KU

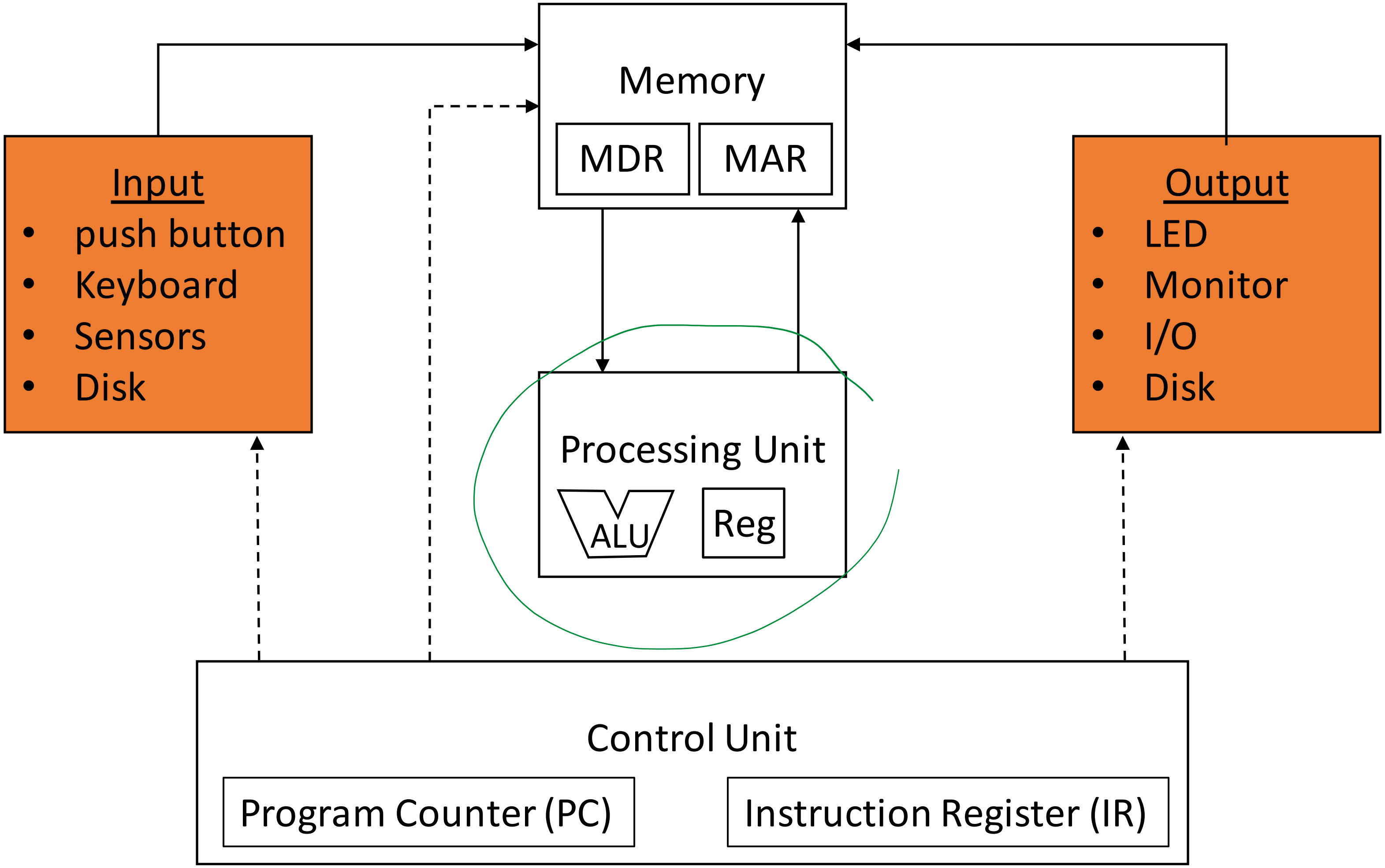
THE UNIVERSITY OF
KANSAS

I/O

EECS388 Fall 2022

Context

- Recommended reading: Chapter 9 of “Introduction to Computing,” Patt, Patel



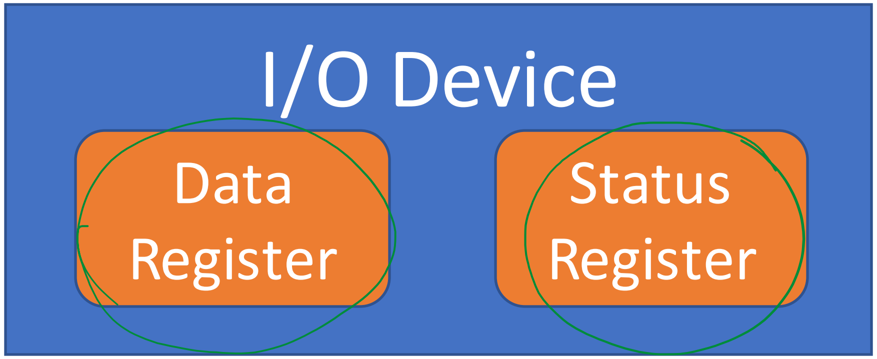
** Picture in the slides are in part taken from “Introduction to Computing From bits & gates to C and beyond” Patt, Patel

- **Fact**: everything in a computer (including I/O devices) is controlled by the instructions in the ISA
- **Question 1**: Does ISA need special instructions for controlling I/O?
- **Question 2**: Does the I/O device need to work at the same speed of the processing unit?
- **Question 3**: Does I/O transfer is initiated by a program or I/O device?

We will answer these question in this lecture.

Simplified Processing unit – I/O device interaction

- Using two registers:
  - Data register: Hold the data being transferred
  - Status register: status of the device (e.g., busy, need attention, etc.)



- **Fact**: everything in a computer (including I/O devices) is controlled by the instructions in the ISA
- **Question 1**: Does ISA need special instructions for controlling I/O?
- **Question 2**: Does the I/O device need to work at the same speed of the processing unit?
- **Question 3**: Does I/O transfer is initiated by a program or I/O device?

How to interact with I/O devices?

Schemes:

1. Special I/O instructions
2. Using data movement instructions

How to interact with I/O devices?

Schemes:

1. Special I/O instructions
2. Using data movement instructions

Special I/O Instructions

instrX <Operands>

- Identify:
  - Which I/O device
  - The operation
  - The operands

- E.g., 1965 DEC PDP-8 computer

110 <9 bits specify Operation and I/O register number>

Opcode

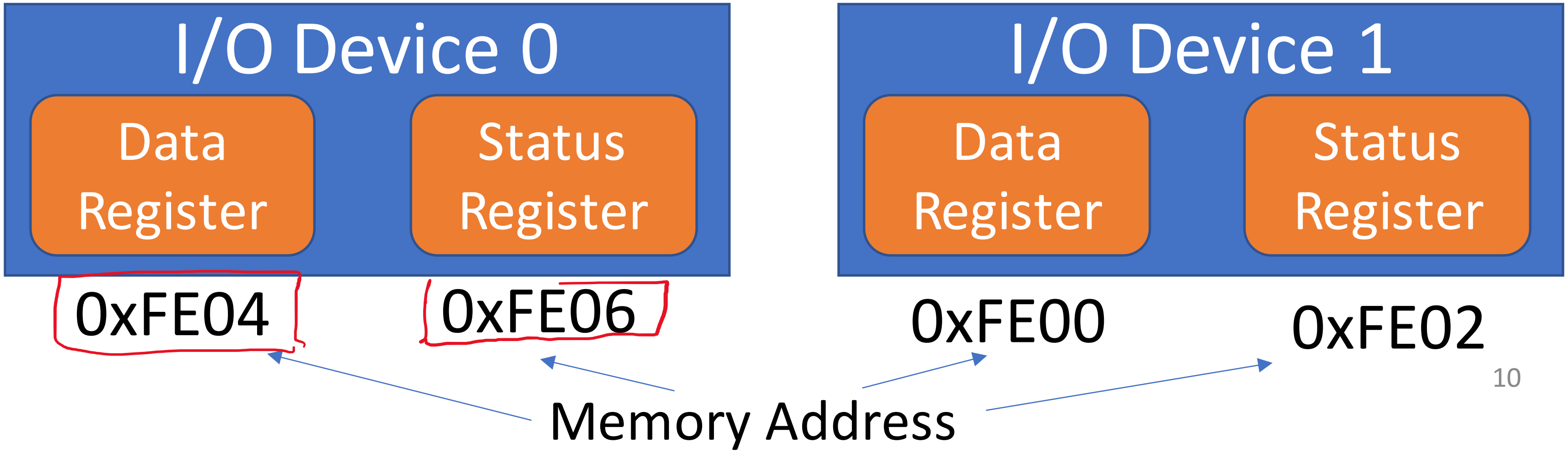
How to interact with I/O devices?

Schemes:

1. Special I/O instructions
2. Using data movement instructions

Memory Mapped I/O

- LD/LDI/LDR <Device Register Address>
  - Input instruction
- ST/STI/STR <Device Register Address>
  - Output instruction



