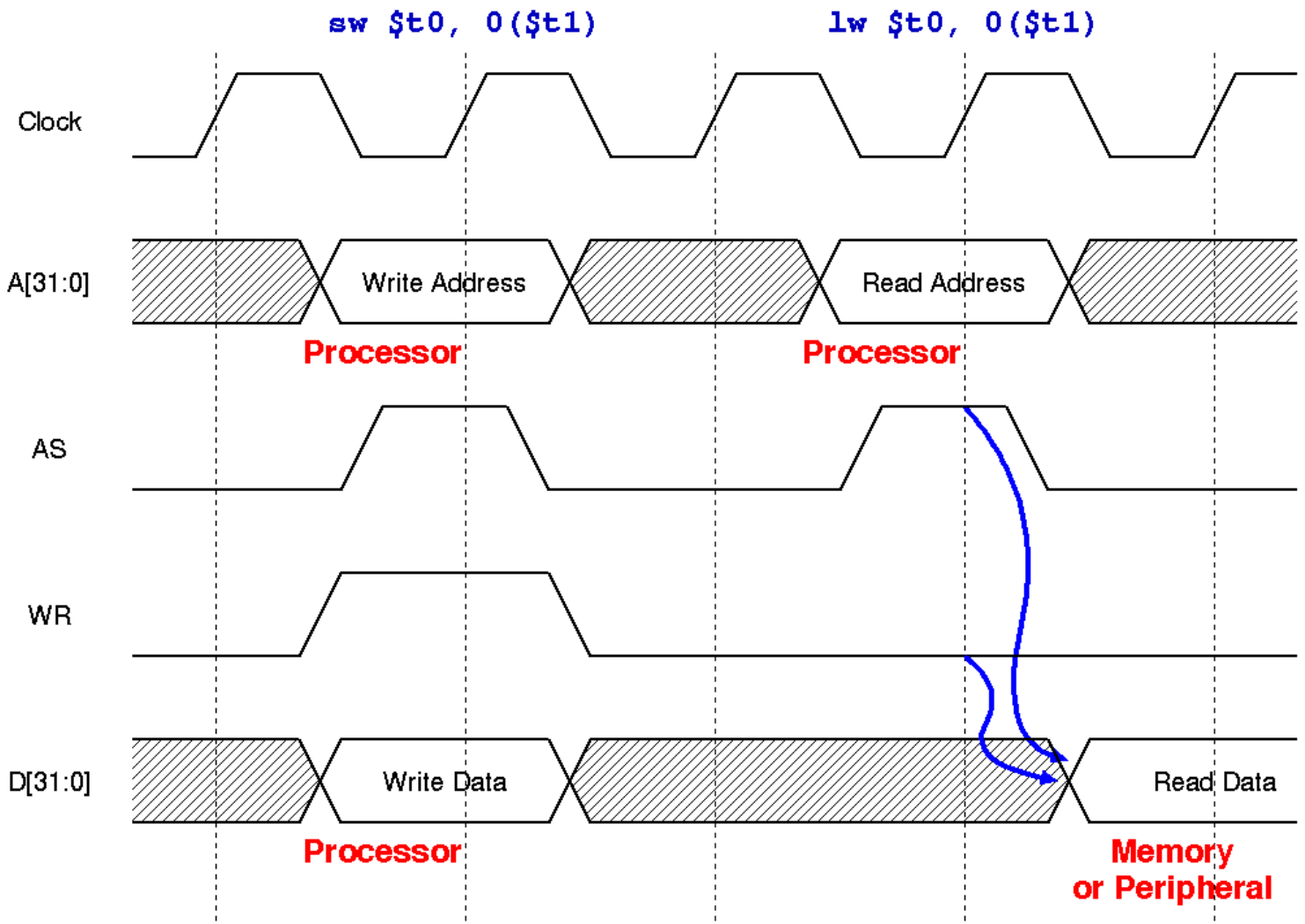
CS-200
Computer Architecture
—
Part 2e. Processor, I/Os, and Exceptions
An Example of I/Os and Exceptions

Paolo Ienne
<paolo.ienne@epfl.ch>

Part Ia: Connecting an Input Peripheral

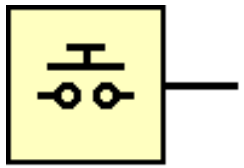
- Consider an hypothetical processor with the following buses and control signals
  - **A[31:0]** → Address bus
  - **D[31:0]** → Data bus
  - **AS** → Address Strobe (active when a valid address is present on A[31:0])
  - **WR** → Write (active with AS when performing a write cycle)

Bus Protocol

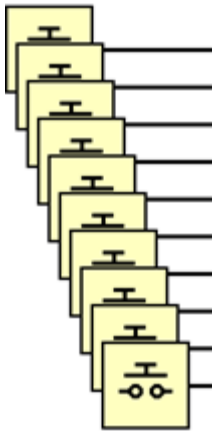


Part Ia: Connecting an Input Peripheral

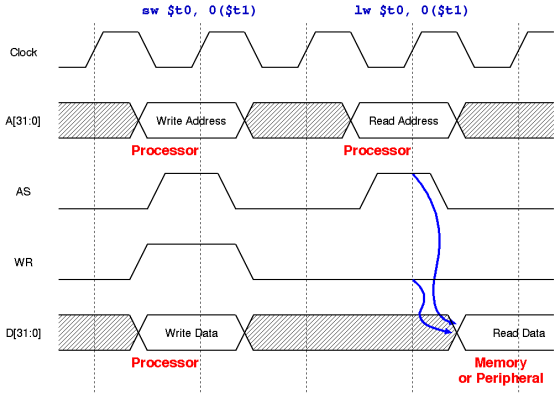
- Connect to the processor **10 buttons numbered from 0 to 9**
- Each button outputs a logic ‘1’ if pressed, ‘0’ otherwise

