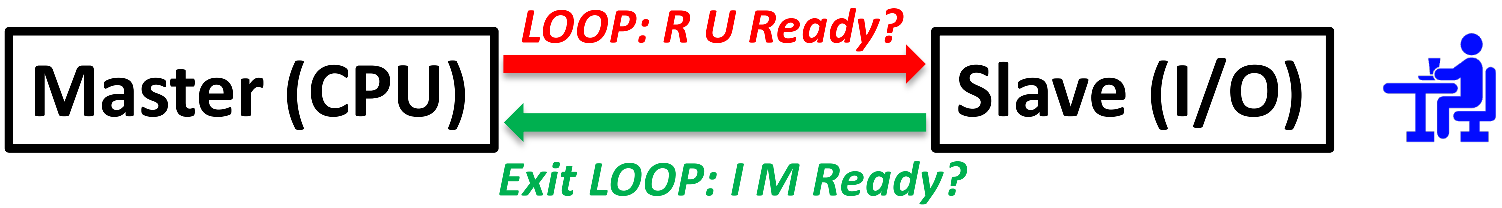
Ch.3 Input and Output

Topics

- Introduction: input and output (I/O)
  - Gaming industry devices
- 3.1.1: I/O device – system interface
- 3.1.2: Program-controlled I/O (Polling)
- 3.2: Interrupt-based I/O
- 3.2.6: Exceptions

2 ways to interact with I/O

3.1.2 Program-Controlled I/O (polling)



LOOP: R U Ready?

Master (CPU)

Slave (I/O)

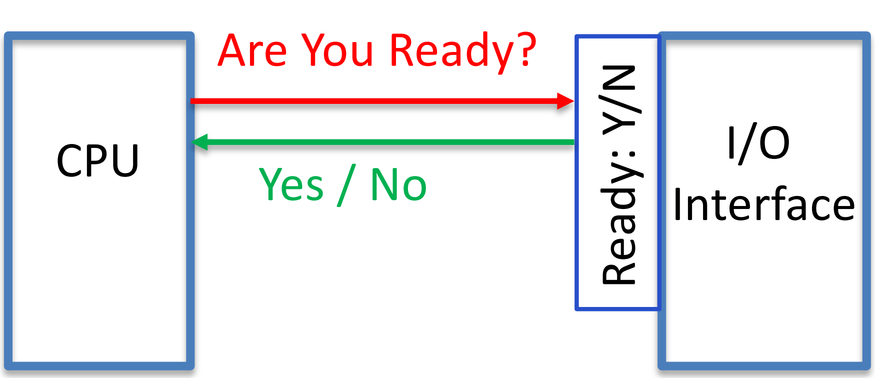
Exit LOOP: I M Ready?

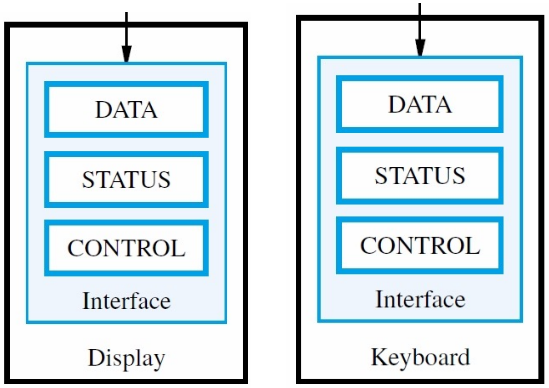
1. CPU checks I/O readiness in regular interval (continuously or every ***X*** msec)
2. A key is pressed on the keyboard
3. I/O (interface) indicates its readiness
4. When the CPU sees this readiness:
   - a. Inputs keyboard character
     - i. Stores readable character in memory

       or

     - ii. Reacts to control function (e.g., ESC, ctrl Q, etc.)
   - b. Displays character on screen (i.e., 'Keyboard Echo' by the host system)