I/O Device 0

Data Register

Status Register

I/O Device 1

Data Register

Status Register

0xFE04

0xFE06

0xFE00

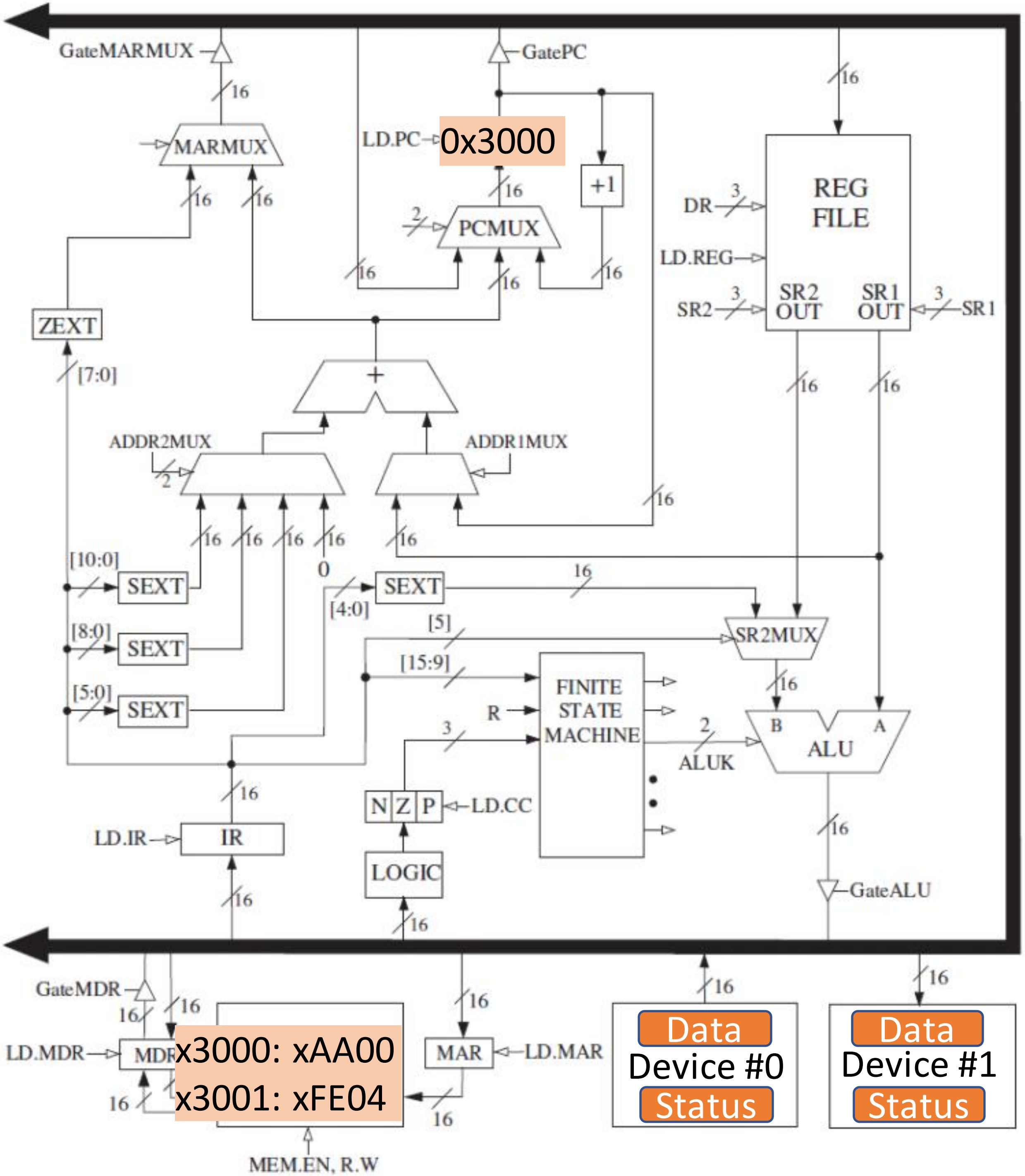
0xFE02

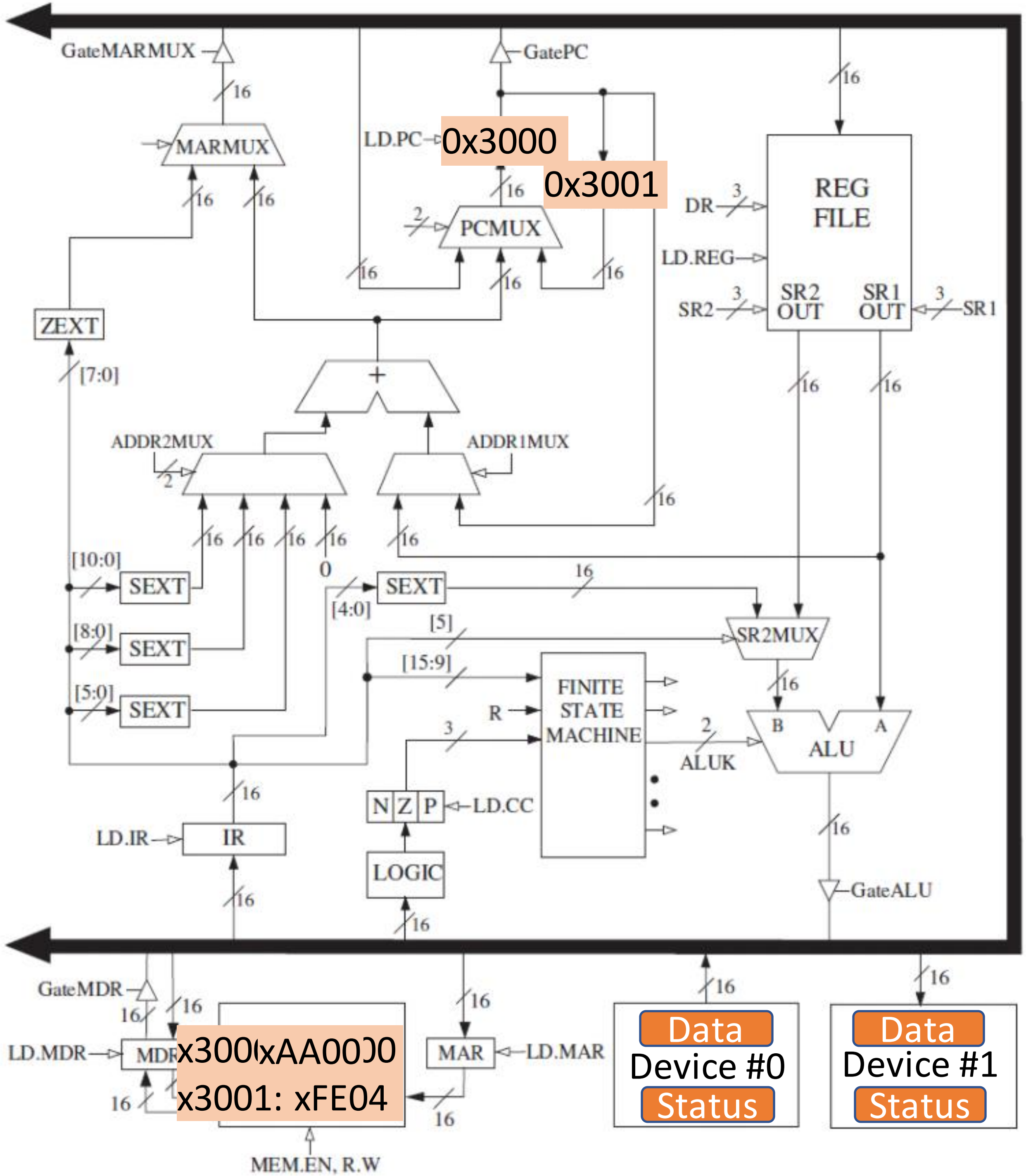
Memory Address

Memory Mapped I/O Example

```
        .ORIG x3000
        LDI   R5, DDR0
DDR0    .FILL xFE04
```

Read 2 bytes from I/O device 0 and store in R5





Memory Mapped I/O Example

```
        .ORIG x3000
        LDI   R5, DDR0
DDR0    .FILL xFE04
```

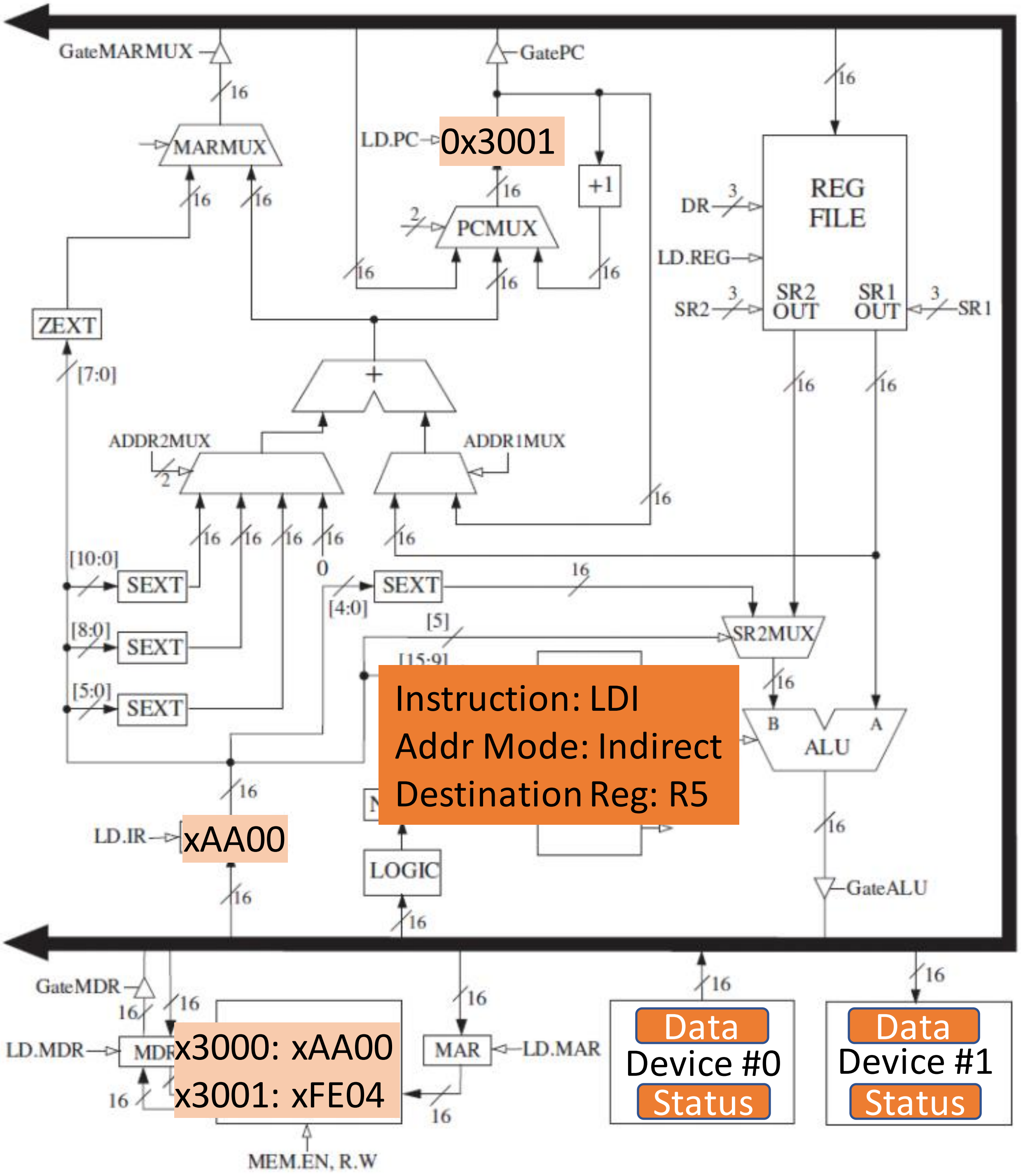
Read 2 bytes from I/O device 0 and store in R5