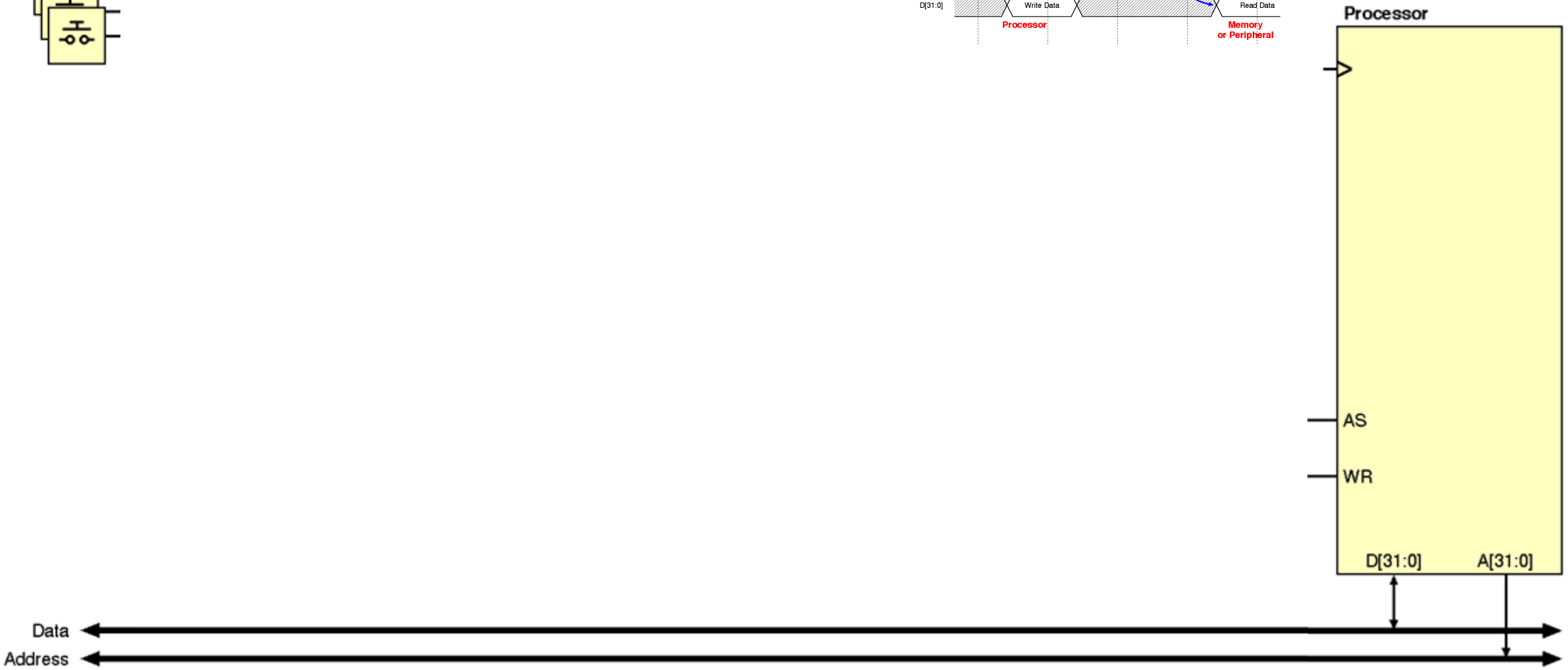


- The processor must read the **state of the buttons** with a read from memory location `0xFFFF’FFF0`: ‘0’ indicates no button pressed, ‘1’ indicates a button pressed
- The processor must read the **number of the button pressed** with a read from memory location `0xFFFF’FFF4`

