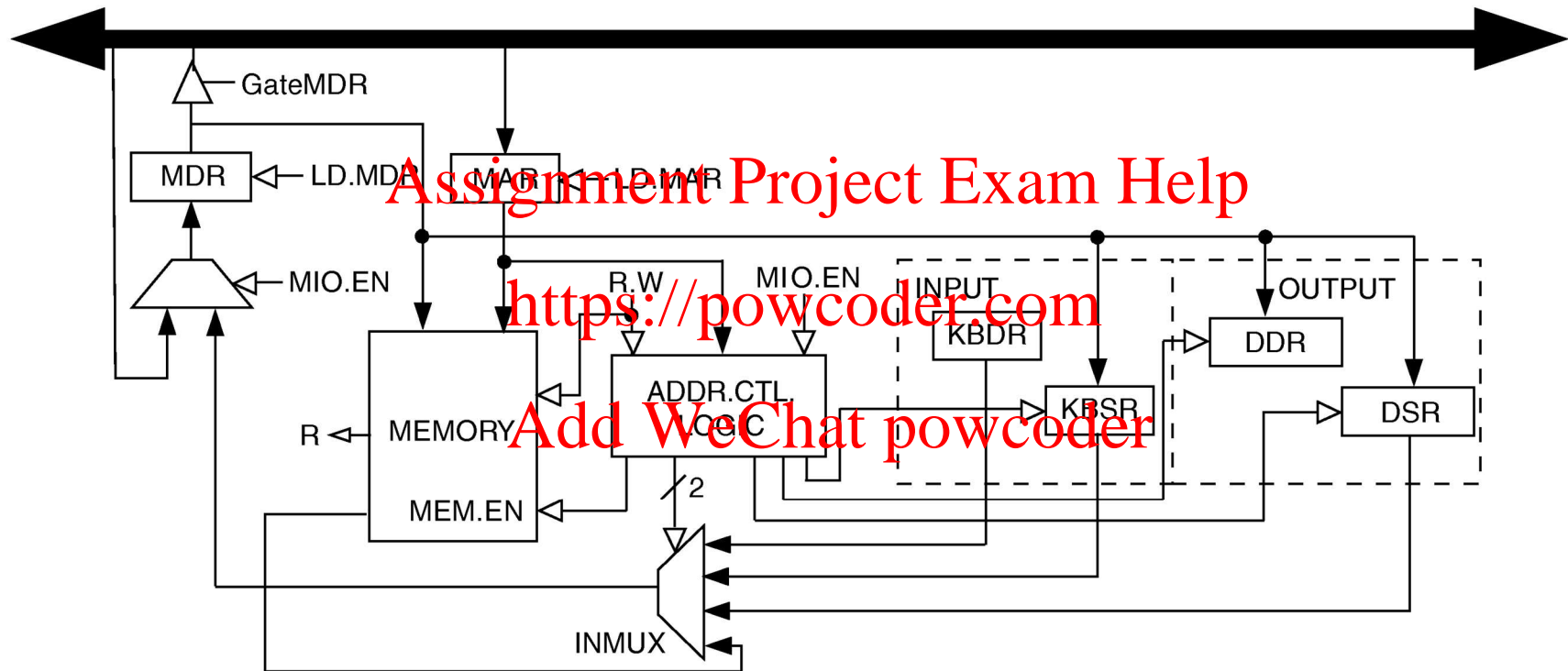**If set, transfers control to interrupt service routine.**

More details in Chapter 10.

8-25

Interrupt Service Routines (ISR)

Memory

xFFFF

....
ADD R2, R3, R4

LD R2, R3   ISR2

x3000   Main Code

JMP ISR2   IR2
JMP ISR1   IR1
JMP ISR0   IR0

x0000

Interrupt 2
Occurs

Full Implementation of LC-3 Memory-Mapped I/O

Because of interrupt enable bits, status registers (KBSR/DSR) must be written, as well as read.