1-Fetch
2-Decode
3-Evaluate Address
4-Fetch Operands
5-Execute
6-Write Results

Memory Mapped I/O Example

```
            .ORIG x3000
            LDI   R5, DDR0
DDR0        .FILL xFE04
```

Read 2 bytes from I/O device 0 and store in R5



0x3001

Instruction: LDI
Addr Mode: Indirect
Destination Reg: R5

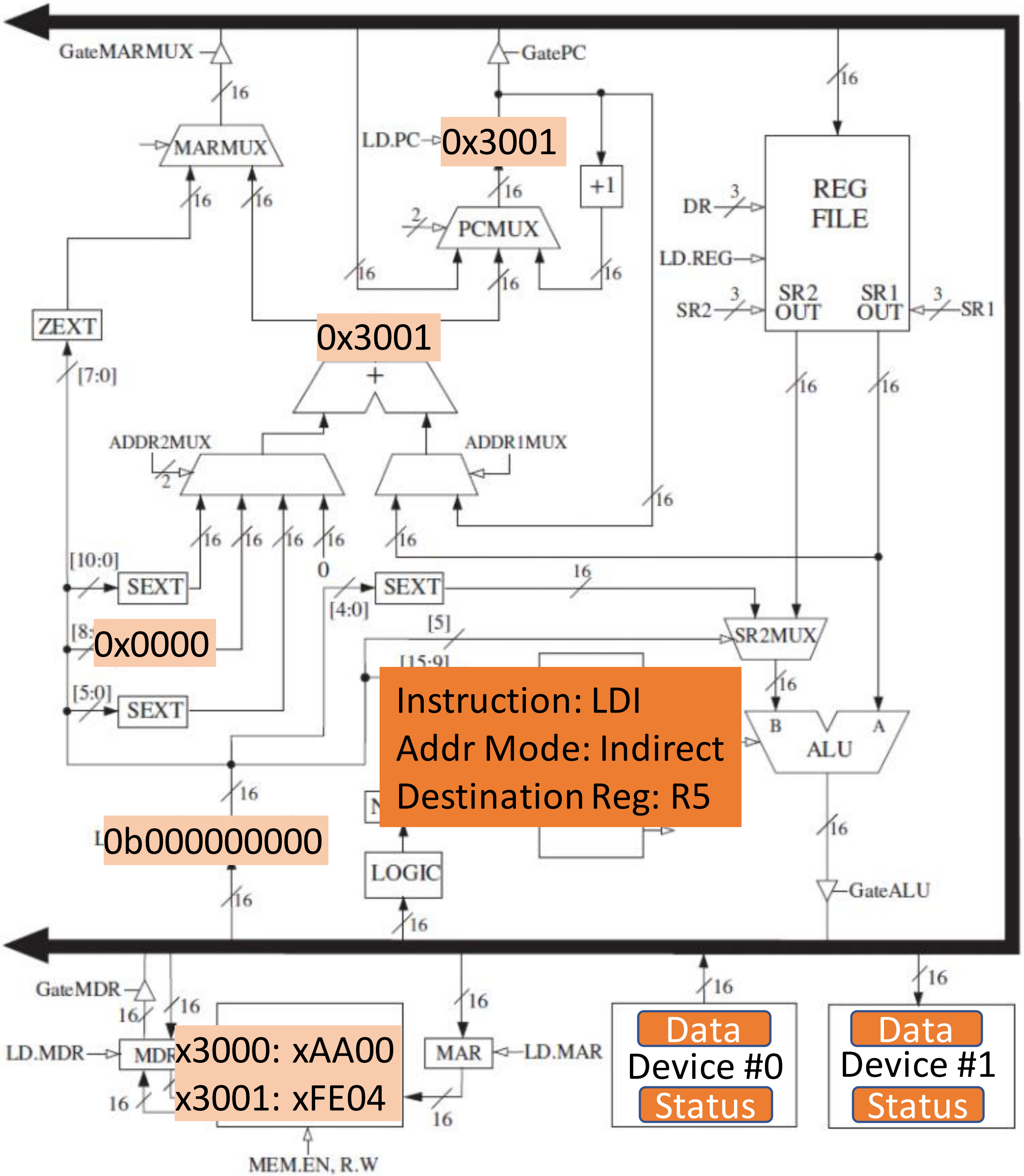
xAA00

x3000: xAA00
x3001: xFE04

Data
Device #0
Status

Data
Device #1
Status

1-Fetch
2-Decode
3-Evaluate Address
4-Fetch Operands
5-Execute
6-Write Results



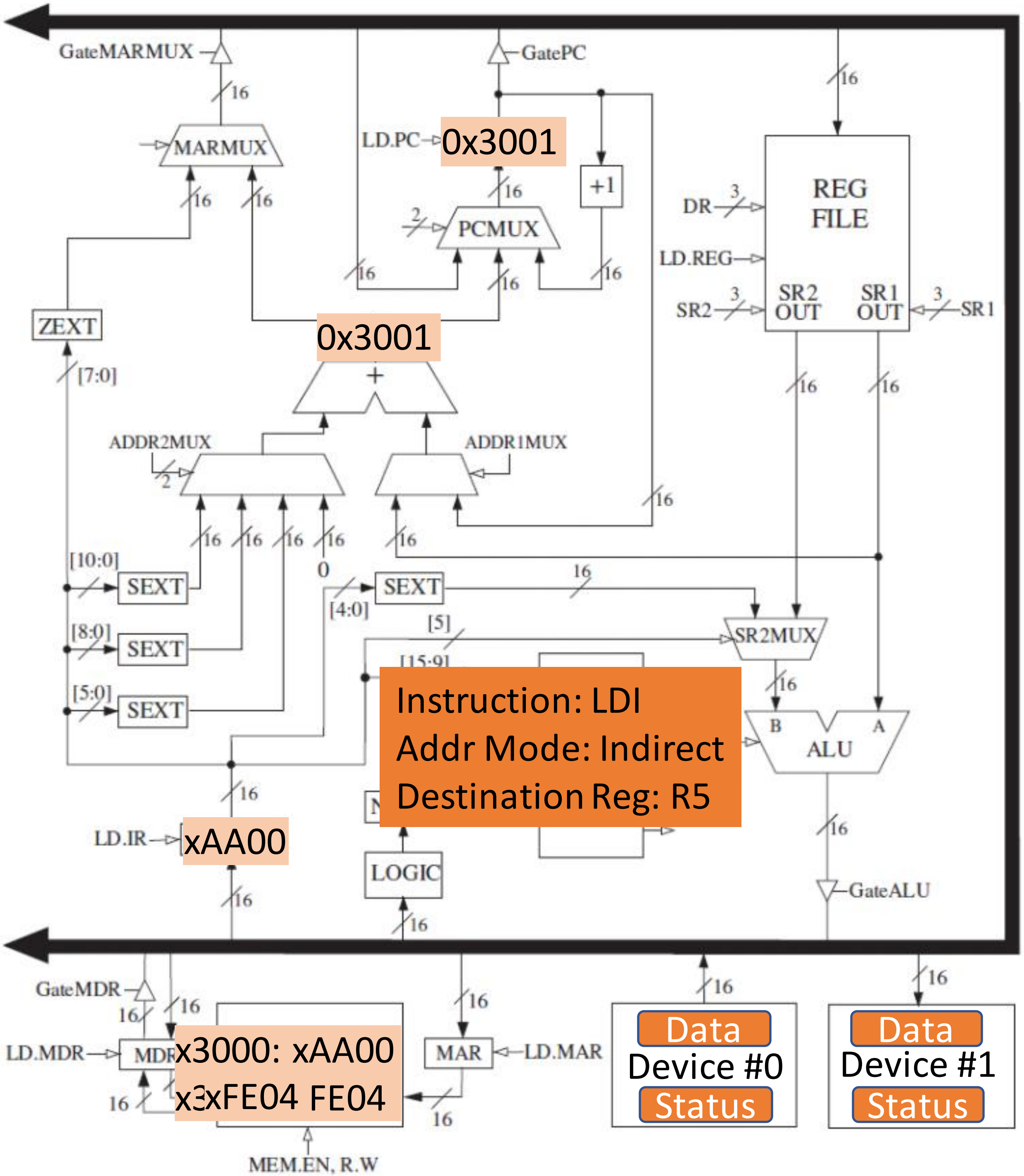
Instruction: LDI
Addr Mode: Indirect
Destination Reg: R5

Memory Mapped I/O Example

```
        .ORIG x3000
        LDI   R5, DDR0
DDR0    .FILL xFE04
```