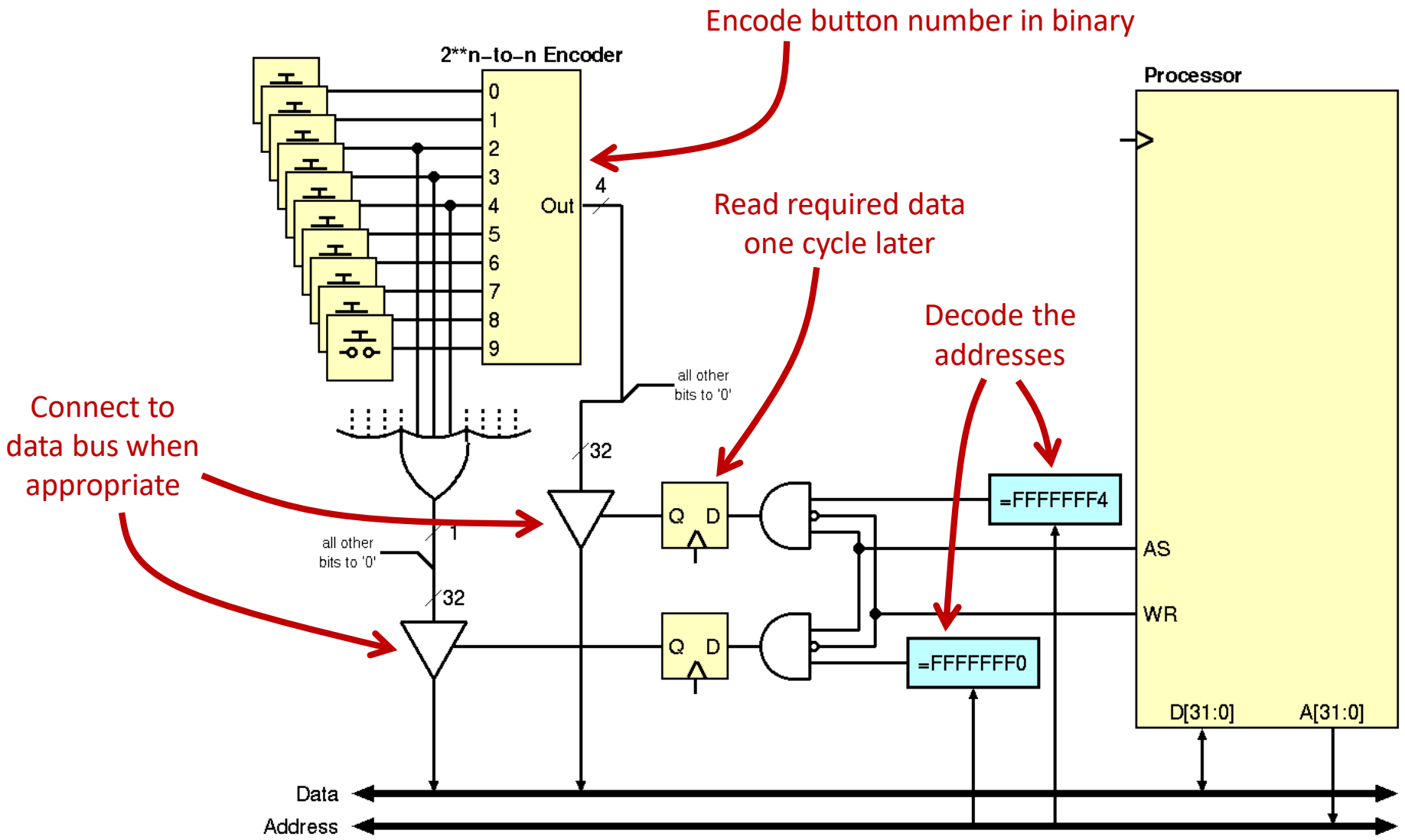


Circuit

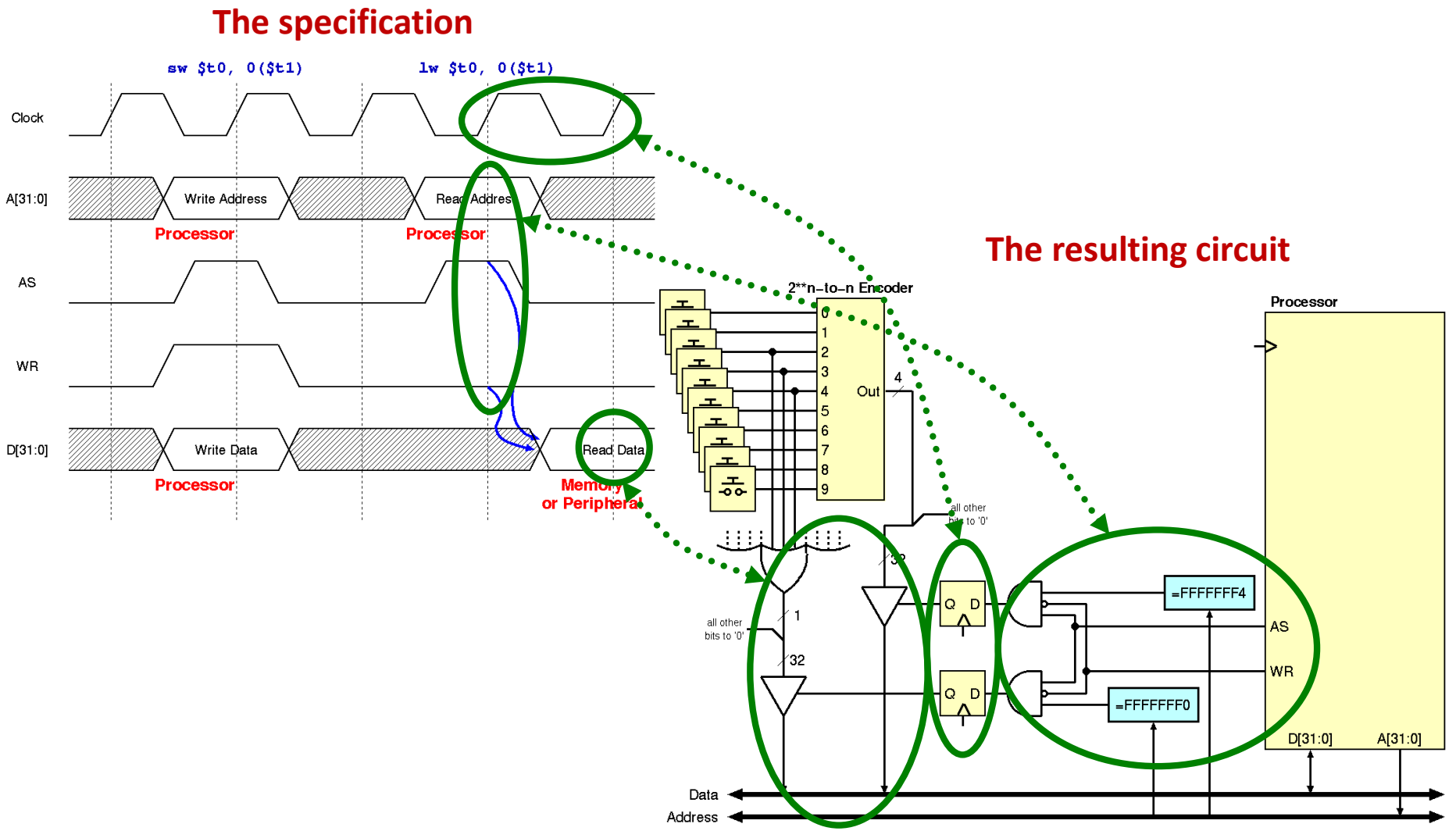

sw $t0, 0($t1)
lw $t0, 0($t1)
Clock
A[31:0]
Write Address
Read Address
Processor
Processor
AS
WR
D[31:0]
Write Data
Read Data
Processor
Memory
or Peripheral


Processor
AS
WR
D[31:0]
A[31:0]
Data
Address

Part Ia: Solution



Part Ia: Solution (Circuit ⇔ Timing)



Part Ib: Reading the Input Ports

- Write a program in RISC-V program **`buttons`** to **poll the state of the buttons**
- **Every time a button is pressed**, call the function **`ShowIt`** with **`a0`** containing the ASCII code of the character corresponding to the button pressed ("0" → 48, "1" → 49, "2" → 50…)
- You don't need to write the function **`showIt`**

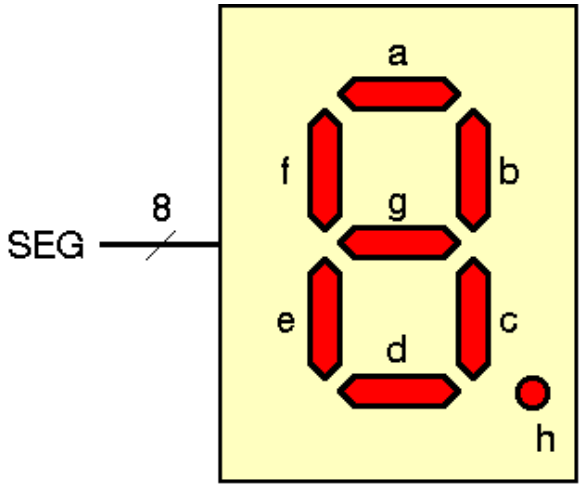
Software: buttons

Part Ib: Solution: buttons

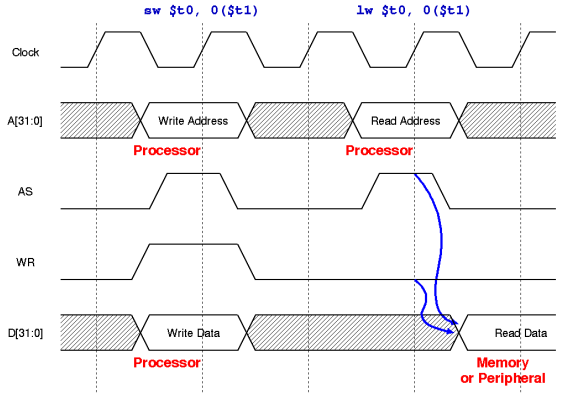
```
buttons:    li      s0, 0xfffffff0      # s0 = 0xffff'fff0

poll:       lw      a0, 0(s0)           # a0 = state (0xffff'fff0)
            beqz    a0, poll            # wait until button pressed

            lw      a0, 4(t0)           # a0 = button (0xffff'fff4)
            addi    a0, a0, 48          # a0 = ASCII of button

            jal     showIt              # Display ASCII digit

            j       poll
```

Part IIa: Connecting an Output Peripheral

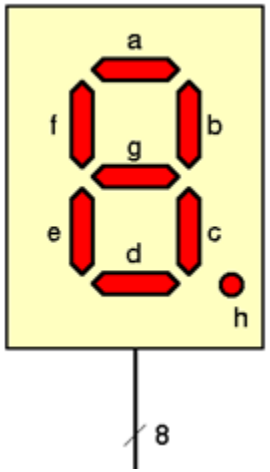
- Connect a **seven-segment display** to the processor



- The input to the peripheral is a **eight-bit signal SEG**: bit 0 is segment *a* (‘1’ means lit), bit 1 is segment *b*, etc.
- The processor must write a **digit to the display** with a write to location `0xFFFF’FFF8`

