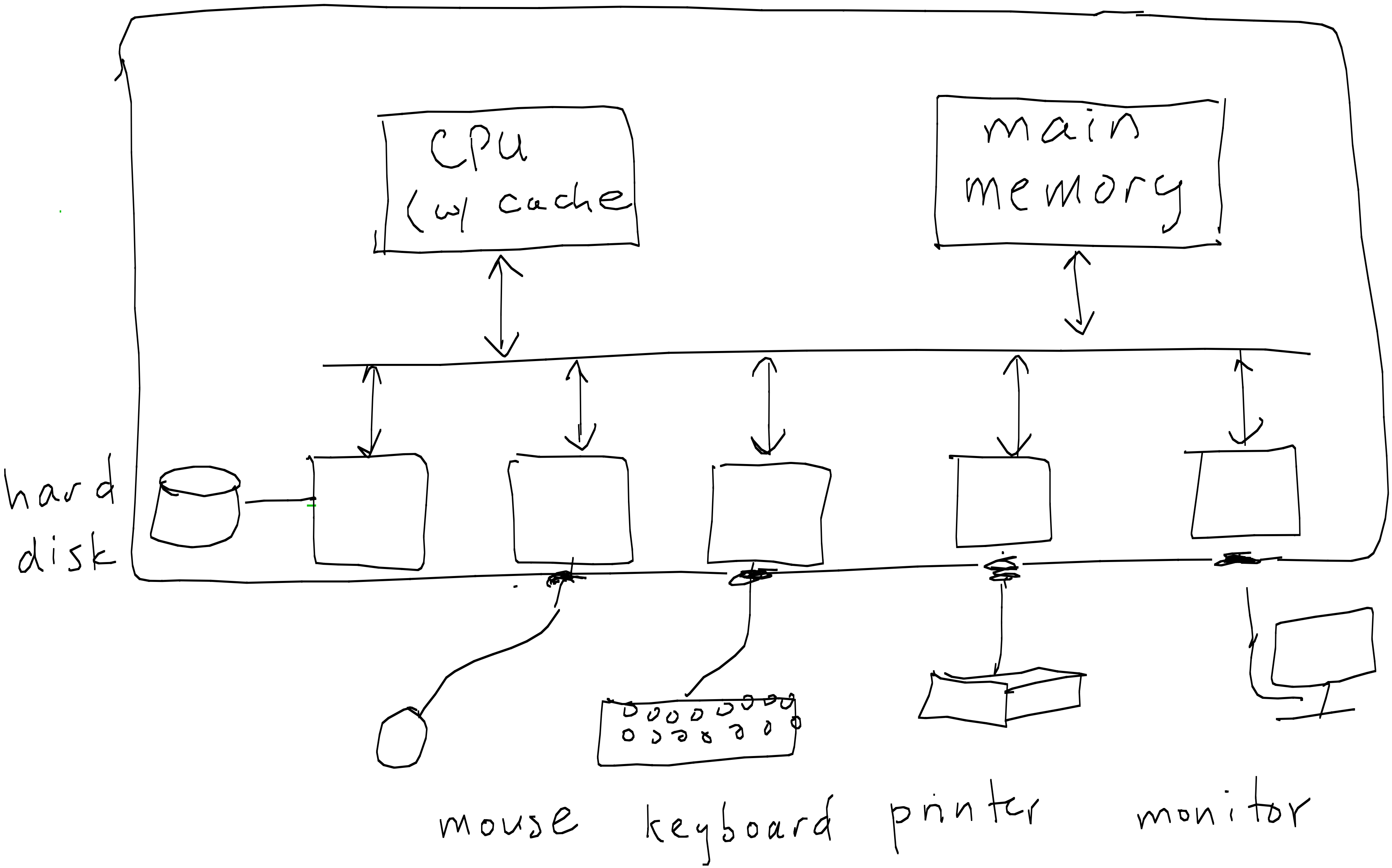
lecture 21

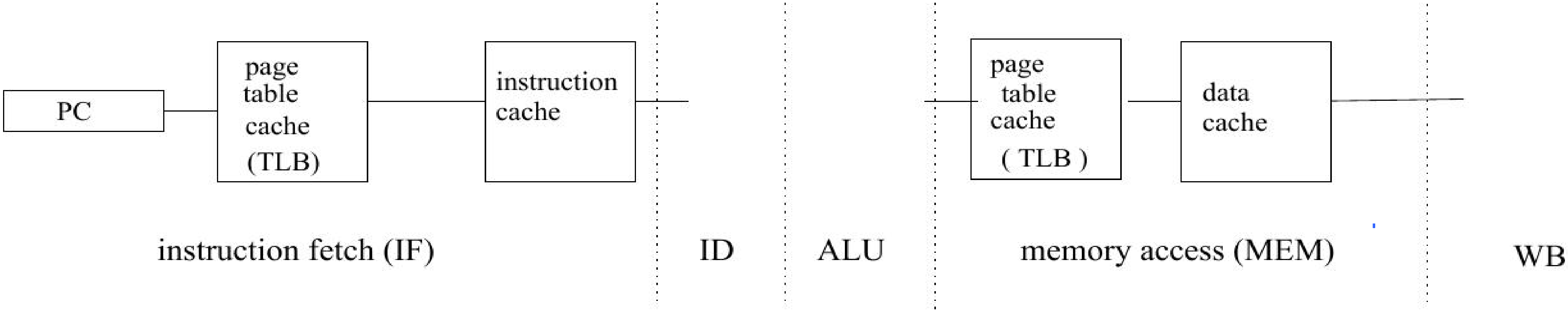
Input / Output (I/O) 3

- system bus and memory (lectures 16-18 revisited)
- system bus and interrupts
- exceptions (lecture 12 revisited)