Read 2 bytes from I/O device 0 and store in R5

1-Fetch
2-Decode
3-Evaluate Address
4-Fetch Operands
5-Execute
6-Write Results



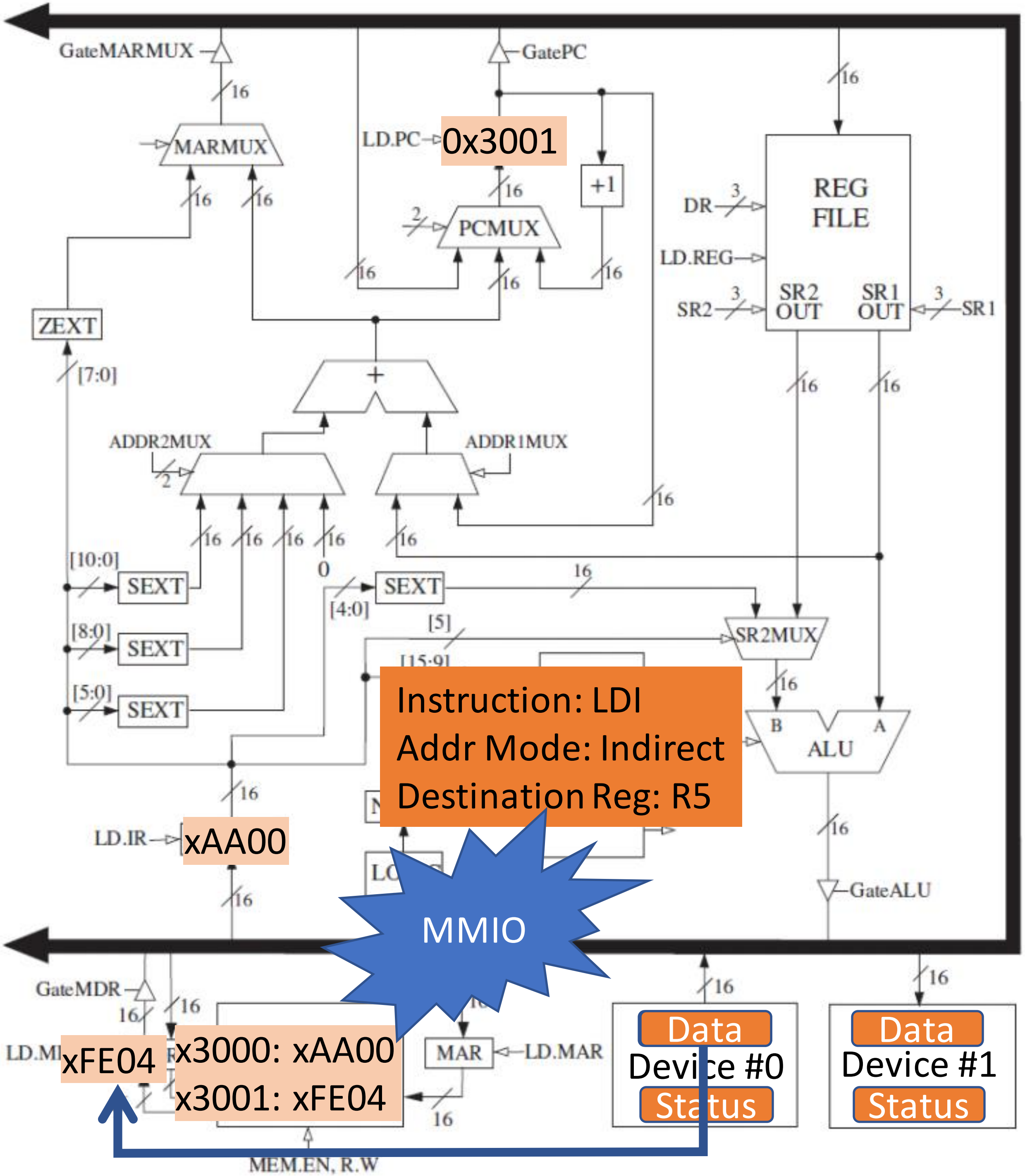
Instruction: LDI
Addr Mode: Indirect
Destination Reg: R5

Memory Mapped I/O Example

```
        .ORIG x3000
        LDI   R5, DDR0
DDR0    .FILL xFE04
```

Read 2 bytes from I/O device 0 and store in R5

1-Fetch
2-Decode
3-Evaluate Address
4-Fetch Operands
5-Execute
6-Write Results



0x3001

xAA00

Instruction: LDI
Addr Mode: Indirect
Destination Reg: R5

xFE04

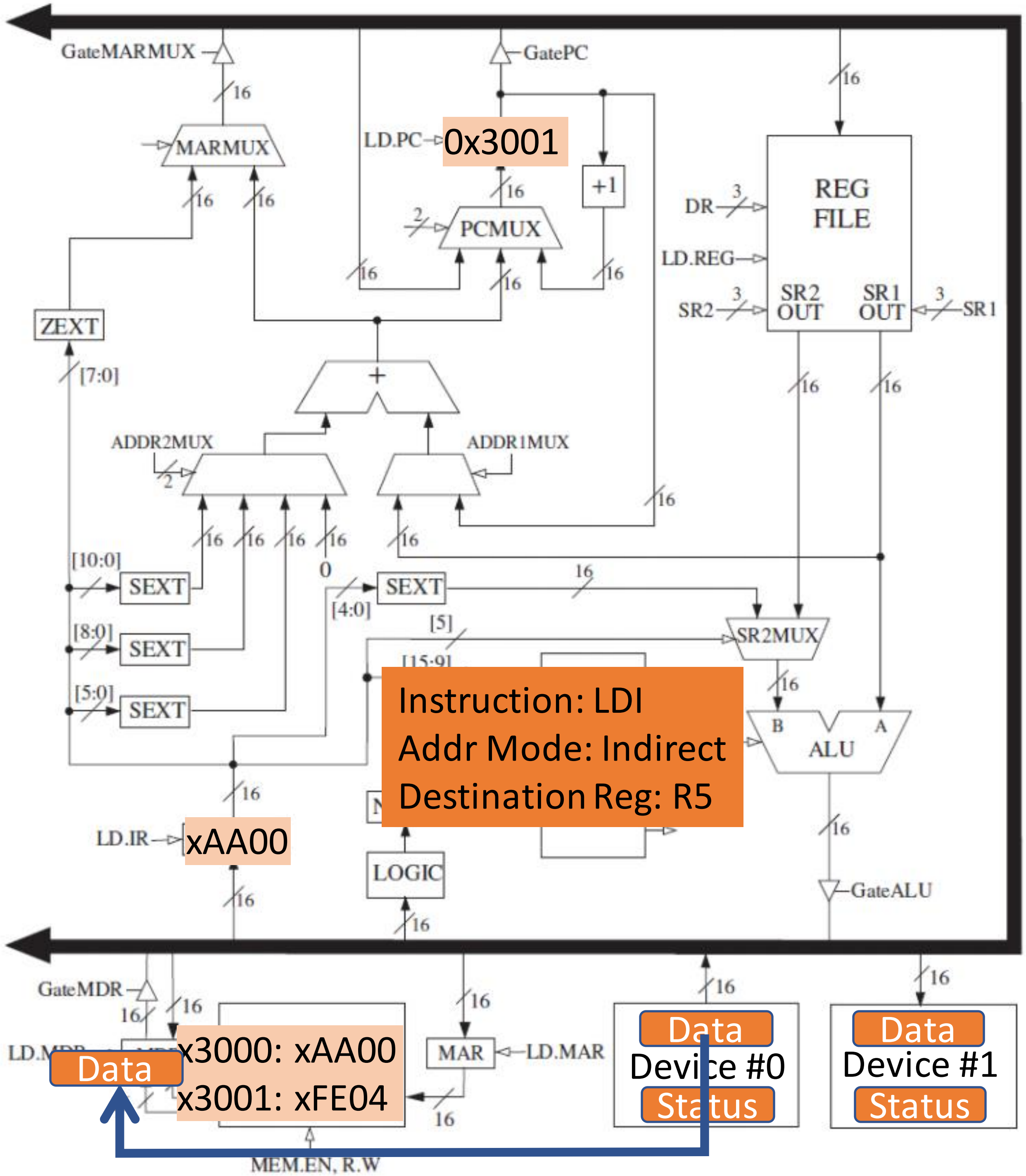
x3000: xAA00
x3001: xFE04

Memory Mapped I/O Example

```
        .ORIG x3000
        LDI   R5, DDR0
DDR0    .FILL xFE04
```

Read 2 bytes from I/O device 0 and store in R5

1-Fetch
2-Decode
3-Evaluate Address
4-Fetch Operands
5-Execute
6-Write Results

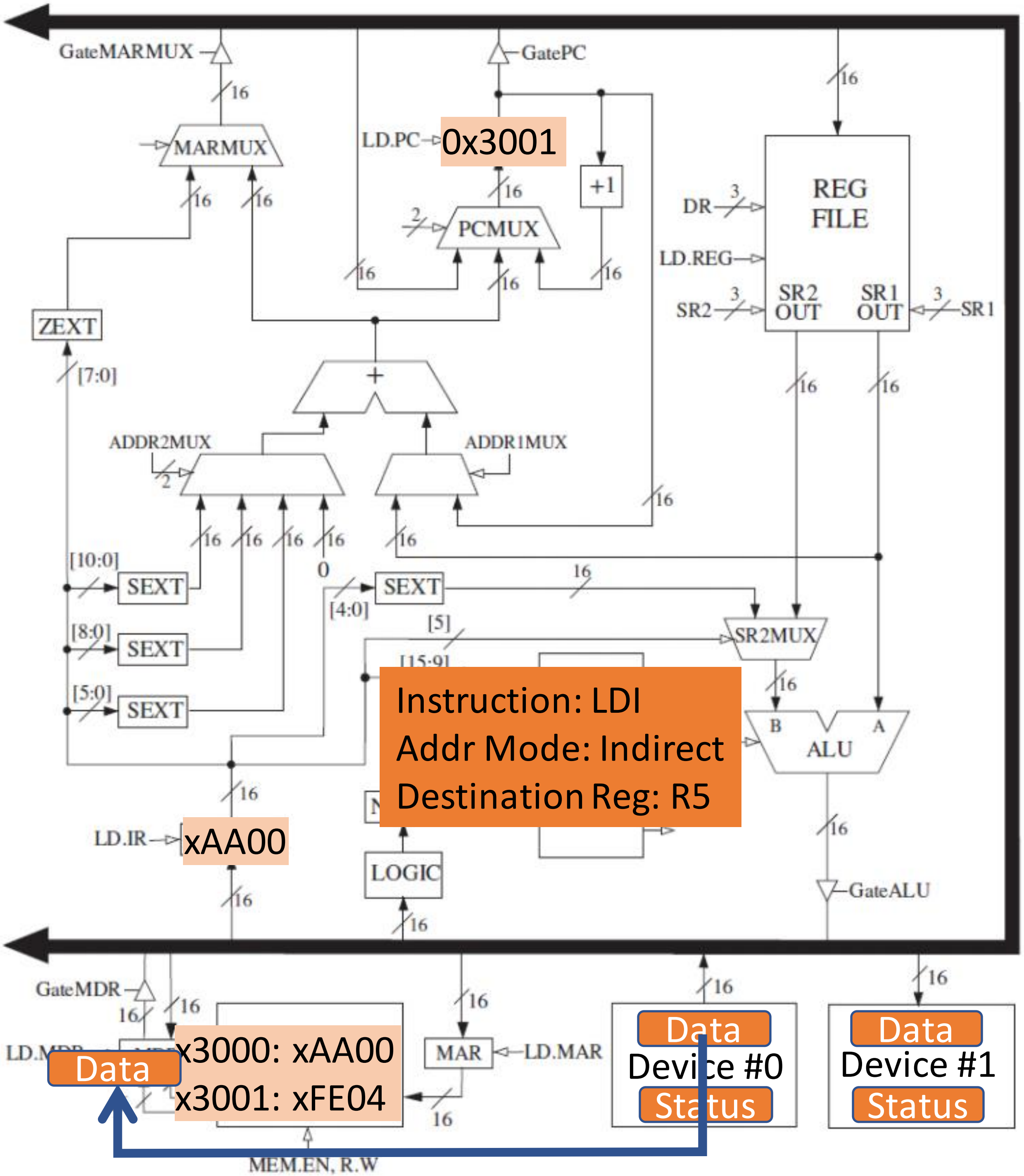


Memory Mapped I/O Example

```
        .ORIG x3000
        LDI   R5, DDR0
DDR0    .FILL xFE04
```