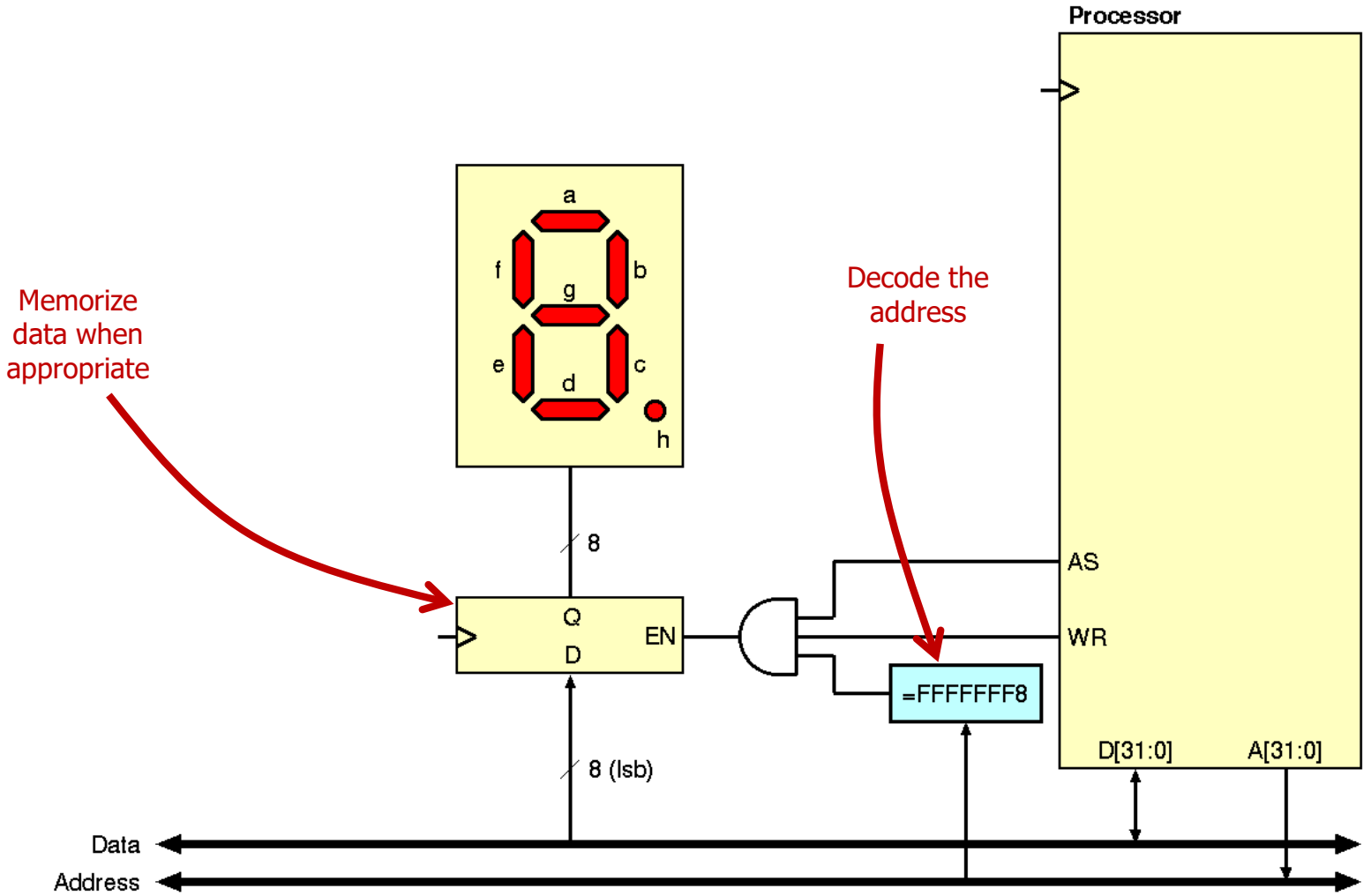

sw $t0, 0($t1)
lw $t0, 0($t1)
Clock
A[31:0]
Write Address
Read Address
Processor
Processor
AS
WR
D[31:0]
Write Data
Read Data
Processor
Memory
or Peripheral

Circuit


a
f
b
g
e
c
d
h
8

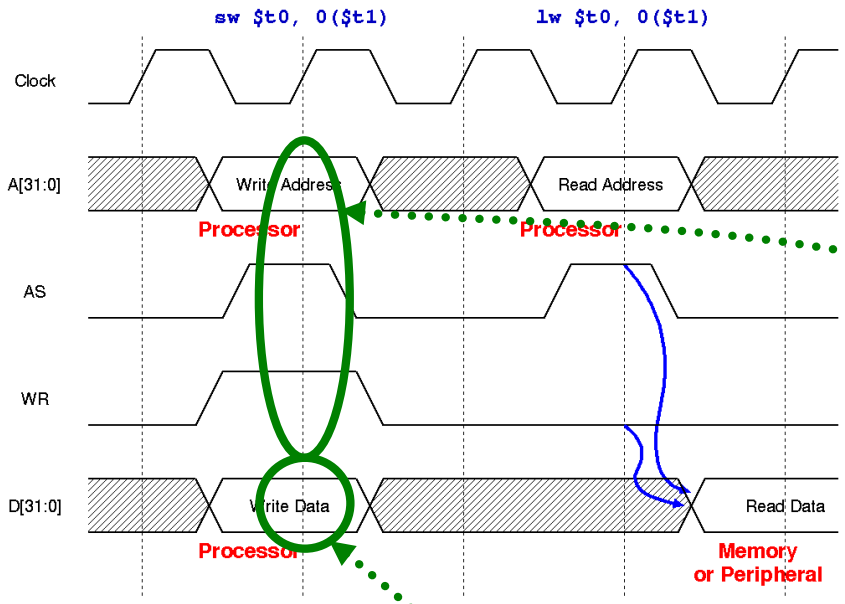
Processor
AS
WR
D[31:0]
A[31:0]
Data
Address

Part IIa: Solution

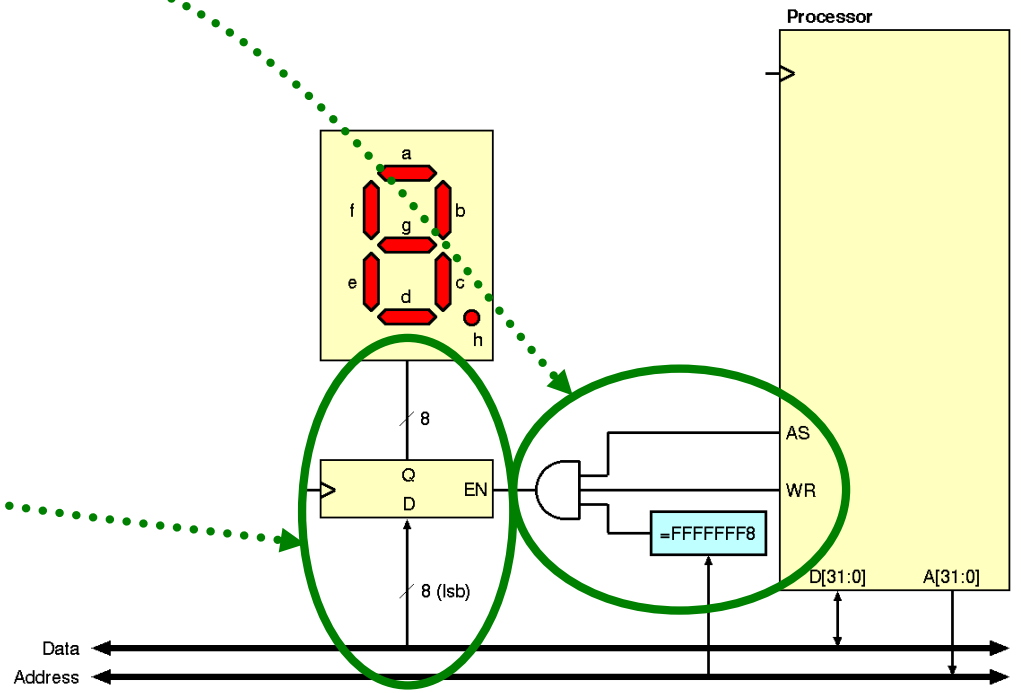


Part IIa: Solution (Circuit ⇔ Timing)