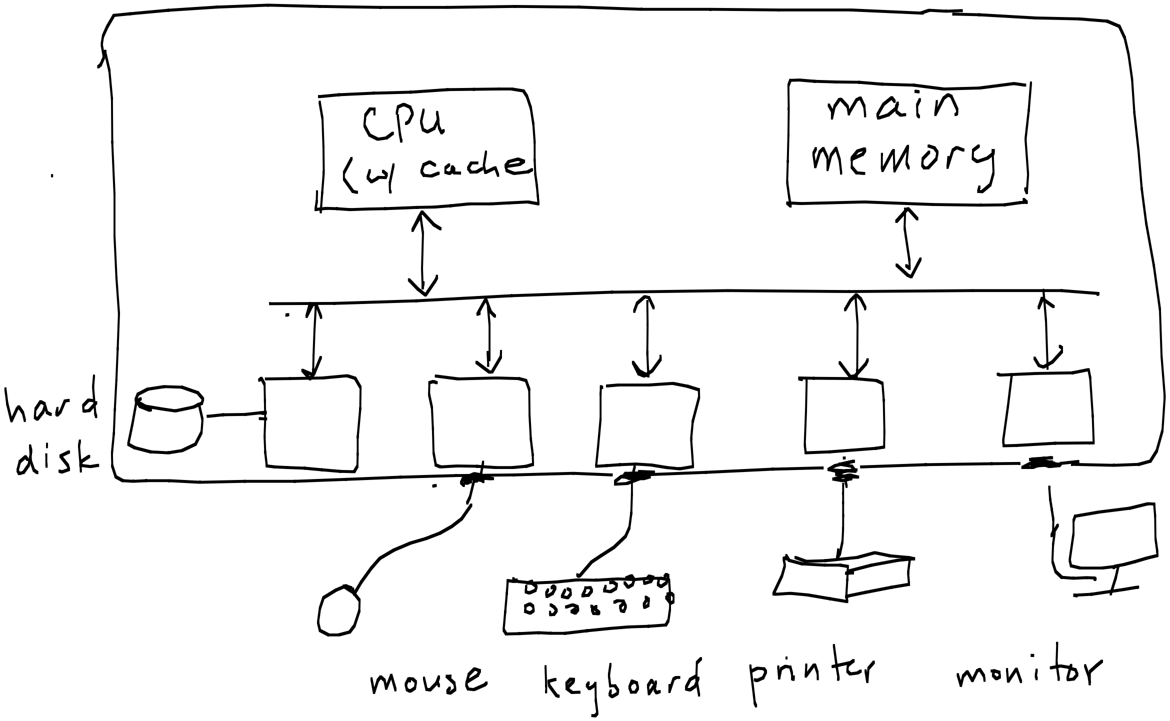
Wed. March 3, 2016


CPU
(w/ cache
main
memory
hard
disk
mouse
keyboard
printer
monitor

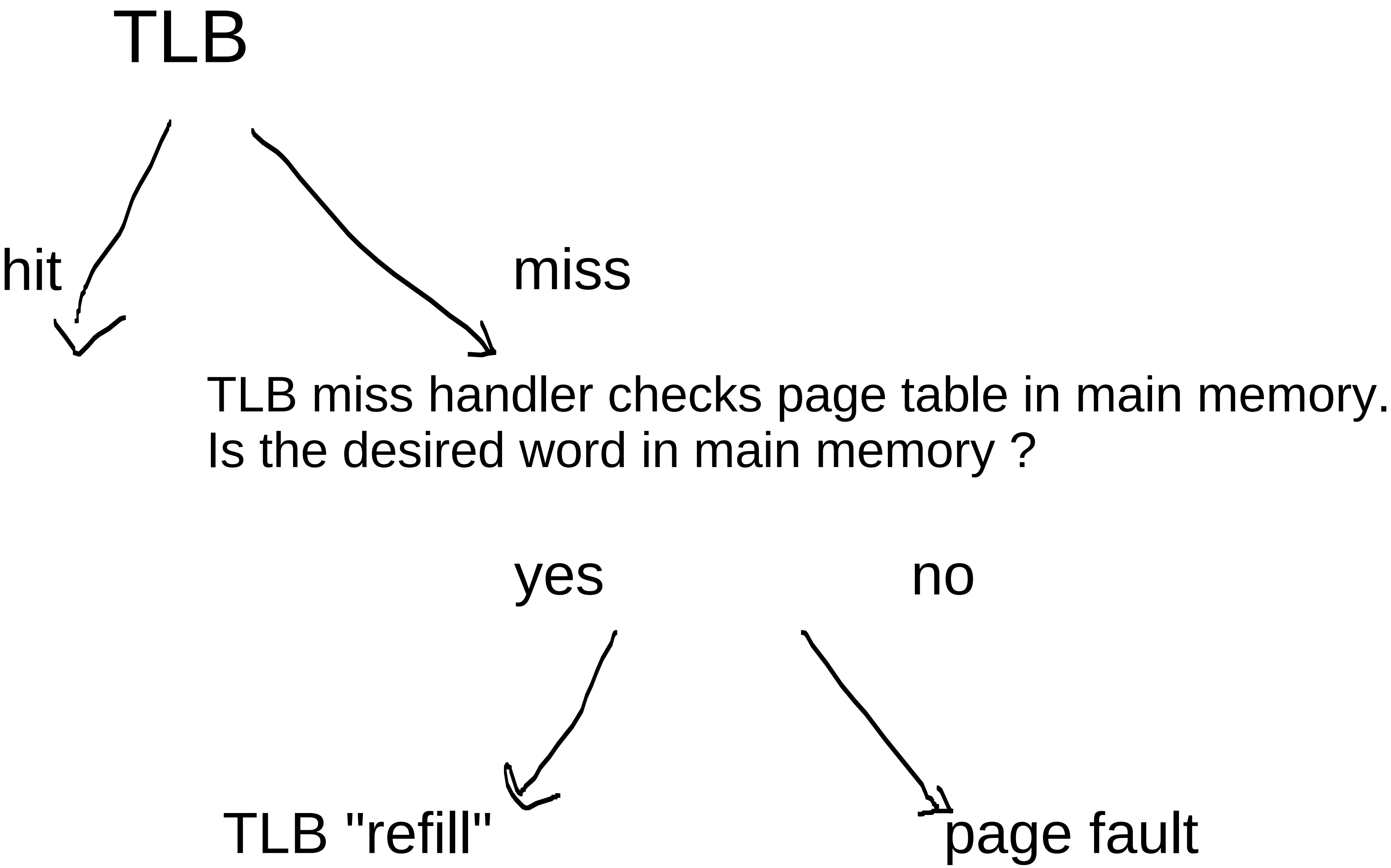
Let's review what happens when there is a TLB miss.



In particular, what happens on the system bus ?



First, recall from lecture 18



TLB

hit

miss

TLB miss handler checks page table in main memory.
Is the desired word in main memory ?