Read 2 bytes from I/O device 0 and store in R5

1-Fetch
2-Decode
3-Evaluate Address
4-Fetch Operands
5-Execute
6-Write Results



0x3001

Instruction: LDI
Addr Mode: Indirect
Destination Reg: R5

xAA00

x3000: xAA00
x3001: xFE04

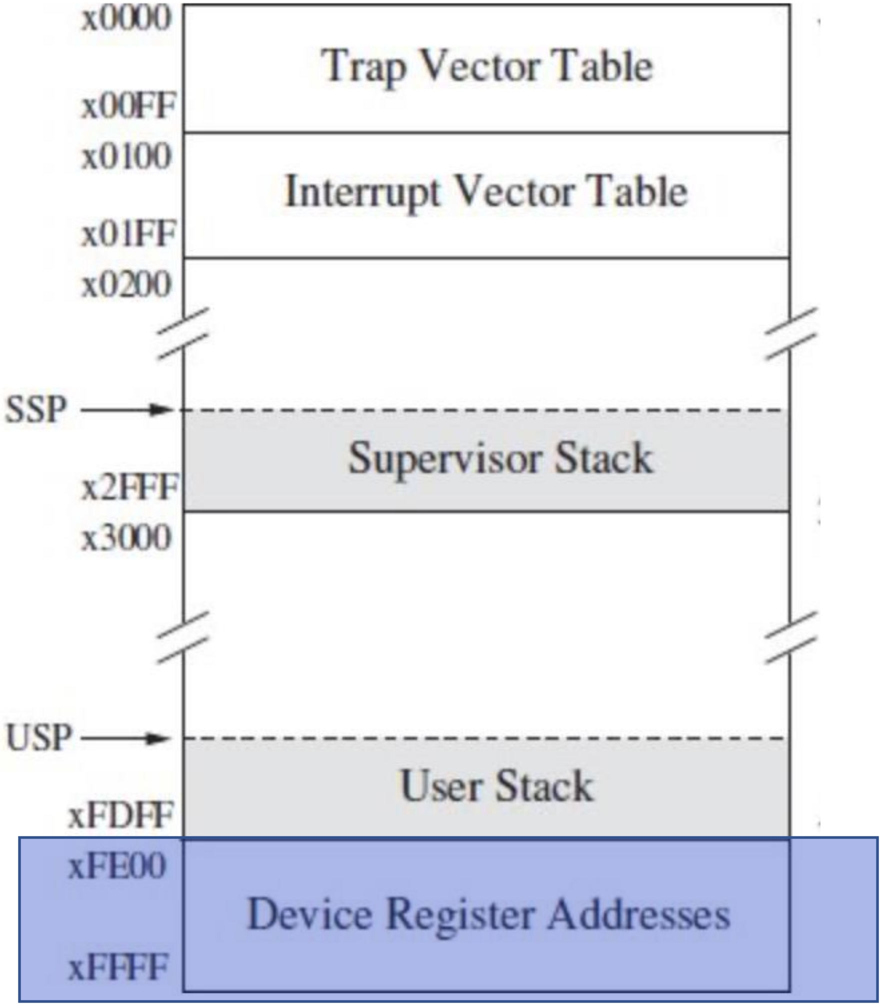
Memory Mapped I/O Example

```
        .ORIG x3000
        LDI   R5, DDR0
DDR0    .FILL xFE04
```

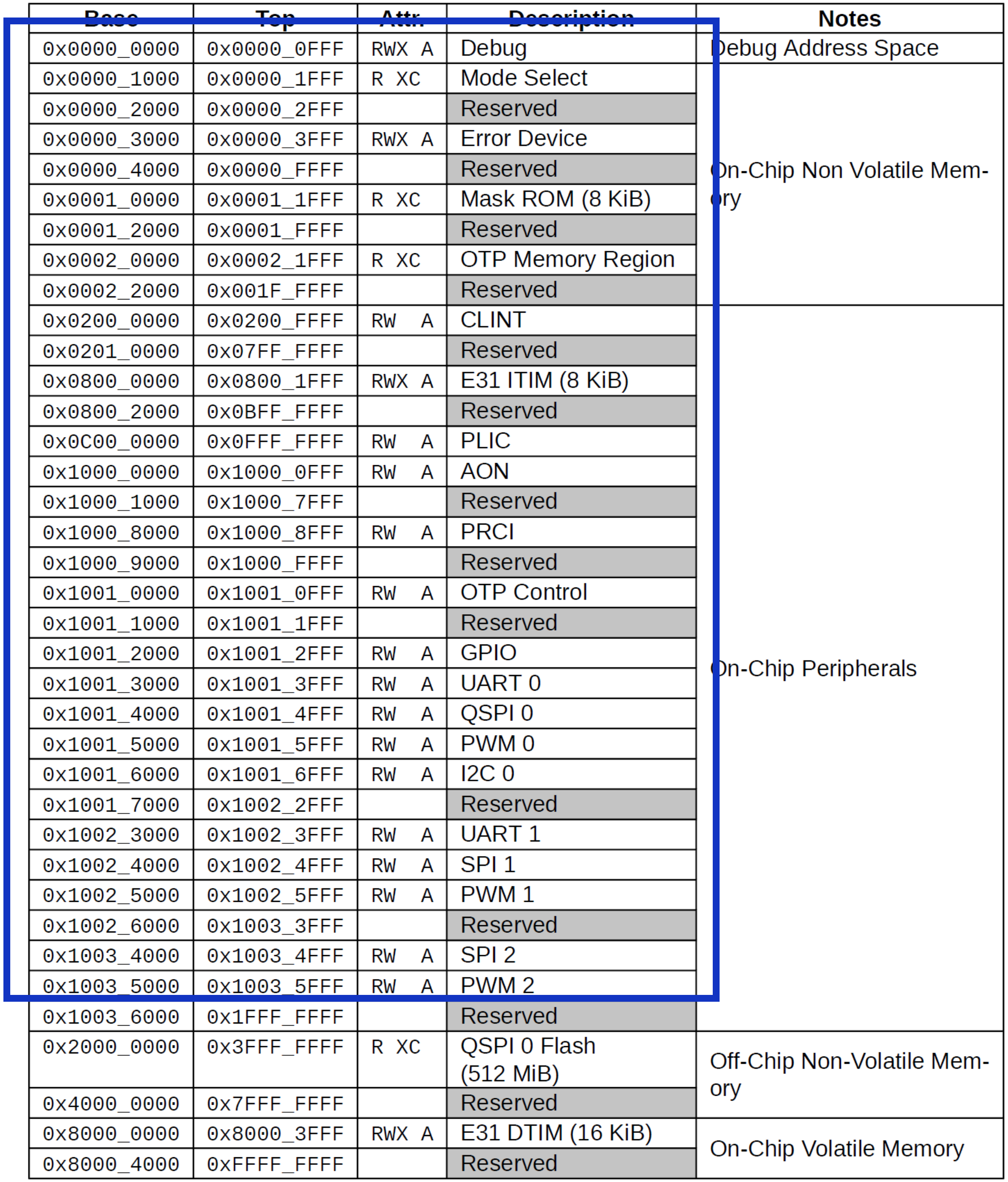
Read 2 bytes from I/O device 0 and store in R5