Signaling Protocol for I/O



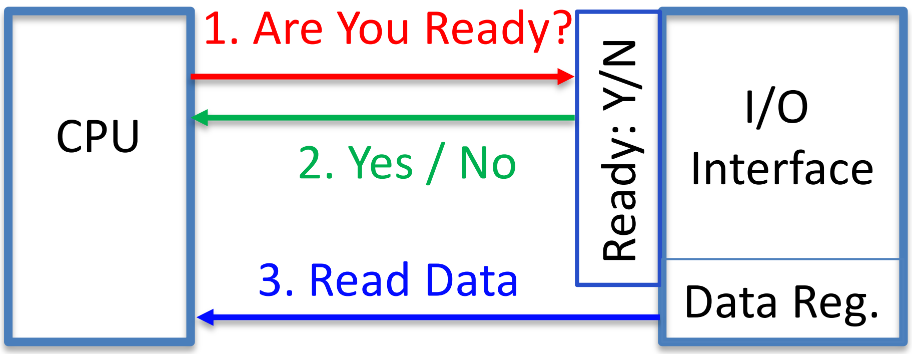


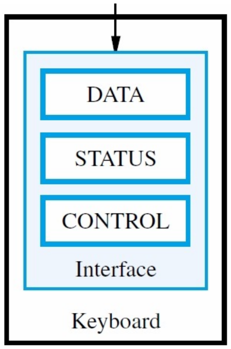
- CPU polls the I/O device '**Are You Ready**?'
- I/O indicates '**Ready**' status: yes or no
- '**Ready**' is a
  - A **flag (bit)** in the **I/O status register**

What are Yes and No ?

1 and 0; On and Off; +5v and 0v

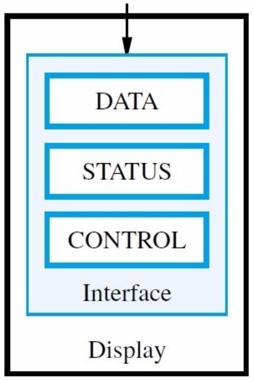
Signaling Protocol: Read/Write

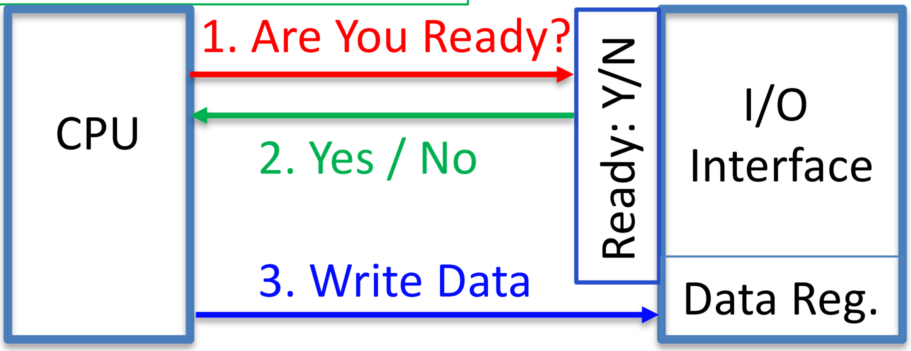




INPUT Device:

Ready: New data in Data Reg.; OK to read/load

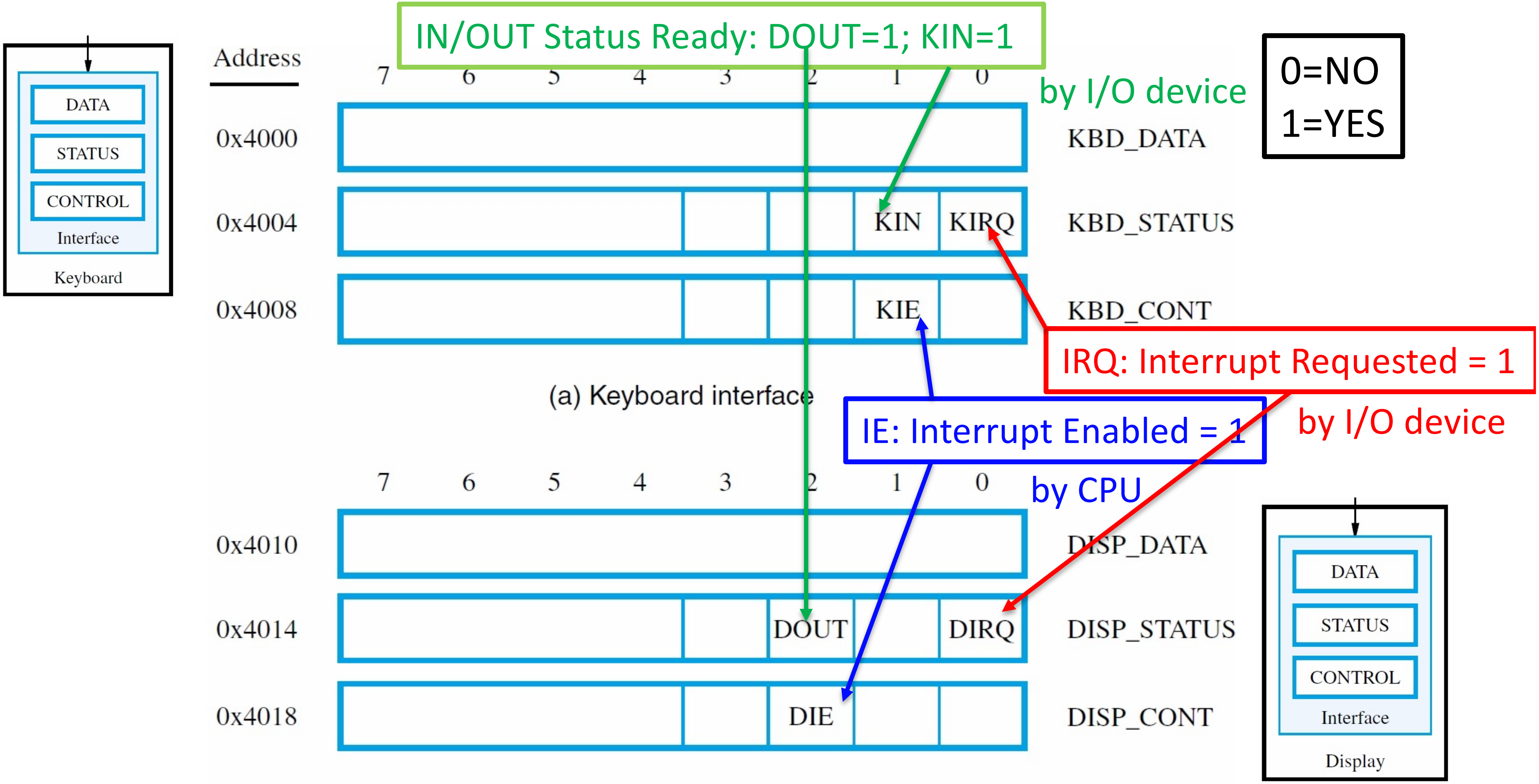




OUTPUT Device:

Ready: Data Reg. available; OK to write/store

Fig. 3.3: Example I/O Registers Layout



IN/OUT Status Ready: DOUT=1; KIN=1

by I/O device

0=NO
1=YES

IRQ: Interrupt Requested = 1

by I/O device

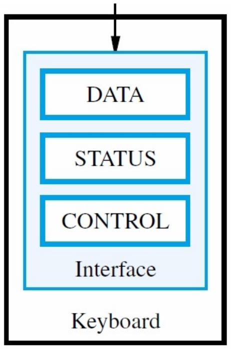
IE: Interrupt Enabled = 1

by CPU

(a) Keyboard interface

(b) Display interface

Keyboard and Keyboard-I/O Register Interaction


DATA

STATUS

CONTROL

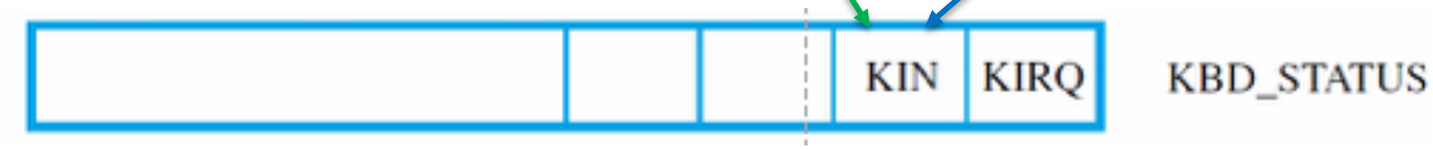
Interface

Keyboard

- When a key is pressed, the **keyboard** circuit:
  1) Places the character in KBD_DATA register
  2) Then sets KIN flag (0→1) in KBD_STATUS register; indicating data is ready for input to CPU
  3) Waits for CPU to read character