The specification



The resulting circuit



Part IIb: Writing to the Output Port

- Write the **`showIt`** function to **display the character** passed through **`a0`** on the seven-segment display
- If the character in **`a0`** is not a numeric (“0” to “9”), display “-”

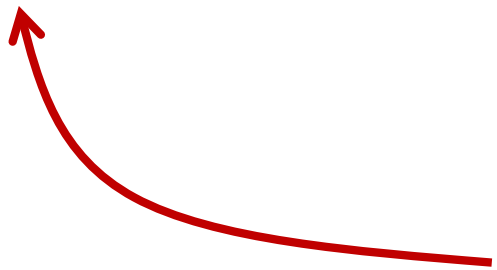
Software: `showIt`

Part Ib: Solution: showIt

```
showIt:     li    t0, 0xfffffff0      # t0 = 0xffff'fff0

poll:       li    t1, 48              # ASCII for '0'
            blt   a0, t1, error       # ASCII < '0' is not a digit
            li    t1, 57              # ASCII for '9'
            bgt   a0, t1, error       # ASCII > '9' is not a digit

readTable:  la    t1, digitTable
            subi  a0, a0, 48          # Button number = index in table
            add   a0, a0, t1          # a0 = digitTable + index
            lbu   a0, 0(a0)           # a0 = digitTable[index] = segments

            sw    a0, 8(t0)           # Display (0xffff'fff4) = a0 (could be "sb")

            ret

error:      li    a0, 10              # a0 = 10 for "-" in digitTable
            j     readTable
```

Part Ib: Solution: `showIt`

```
digitTable:   .byte   0b00111111          # segments for “0”
              .byte   0b00000110          # segments for “1”
              .byte   0b01011011          # segments for “2”
              .byte   0b01001111          # segments for “3”
              .byte   0b01100110          # segments for “4”
              .byte   0b01101101          # segments for “5”
              .byte   0b01111101          # segments for “6”
              .byte   0b00000111          # segments for “7”
              .byte   0b01111111          # segments for “8”
              .byte   0b01101111          # segments for “9”
              .byte   0b01000000          # segments for “-”
```

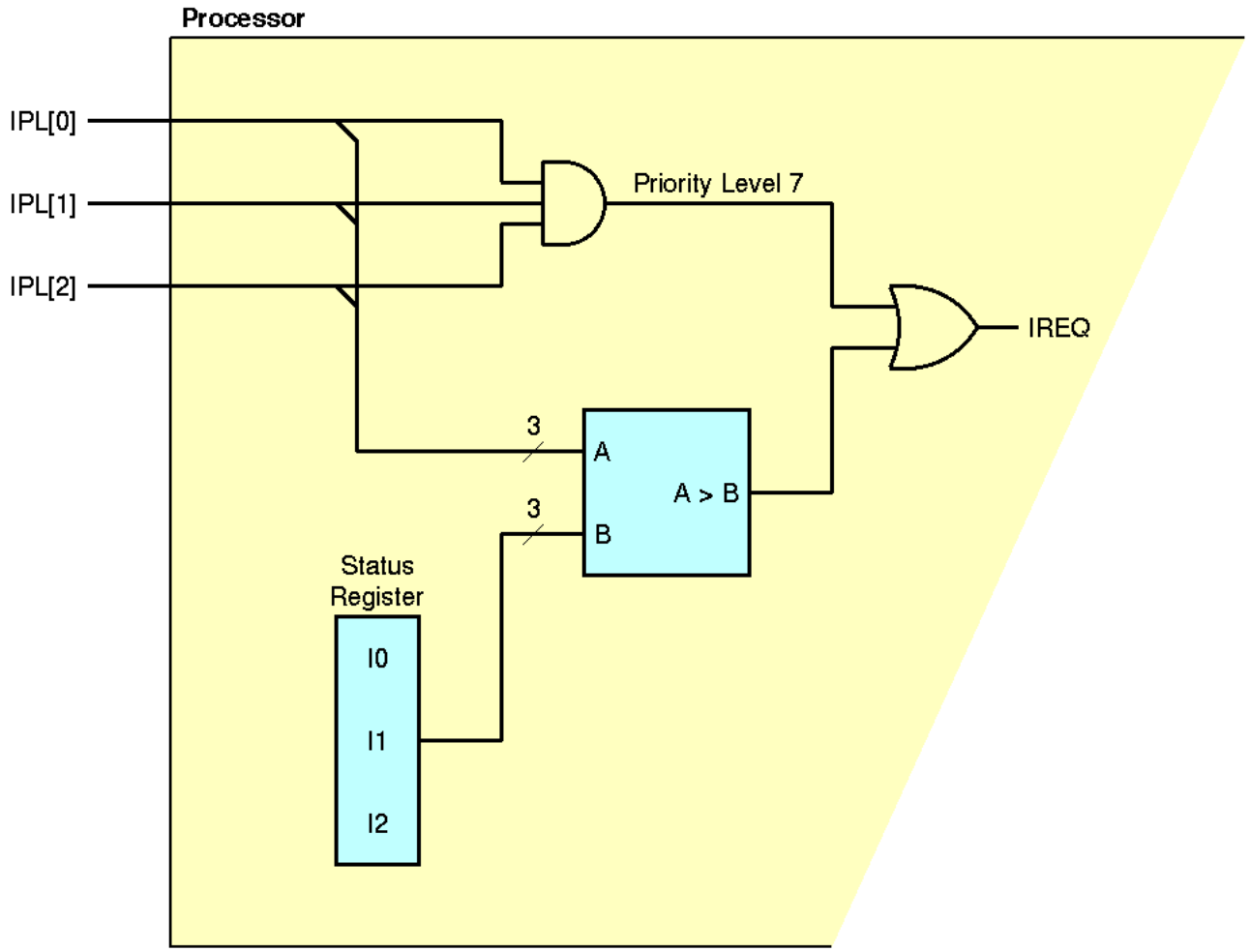


A **character generator**: for each possible digit, the segments which must be lit

Part IIIa: Use Interrupts

- The processor has three **Interrupt Priority Level** input pins **IPL[2:0]** for I/O devices to interrupt it
- The binary number on the pins represents the **Priority Level** of the interrupt (0 = no interrupt request, 1 = lowest priority, 7 = highest)
- An interrupt is **served only if the declared priority level is higher** than the level stored in a special status register of the processor
- Priority **level 7 is always served** (nonmaskable)