1-Fetch
2-Decode
3-Evaluate Address
4-Fetch Operands
5-Execute
6-Write Results

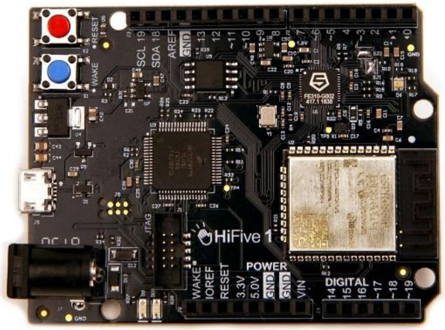
LC-3 Uses Memory Mapped I/O



Memory Map of SiFive FE310



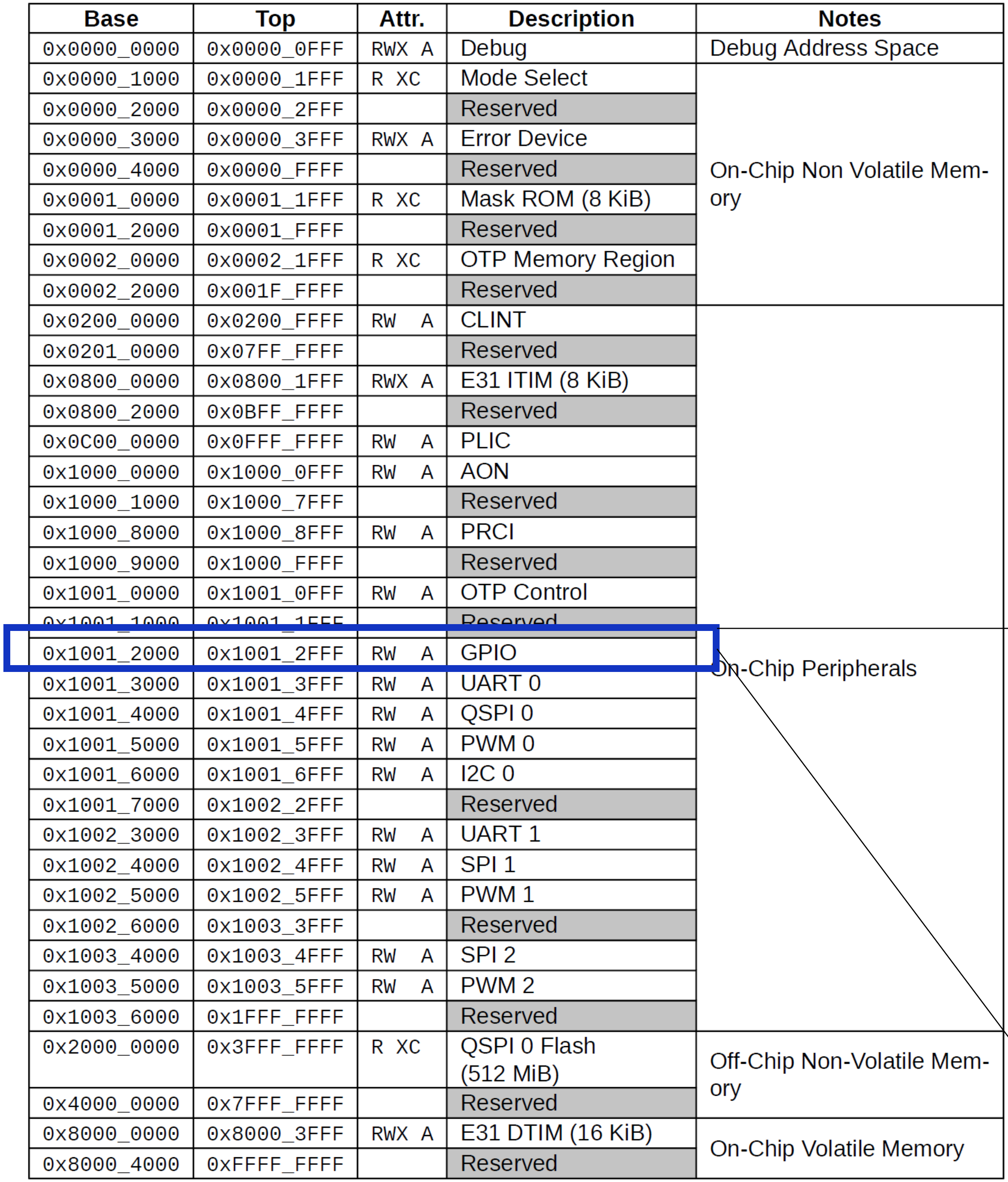
Base	Top	Attr.	Description	Notes
0x0000_0000	0x0000_0FFF	RWX A	Debug	Debug Address Space
0x0000_1000	0x0000_1FFF	R XC	Mode Select	On-Chip Non Volatile Memory
0x0000_2000	0x0000_2FFF		Reserved	
0x0000_3000	0x0000_3FFF	RWX A	Error Device	
0x0000_4000	0x0000_FFFF		Reserved	
0x0001_0000	0x0001_1FFF	R XC	Mask ROM (8 KiB)	
0x0001_2000	0x0001_FFFF		Reserved	
0x0002_0000	0x0002_1FFF	R XC	OTP Memory Region	
0x0002_2000	0x001F_FFFF		Reserved	
0x0200_0000	0x0200_FFFF	RW A	CLINT	On-Chip Peripherals
0x0201_0000	0x07FF_FFFF		Reserved	
0x0800_0000	0x0800_1FFF	RWX A	E31 ITIM (8 KiB)	
0x0800_2000	0x0BFF_FFFF		Reserved	
0x0C00_0000	0x0FFF_FFFF	RW A	PLIC	
0x1000_0000	0x1000_0FFF	RW A	AON	
0x1000_1000	0x1000_7FFF		Reserved	
0x1000_8000	0x1000_8FFF	RW A	PRCI	
0x1000_9000	0x1000_FFFF		Reserved	
0x1001_0000	0x1001_0FFF	RW A	OTP Control	
0x1001_1000	0x1001_1FFF		Reserved	
0x1001_2000	0x1001_2FFF	RW A	GPIO	
0x1001_3000	0x1001_3FFF	RW A	UART 0	
0x1001_4000	0x1001_4FFF	RW A	QSPI 0	
0x1001_5000	0x1001_5FFF	RW A	PWM 0	
0x1001_6000	0x1001_6FFF	RW A	I2C 0	
0x1001_7000	0x1002_2FFF		Reserved	
0x1002_3000	0x1002_3FFF	RW A	UART 1	
0x1002_4000	0x1002_4FFF	RW A	SPI 1	
0x1002_5000	0x1002_5FFF	RW A	PWM 1	
0x1002_6000	0x1003_3FFF		Reserved	
0x1003_4000	0x1003_4FFF	RW A	SPI 2	
0x1003_5000	0x1003_5FFF	RW A	PWM 2	
0x1003_6000	0x1FFF_FFFF		Reserved	
0x2000_0000	0x3FFF_FFFF	R XC	QSPI 0 Flash (512 MiB)	Off-Chip Non-Volatile Memory
0x4000_0000	0x7FFF_FFFF		Reserved	
0x8000_0000	0x8000_3FFF	RWX A	E31 DTIM (16 KiB)	On-Chip Volatile Memory
0x8000_4000	0xFFFF_FFFF		Reserved	



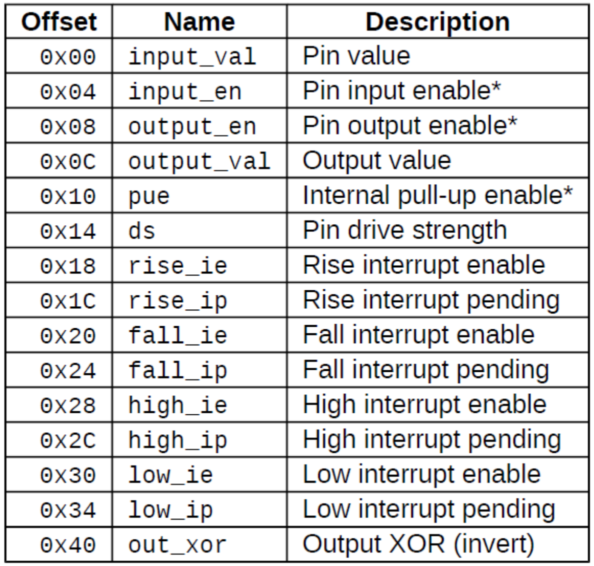
Memory mapped I/O regions

Memory Map of SiFive FE310

GPIO registers are mapped at 0x10012000 – 0x10012FFF



Base	Top	Attr.	Description	Notes
0x0000_0000	0x0000_0FFF	RWX A	Debug	Debug Address Space
0x0000_1000	0x0000_1FFF	R XC	Mode Select	On-Chip Non Volatile Memory
0x0000_2000	0x0000_2FFF		Reserved	
0x0000_3000	0x0000_3FFF	RWX A	Error Device	
0x0000_4000	0x0000_FFFF		Reserved	
0x0001_0000	0x0001_1FFF	R XC	Mask ROM (8 KiB)	
0x0001_2000	0x0001_FFFF		Reserved	
0x0002_0000	0x0002_1FFF	R XC	OTP Memory Region	
0x0002_2000	0x001F_FFFF		Reserved	
0x0200_0000	0x0200_FFFF	RW A	CLINT	On-Chip Peripherals
0x0201_0000	0x07FF_FFFF		Reserved	
0x0800_0000	0x0800_1FFF	RWX A	E31 ITIM (8 KiB)	
0x0800_2000	0x0BFF_FFFF		Reserved	
0x0C00_0000	0x0FFF_FFFF	RW A	PLIC	
0x1000_0000	0x1000_0FFF	RW A	AON	
0x1000_1000	0x1000_7FFF		Reserved	
0x1000_8000	0x1000_8FFF	RW A	PRCI	
0x1000_9000	0x1000_FFFF		Reserved	
0x1001_0000	0x1001_0FFF	RW A	OTP Control	
0x1001_1000	0x1001_1FFF		Reserved	
0x1001_2000	0x1001_2FFF	RW A	GPIO	
0x1001_3000	0x1001_3FFF	RW A	UART 0	
0x1001_4000	0x1001_4FFF	RW A	QSPI 0	
0x1001_5000	0x1001_5FFF	RW A	PWM 0	
0x1001_6000	0x1001_6FFF	RW A	I2C 0	
0x1001_7000	0x1002_2FFF		Reserved	
0x1002_3000	0x1002_3FFF	RW A	UART 1	
0x1002_4000	0x1002_4FFF	RW A	SPI 1	
0x1002_5000	0x1002_5FFF	RW A	PWM 1	
0x1002_6000	0x1003_3FFF		Reserved	
0x1003_4000	0x1003_4FFF	RW A	SPI 2	
0x1003_5000	0x1003_5FFF	RW A	PWM 2	
0x1003_6000	0x1FFF_FFFF		Reserved	
0x2000_0000	0x3FFF_FFFF	R XC	QSPI 0 Flash (512 MiB)	Off-Chip Non-Volatile Memory
0x4000_0000	0x7FFF_FFFF		Reserved	
0x8000_0000	0x8000_3FFF	RWX A	E31 DTIM (16 KiB)	On-Chip Volatile Memory
0x8000_4000	0xFFFF_FFFF		Reserved	



Offset	Name	Description
0x00	input_val	Pin value
0x04	input_en	Pin input enable*
0x08	output_en	Pin output enable*
0x0C	output_val	Output value
0x10	pue	Internal pull-up enable*
0x14	ds	Pin drive strength
0x18	rise_ie	Rise interrupt enable
0x1C	rise_ip	Rise interrupt pending
0x20	fall_ie	Fall interrupt enable
0x24	fall_ip	Fall interrupt pending
0x28	high_ie	High interrupt enable
0x2C	high_ip	High interrupt pending
0x30	low_ie	Low interrupt enable
0x34	low_ip	Low interrupt pending
0x40	out_xor	Output XOR (invert)