  ………. time passes

  4) CPU reads character
  5) When KBD_DATA is read, clears KIN flag (1 → 0); indicating no char to input; wait for typing input


KIN

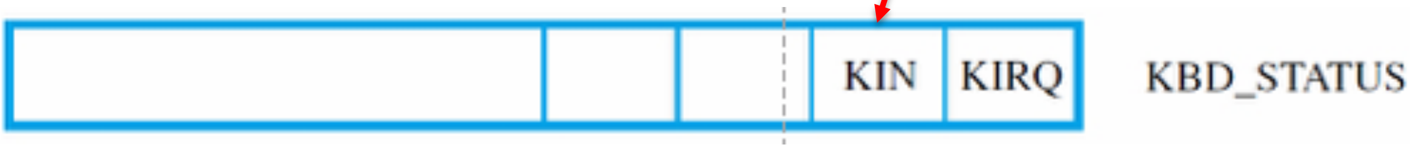
KIRQ

KBD_STATUS

Processor and Keyboard-I/O Register Interaction

- **Processor**:
  a) Polls (reads) KBD_STATUS register
  b) Checks whether KIN flag is 0 or 1
  c) If KIN is 0, not ready, poll again, go to 1
  d) else KIN = 1, reads KBD_DATA register
  e) Continue processing

  ……….



Processor and Keyboard Handshaking

- **Processor**:
  - a) Polls (reads) KBD_STATUS register
  - b) Checks whether KIN flag is 0 or 1
  - c) If KIN is 0, not ready, poll again, go to 1

  ……….

Handshaking Sequence:

a, b, c;

1, 2, 3;

d, e;

4, 5;

Processor and Keyboard Handshaking

- **Processor**:
  a) Polls (reads) KBD_STATUS register
  b) Checks whether KIN flag is 0 or 1
  c) If KIN is 0, not ready, poll again, go to 1
  ……….

Handshaking Sequence:
a, b, c;
1, 2, 3;
d, e;
4, 5;

- When a key is pressed, the **keyboard** circuit:
  1) Places the character in KBD_DATA register
  2) Then sets KIN flag (0→1) in KBD_STATUS register
  3) Waits for CPU to read character
  ………. time passes

Processor and Keyboard Handshaking

- **Processor**:
  a) Polls (reads) KBD_STATUS register
  b) Checks whether KIN flag is 0 or 1
  c) If KIN is 0, not ready, poll again, go to 1
  d) else KIN = 1, reads KBD_DATA register
  e) Continue processing
  ……….

Handshaking Sequence:
a, b, c;
1, 2, 3;
d, e;
4, 5;

- When a key is pressed, the **keyboard** circuit:
  1) Places the character in KBD_DATA register
  2) Then sets KIN flag (0→1) in KBD_STATUS register
  3) Waits for CPU to read character
  ………. time passes

Processor and Keyboard Handshaking

- **Processor**:
  a) Polls (reads) KBD_STATUS register
  b) Checks whether KIN flag is 0 or 1
  c) If KIN is 0, not ready, poll again, go to 1
  d) else KIN = 1, reads KBD_DATA register
  e) Continue processing

  ……….