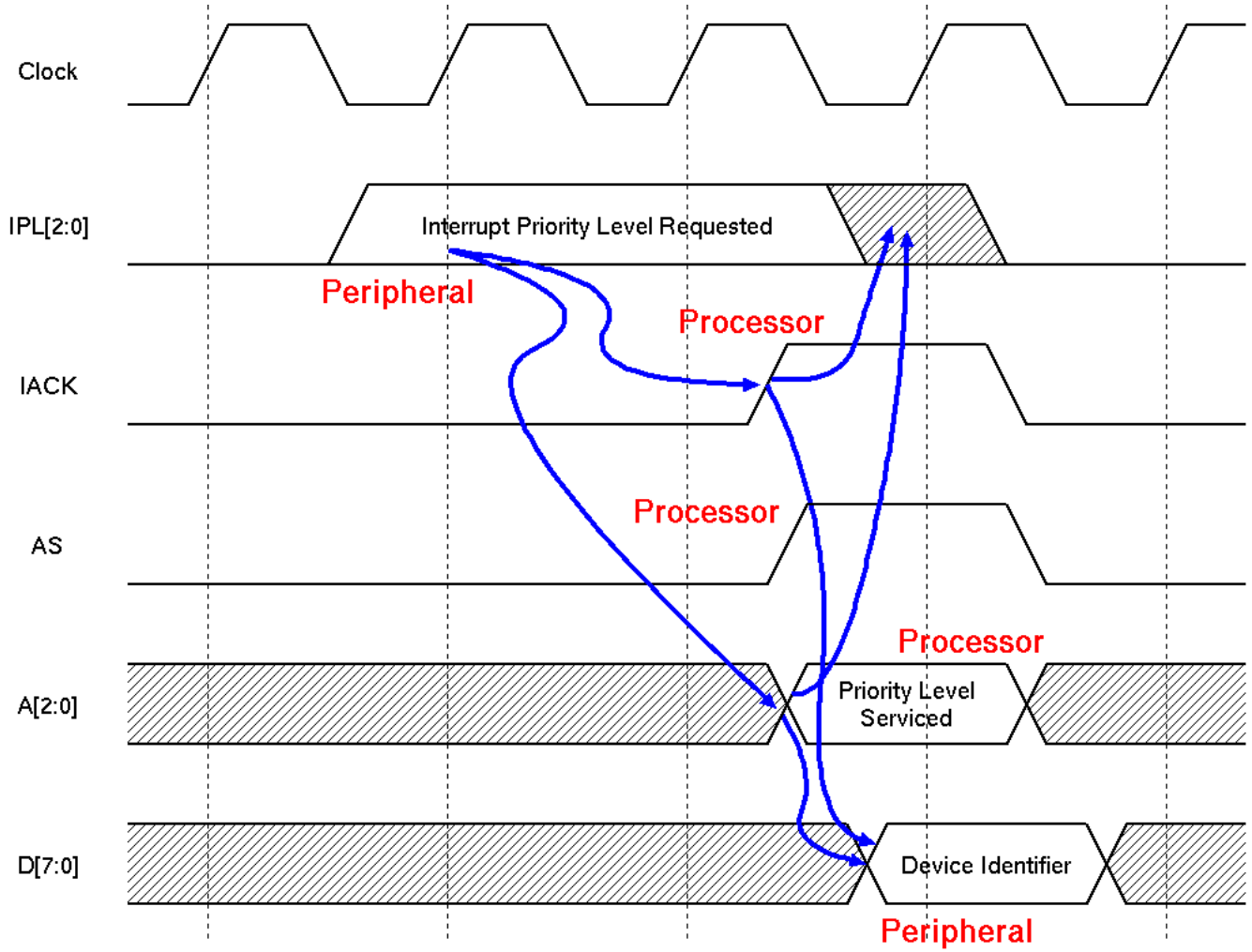
Internal IREQ Mechanism



Part IIIa: Use Interrupts

- The processor indicates **Interrupt Acknowledge** with the output signal **IACK**
- During the acknowledge, the processor indicates the **interrupt level being serviced** on the address pins **A[2:0]** and sets the Address Strobe signal **AS** active
- The interrupting peripheral being acknowledged must give an **exception identifier** through the 8 least significant bits of the data bus **D[7:0]**

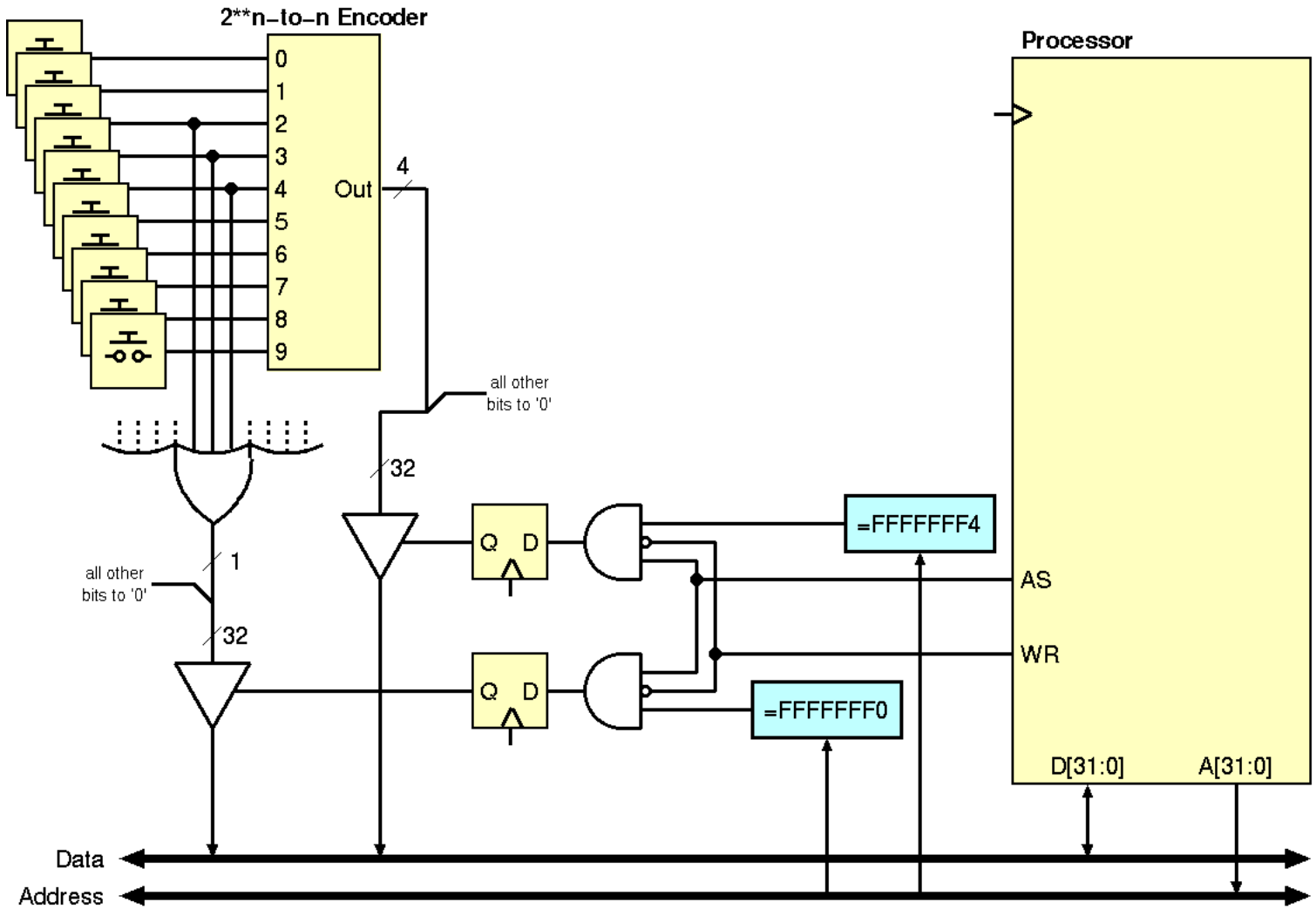
Interrupt Acknowledgement



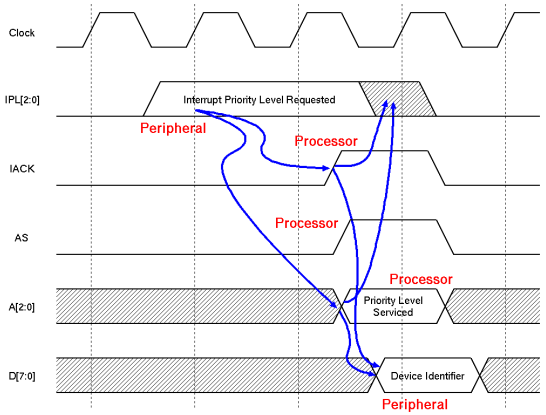
Part IIIa: Use Interrupts

- Modify the button interface to generate an **interrupt with priority level 3** and **identifier 0x45** when a button is pressed
- The port at memory location `0xFFFF'FFF0` is no longer used