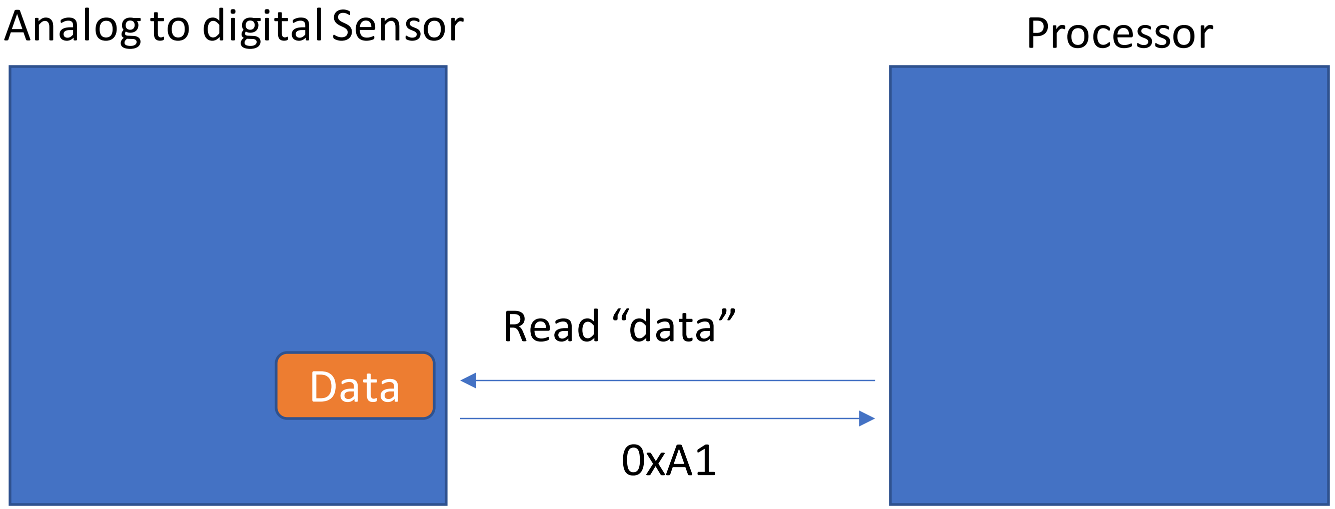
- **Fact**: everything in a computer (including I/O devices) is controlled by the instructions in the ISA
- **Question 1**: Does ISA need special instructions for controlling I/O?
- **Question 2**: Does the I/O device need to work at the same speed of the processing unit?
- **Question 3: Does I/O transfer is initiated by a program or I/O device?**

Does the I/O device need to work at the same speed of the processing unit?

- I/O device is often slower than processor
  - E.g., typist speed vs. clock speed
    - 1GHz clock speed means one clock every 1ns!
- One option: design a processor that accept typed chars at lower speed, e.g., accept a character every 200 million cycles
- **Synchronous vs. Asynchronous I/O**

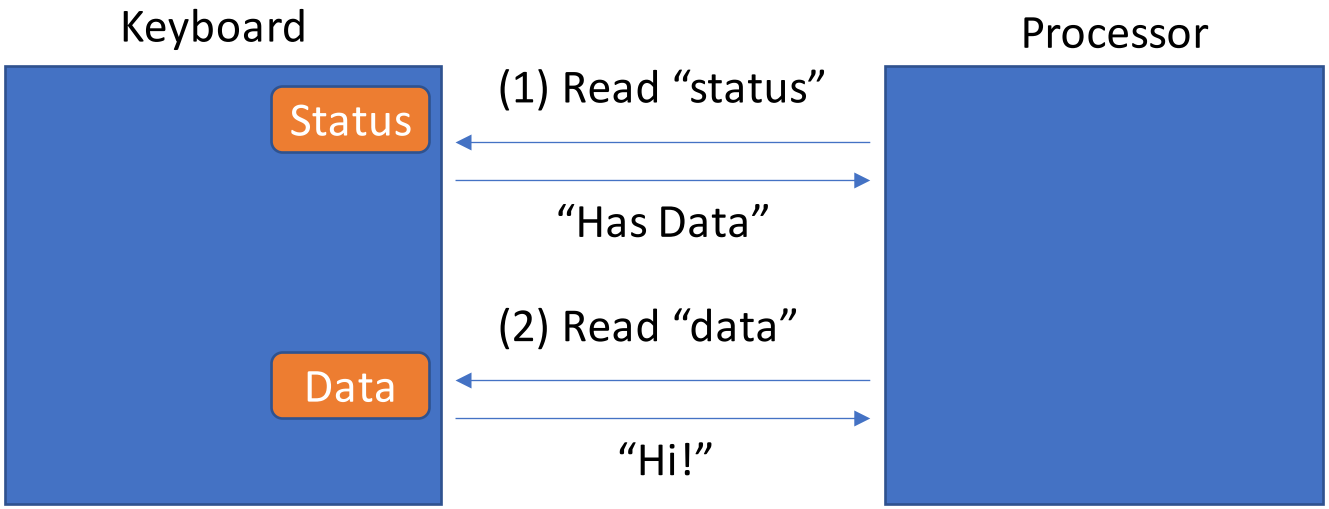
Synchronous vs. Asynchronous I/O

- Synchronous I/O
  - Fixed speed I/O device
  - Read data registers in a fixed interval, e.g., every 1ms



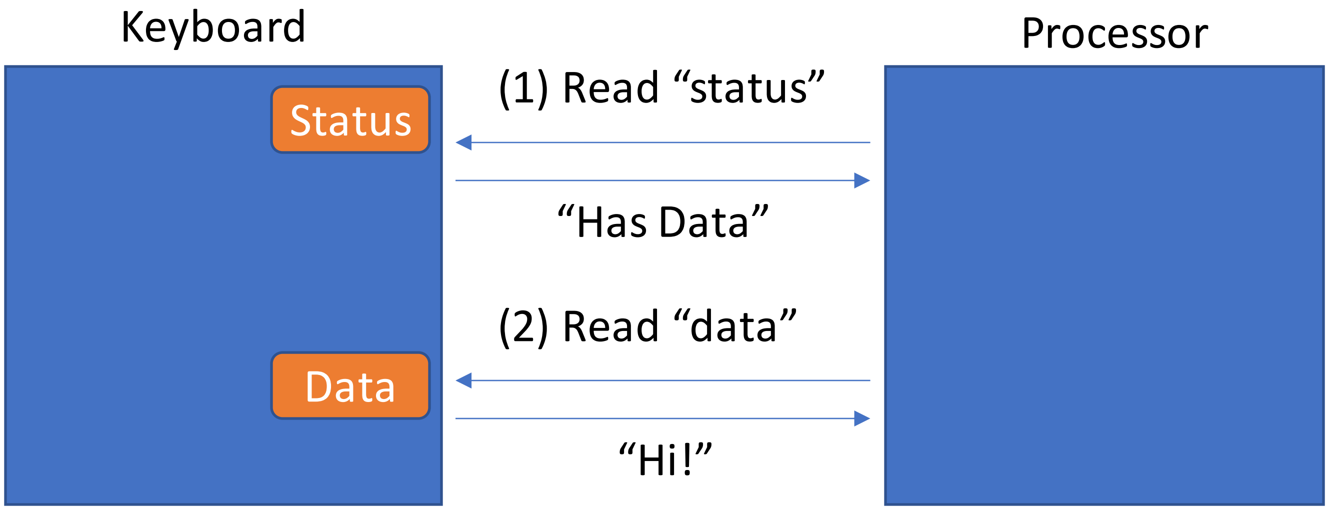
Synchronous vs. Asynchronous I/O

- Asynchronous I/O
  - Speed of I/O device varies
  - Controlled using a *handshaking protocol*



- **Fact**: everything in a computer (including I/O devices) is controlled by the instructions in the ISA
- **Question 1**: Does ISA need special instructions for controlling I/O?
- **Question 2**: Does the I/O device need to work at the same speed of the processing unit?
- **Question 3**: Does I/O transfer is initiated by a program or I/O device?

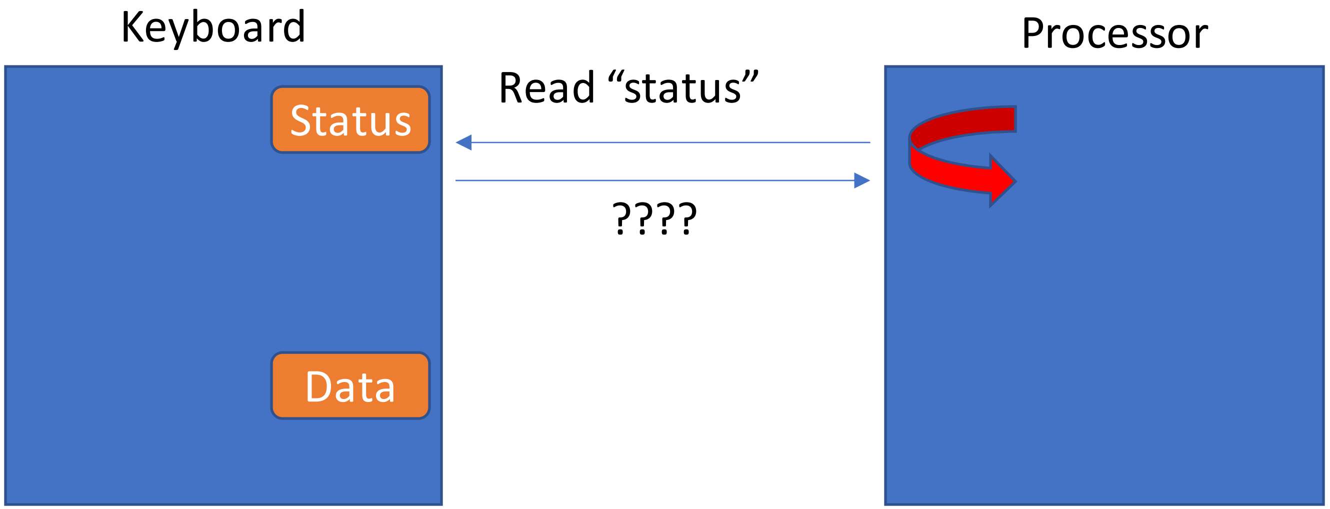
Does processor periodically check keyboard’s “status” OR the keyboard notify the processor when there is a keystroke?



