Computer Organization & Architecture

7-2 Program Controlled I/O

Wang Guohua

School of Software Engineering

Contents

- Communication Methods with CPU
  - Program-controlled I/O (Polling)
  - Interrupt-driven I/O
  - Direct Memory Access (DMA)

Program-Controlled I/O

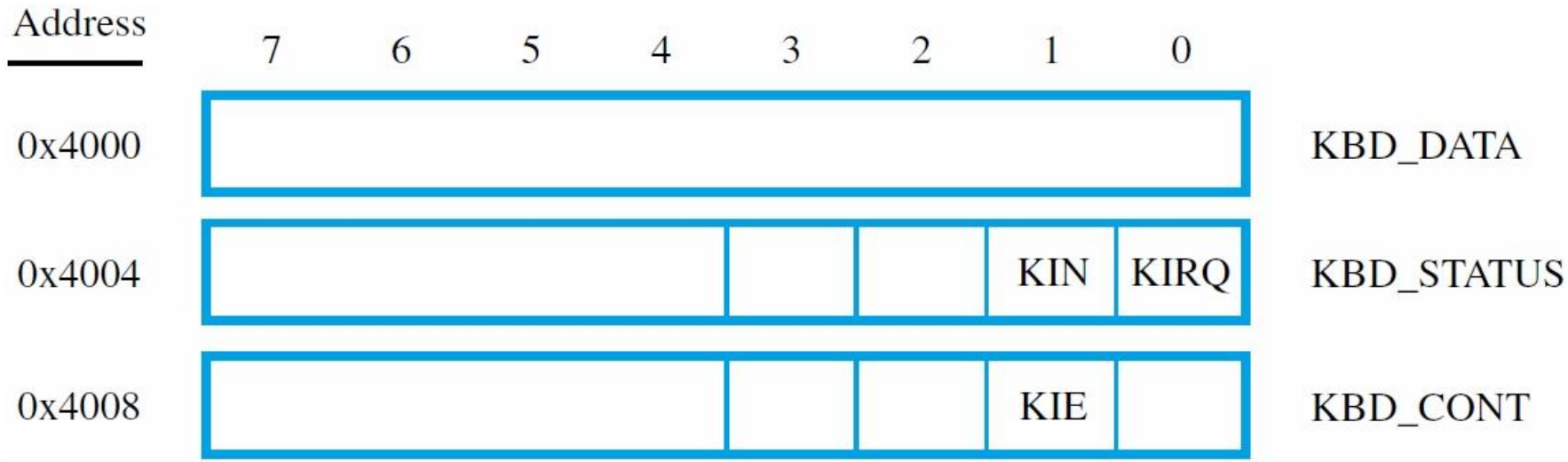
- Discuss I/O issues using keyboard & display.
- Read keyboard characters, store in memory, and display on screen.
- Implement this task with a program that performs all of the relevant functions.
- This approach called program-controlled I/O.
- How can we ensure correct timing of actions and synchronized transfers between devices?