Handshaking Sequence:
a, b, c;
1, 2, 3;
d, e;
4, 5;

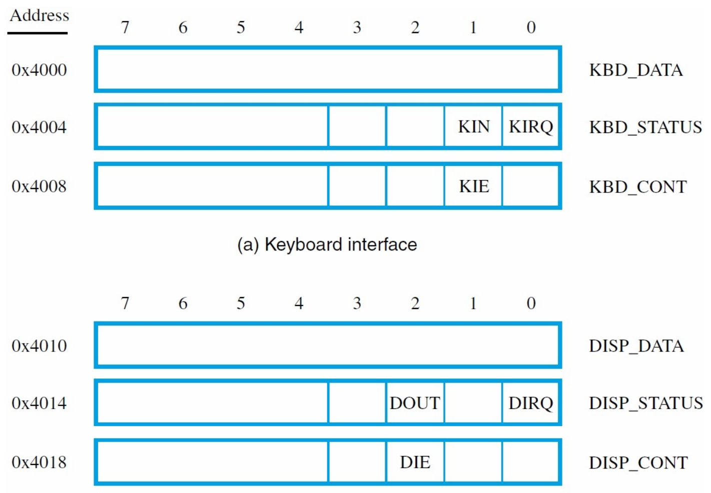
- When a key is pressed, the **keyboard** circuit:
  1) Places the character in KBD_DATA register
  2) Then sets KIN flag (0→1) in KBD_STATUS register
  3) Waits for CPU to read character

  ………. time passes

  4) CPU reads character
  5) When KBD_DATA is read, clears KIN flag (1 → 0)

Polling Keyboard Input Status Continuously

Pseudo Code: polling using a **wait loop**



Address	7	6	5	4	3	2	1	0	
0x4000									KBD_DATA
0x4004							KIN	KIRQ	KBD_STATUS
0x4008							KIE		KBD_CONT

(a) Keyboard interface

	7	6	5	4	3	2	1	0	
0x4010									DISP_DATA
0x4014						DOUT		DIRQ	DISP_STATUS
0x4018						DIE			DISP_CONT

(b) Display interface

Loading Byte ? Not Word?

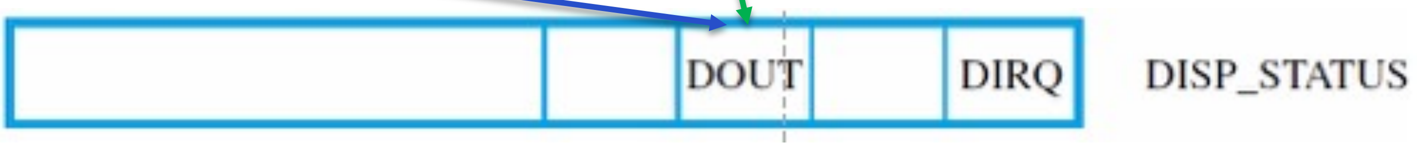
```
READWAIT:   LoadByte           R4, KBD_STATUS; 0x4004
            And                R4, R4, #2; 00000010 and XXXX XX?X
            Branch_if_[R4]=0   READWAIT; branch if ? = 0
            LoadByte           R5, KBD_DATA; 0x4000
```

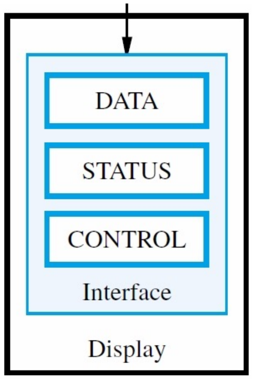
Display and Display-I/O Register Interaction

- **Display** circuit:
  1) Displays a character on screen
  2) Then sets DOUT flag (0→1) in DISP_STATUS; indicating ready for next char
  3) Waits for CPU to send character

  ……….time passes

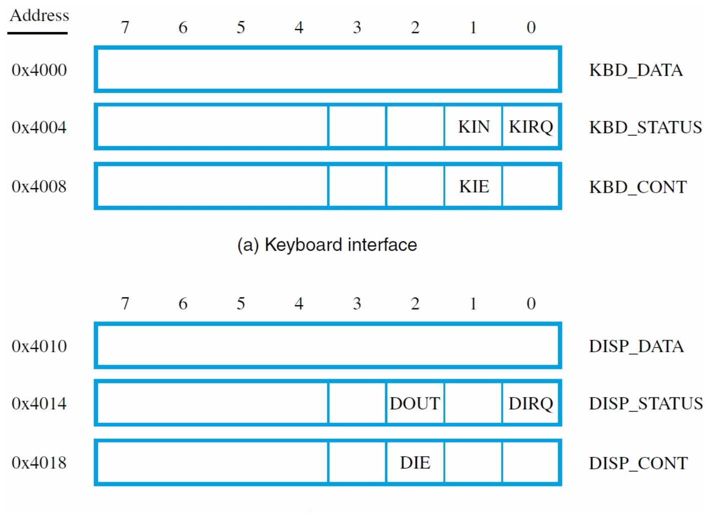
  4) CPU sends character
  5) When DISP_DATA register is written into, clears DOUT flag (1→0); indicating busy and not ready for next char