Polling vs. Interrupt Driven I/O

- Polling: processor periodically checks the status register

```
While (status != "Has Data")
    status = *STATUS_MMIO_ADDRESS
```

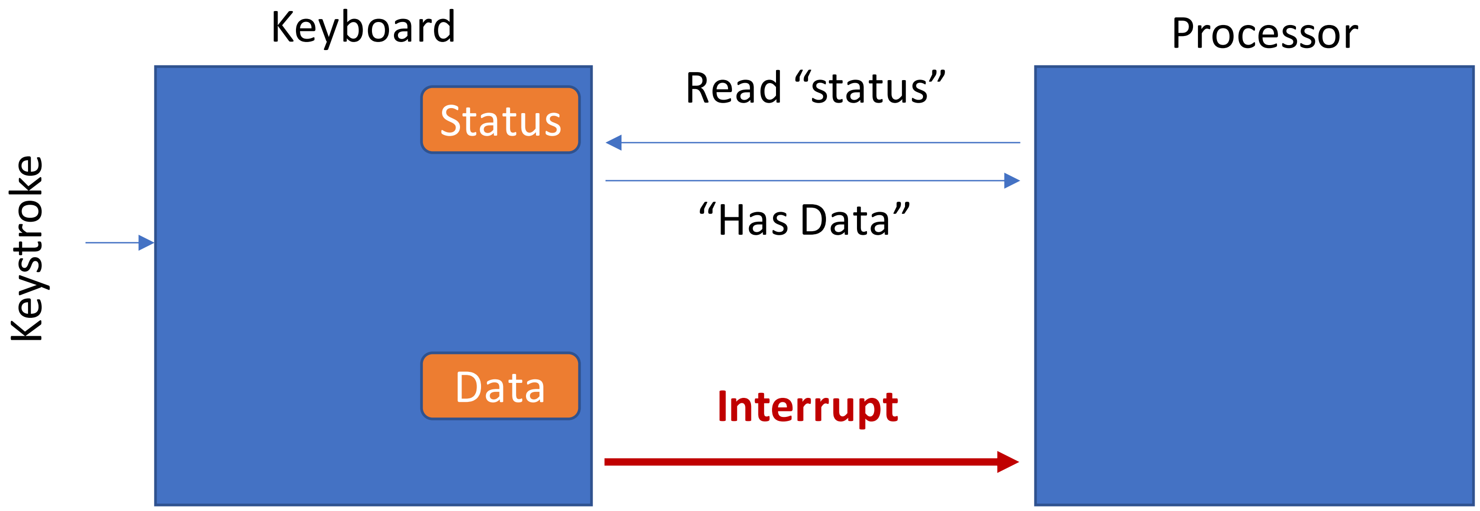


Polling vs. Interrupt Driven I/O

- Polling: processor periodically checks the status register

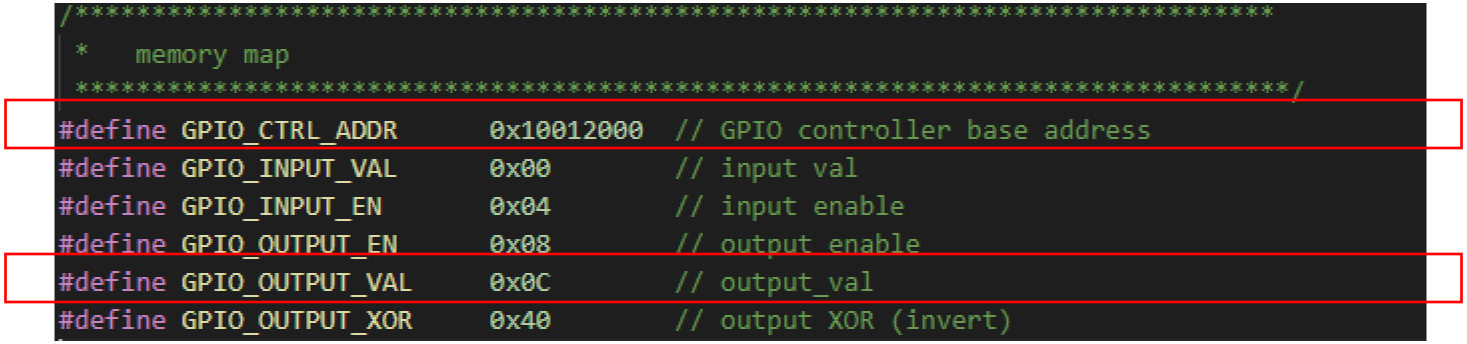
```
While (status != "Has Data")
    status = *STATUS_MMIO_ADDRESS
```

- Interrupt: notification from the I/O dev

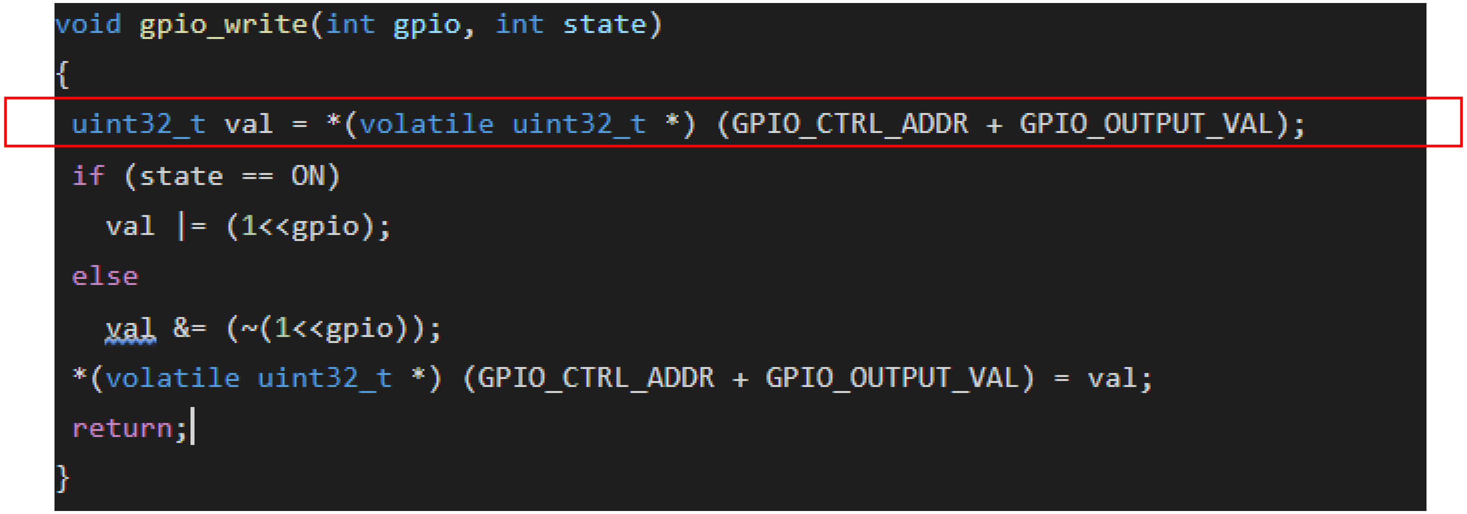


```
status = *STATUS_MMIO_ADDRESS
if status == "Has Data"
   input = *DATA_MMIO_ADDRESS
```

Blinky LED on HiFive Board

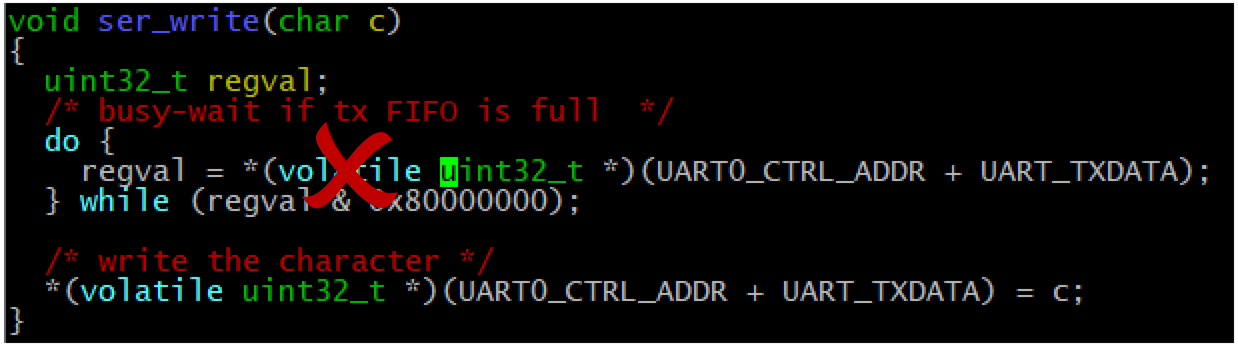


```
/*****************************************************************************
 *   memory map
 *****************************************************************************/
#define GPIO_CTRL_ADDR      0x10012000  // GPIO controller base address
#define GPIO_INPUT_VAL      0x00        // input val
#define GPIO_INPUT_EN       0x04        // input enable
#define GPIO_OUTPUT_EN      0x08        // output enable
#define GPIO_OUTPUT_VAL     0x0C        // output_val
#define GPIO_OUTPUT_XOR     0x40        // output XOR (invert)
```



```
void gpio_write(int gpio, int state)
{
 uint32_t val = *(volatile uint32_t *) (GPIO_CTRL_ADDR + GPIO_OUTPUT_VAL);
 if (state == ON)
   val |= (1<<gpio);
 else
   val &= (~(1<<gpio));
 *(volatile uint32_t *) (GPIO_CTRL_ADDR + GPIO_OUTPUT_VAL) = val;
 return;
}
```

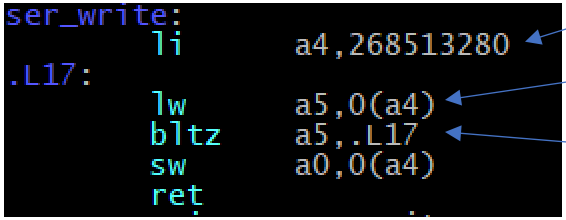
Volatile in C



```
void ser_write(char c)
{
  uint32_t regval;
  /* busy-wait if tx FIFO is full  */
  do {
    regval = *(volatile uint32_t *)(UART0_CTRL_ADDR + UART_TXDATA);
  } while (regval & 0x80000000);

  /* write the character */
  *(volatile uint32_t *)(UART0_CTRL_ADDR + UART_TXDATA) = c;
}
```

volatile uint32_t



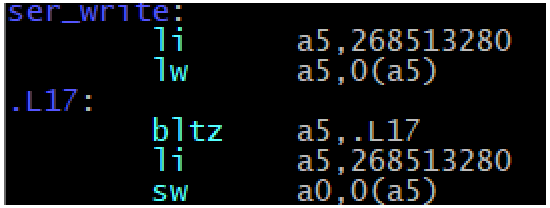
```
ser_write:
        li      a4,268513280
.L17:
        lw      a5,0(a4)
        bltz    a5,.L17
        sw      a0,0(a4)
        ret
```

a4 = 0x10013000

a5 = *a4

Branch to .L17 if a5 < zero

uint32_t



```
ser_write:
        li      a5,268513280
        lw      a5,0(a5)
.L17:
        bltz    a5,.L17
        li      a5,268513280
        sw      a0,0(a5)
```