Signaling Protocol for I/O Devices

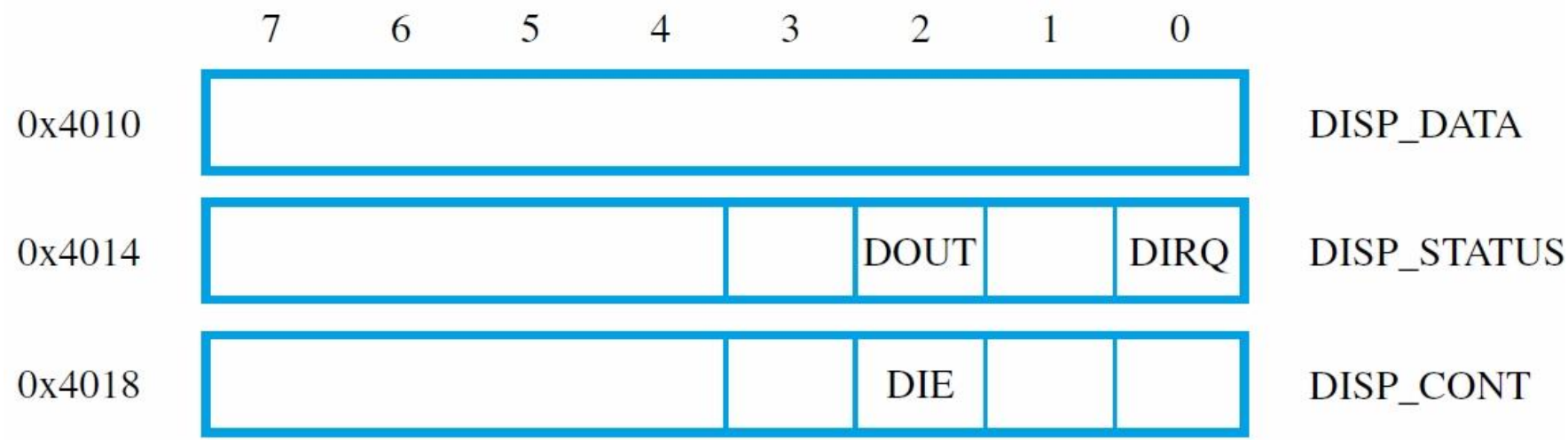
- Assume that the I/O devices have a way to send a "*ready*" signal to the processor.
- For keyboard, indicates character can be read so processor uses Load to access data register.
- For display, indicates character can be sent so processor uses Store to access data register.
- The "*ready*" signal in each case is a status flag in status register that is polled by processor.

Example I/O Registers

- For sample I/O programs that follow, assume specific addresses & bit positions for registers.
- Registers are 8 bits in width and word-aligned.
- For example, keyboard has KIN status flag in bit b_1 of KBD_STATUS register at address 0x4004.
- Processor polls KBD_STATUS register, checking whether KIN flag is 0 or 1.
- If KIN is 1, processor reads KBD_DATA register.



(a) Keyboard interface



(b) Display interface

Wait Loop for Polling I/O Status (1)

- Program-controlled I/O implemented with a wait loop for polling keyboard status register:

```
READWAIT:   LoadByte          R4, KBD_STATUS
            And               R4, R4, #2
            Branch_if_[R4]=0  READWAIT
            LoadByte          R5, KBD_DATA
```

- Keyboard circuit places character in KBD_DATA and sets KIN flag in KBD_STATUS.
- Circuit clears KIN flag when KBD_DATA is read.

Wait Loop for Polling I/O Status (2)

- Similar wait loop for display device:

```
WRITEWAIT :     LoadByte            R4, DISP_STATUS
                And                 R4, R4, #4
                Branch_if_[R4]=0    WRITEWAIT
                StoreByte           R5, DISP_DATA
```

- Display circuit sets DOUT flag in DISP_STATUS after previous character has been displayed.
- Circuit automatically clears DOUT flag when a character is transferred to DISP_DATA.

RISC-style I/O Program (1)

- Consider complete programs that use polling to read, store, and display a line of characters.
- Program finishes when carriage return (CR) character is entered on keyboard.
- Each keyboard character *echoed* to display.
- ***LOC*** is address of first character in stored line.

RISC-style I/O Program (2)

```
        Move            R2, #LOC          Initialize pointer register R2 to point to the
                                            address of the first location in main memory
                                            where the characters are to be stored.
        MoveByte        R3, #CR           Load ASCII code for Carriage Return into R3.
READ:   LoadByte        R4, KBD_STATUS    Wait for a character to be entered.
        And             R4, R4, #2        Check the KIN flag.
        Branch_if_[R4]=0 READ
        LoadByte        R5, KBD_DATA      Read the character from KBD_DATA
                                            (this clears KIN to 0).
        StoreByte       R5, (R2)          Write the character into the main memory and
        Add             R2, R2, #1          increment the pointer to main memory.
ECHO:   LoadByte        R4, DISP_STATUS   Wait for the display to become ready.
        And             R4, R4, #4        Check the DOUT flag.
        Branch_if_[R4]=0 ECHO
        StoreByte       R5, DISP_DATA     Move the character just read to the display
                                            buffer register (this clears DOUT to 0).
        Branch_if_[R5]≠[R3] READ          Check if the character just read is the
                                            Carriage Return. If it is not, then
                                            branch back and read another character.
```