Polling Display Output Status Continuously

Pseudo Code: polling using a **wait loop**:

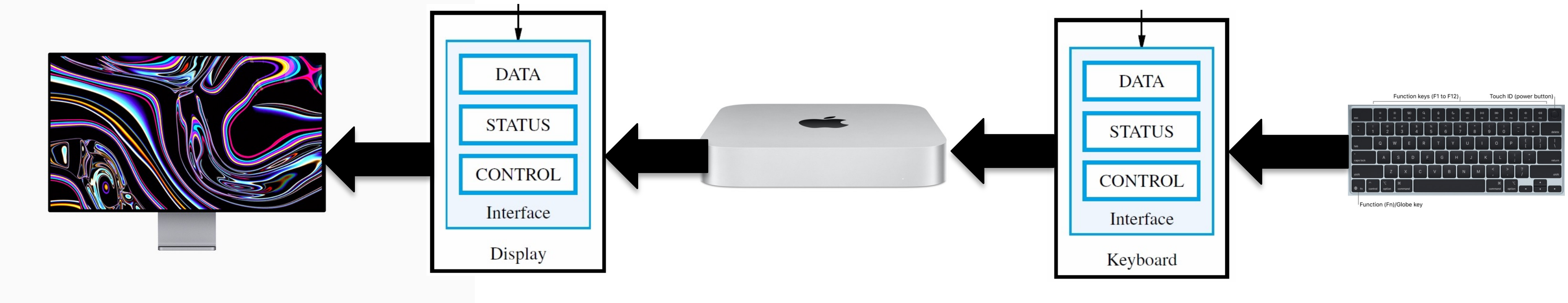


(a) Keyboard interface

(b) Display interface

```
WRITEWAIT:    LoadByte            R4, DISP_STATUS; 0x4014
              And                 R4, R4, #4; 00000100 and XXXX X?XX
              Branch_if_[R4]=0    WRITEWAIT
              StoreByte           R5, DISP_DATA; 0x4018
```