Circuit

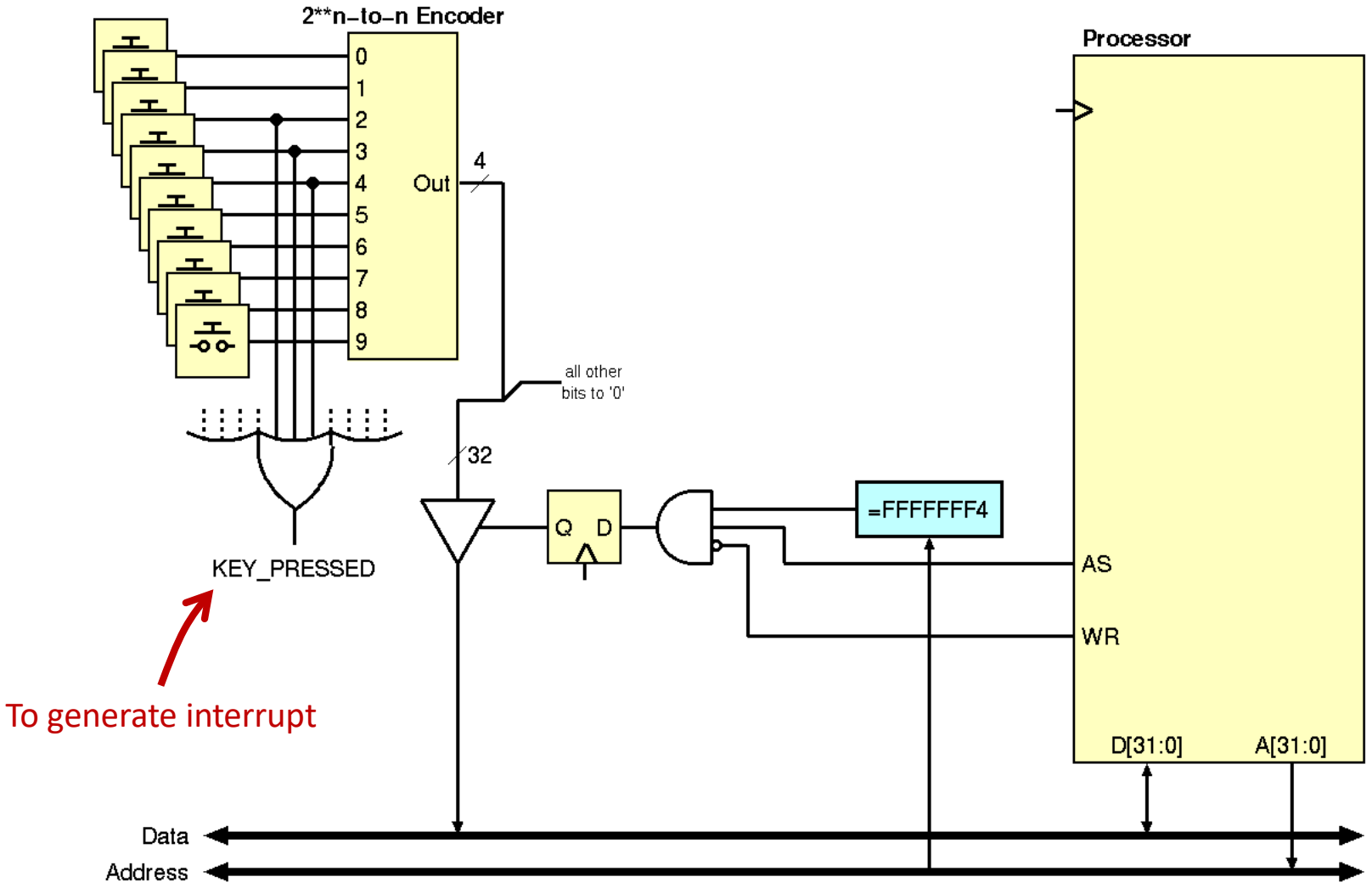

2**n–to–n Encoder
0
1
2
3
4
5
6
7
8
9
Out
4
all other
bits to '0'
32
1
all other
bits to '0'
32
Q D
Q D
=FFFFFFF4
=FFFFFFF0
Processor
AS
WR
D[31:0]
A[31:0]
Data
Address