Keyboard CPU Display Layout



DATA

STATUS

CONTROL

Interface

Display

DATA

STATUS

CONTROL

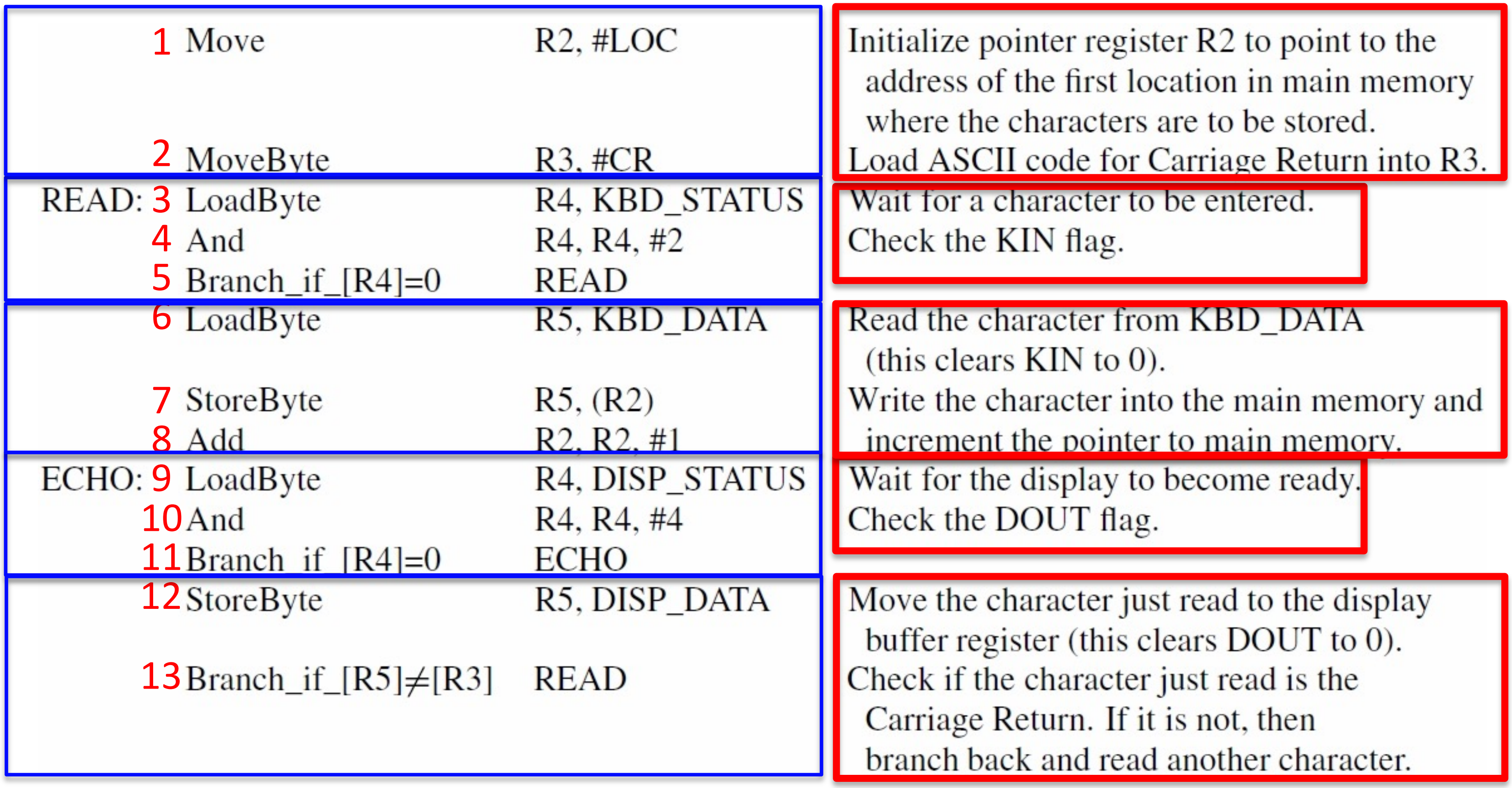
Interface

Keyboard

3.1.3 Keyboard-CPU-Display Interaction

- Uses polling to read, store, and display a line of characters
- Echoes each keyed-in character to Display
- Exits Program when carriage return (CR) is entered and detected

Fig. 3.4 Sample Program



```
        1  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.
        2  MoveByte              R3, #CR           Load ASCII code for Carriage Return into R3.
READ:   3  LoadByte              R4, KBD_STATUS    Wait for a character to be entered.
        4  And                   R4, R4, #2        Check the KIN flag.
        5  Branch_if_[R4]=0      READ
        6  LoadByte              R5, KBD_DATA      Read the character from KBD_DATA
                                                     (this clears KIN to 0).
        7  StoreByte             R5, (R2)          Write the character into the main memory and
        8  Add                   R2, R2, #1          increment the pointer to main memory.
ECHO:   9  LoadByte              R4, DISP_STATUS   Wait for the display to become ready.
       10  And                   R4, R4, #4        Check the DOUT flag.
       11  Branch_if_[R4]=0      ECHO
       12  StoreByte             R5, DISP_DATA     Move the character just read to the display
                                                     buffer register (this clears DOUT to 0).
       13  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.
```

Issues with Polling

- Wait loop is simple (Software)
- Under program control

BUT

- Processor checks status at regular interval
  - Processor is not able to do anything else
- When will I/O device becomes ready?
  - **Non-deterministic delay**
- Not using resources efficiently
- Solution: De-centralized
  - Let I/O device alerts the processor
  - By an **interrupt-request signal** (Hardware)