Circuit

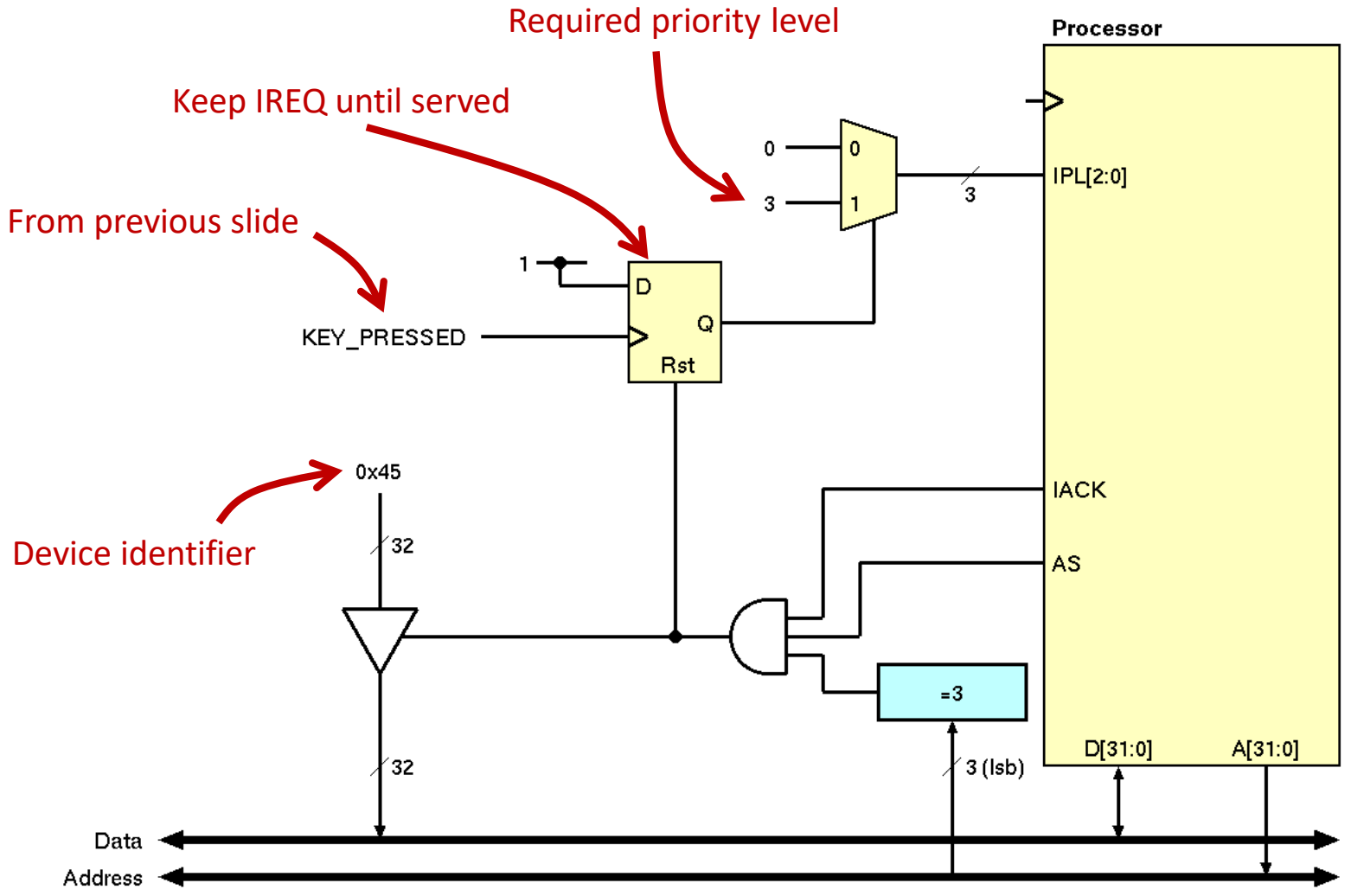

Clock
IPL[2:0]
Interrupt Priority Level Requested
Peripheral
Processor
IACK
Processor
AS
Processor
A[2:0]
Priority Level Serviced
D[7:0]
Device Identifier
Peripheral


Processor
IPL[2:0]
IACK
AS
D[31:0]
A[31:0]
Data
Address

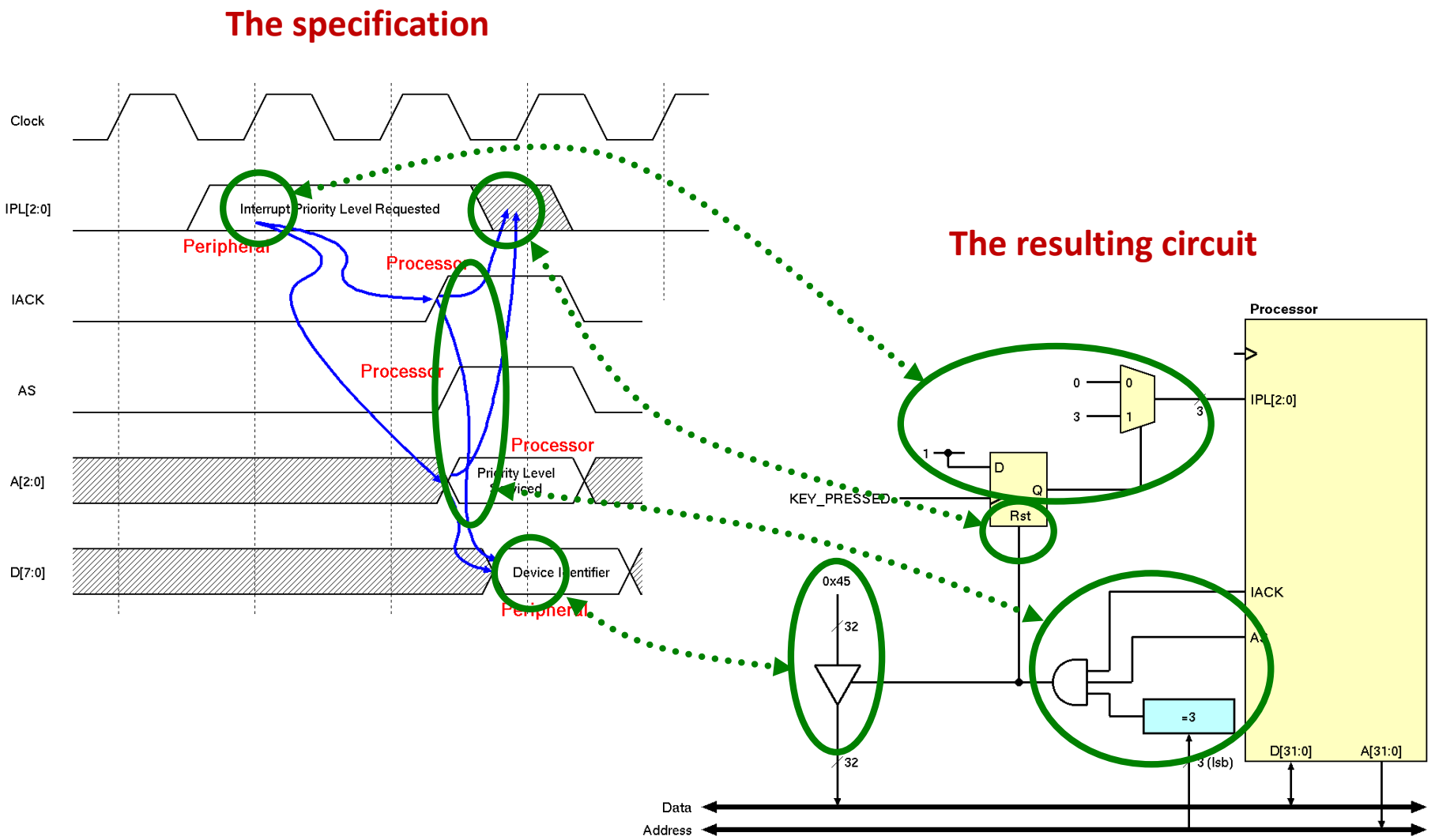
Part IIIa: Solution (What's Left of the Inputs)



Part IIIa: Solution (What's New)



Part IIIa: Solution (Circuit ⇔ Timing)



Part IIIb: Interrupt Handler

- The **main exception handler is part of the operating system**: it determines the exception cause and jumps to an appropriate routine with `mepc` containing the return address
- It **calls** the routine **`interrupt45`** when it receives an interrupt with identifier 0x45 is received
- Write the **`interrupt45`** routine to **read the button pressed and display it** on the seven-segment display
- The stack pointer `sp` can be used but **all registers must be preserved**
- On return, interrupt **enables should be restored** to their original state

Software: `interrupt45`

Part IIIb: Solution: interrupt45

```
interrupt45:    addi  sp, sp, -120          # Save all registers but zero and sp
                sw    x1, 0(sp)
                sw    x3, 4(sp)
                   … etc. …
                sw    x31, 116(sp)

                li    a0, 0xfffffff0        # a0 = 0xffff'fff0
                lw    a0, 4(a0)             # a0 = button (0xffff'fff4)
                addi  a0, a0, 48            # a0 = ASCII of button
                jal   showIt                # Display ASCII digit

restore:        lw    x1, 0(sp)             # Restore all registers but zero and sp
                lw    x3, 4(sp)
                   … etc. …
                lw    x31, 116(sp)
                addi  sp, sp, 120
                mret
```