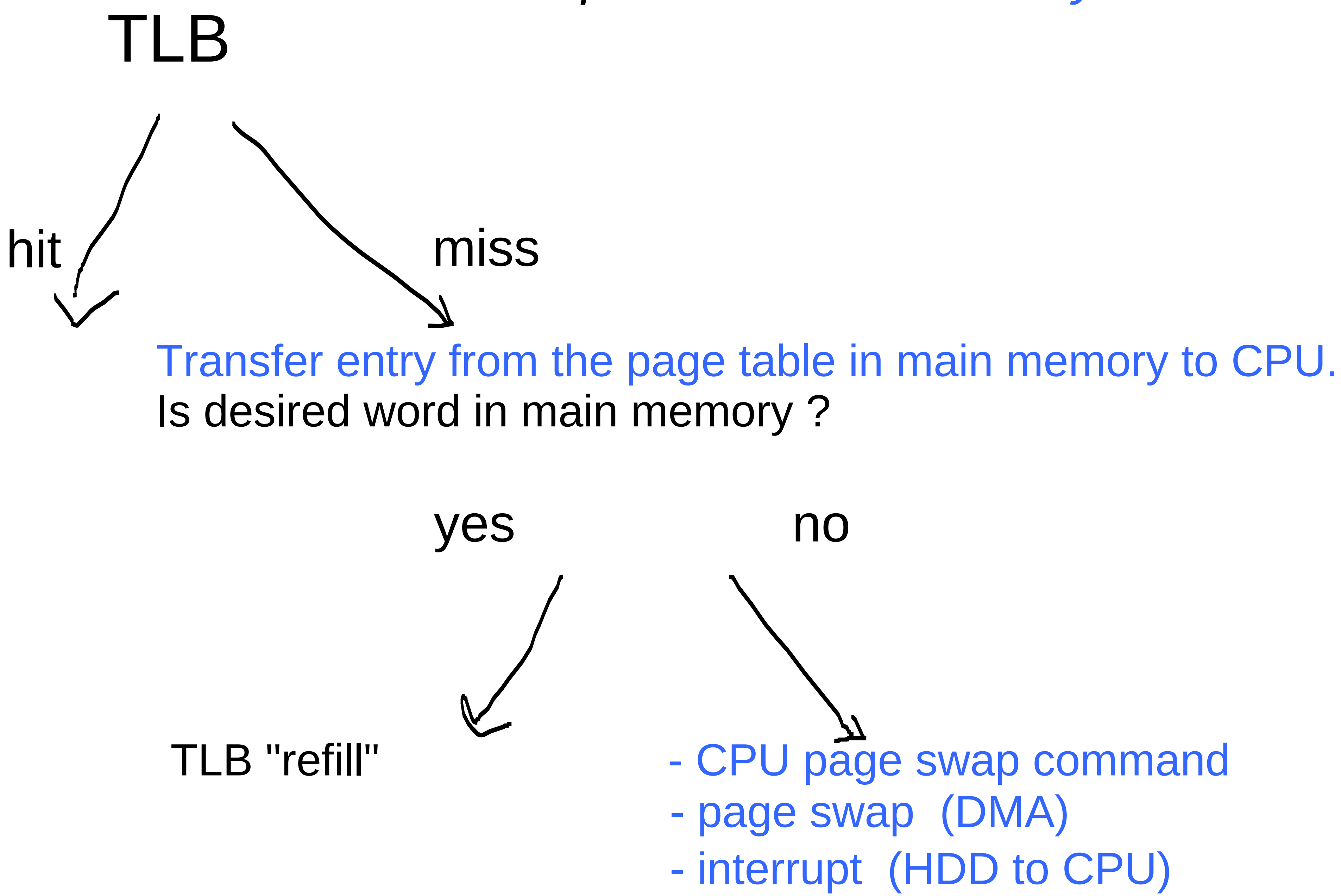
yes

no

TLB "refill"

page fault

Now think about how each step requires access to the system bus.



TLB

hit

miss

Transfer entry from the page table in main memory to CPU.

Is desired word in main memory ?

yes

no

TLB "refill"

- CPU page swap command
- page swap (DMA)
- interrupt (HDD to CPU)

lecture 21

Input / Output (I/O) 3

- system bus and memory (lectures 16-18 revisited)
- interrupts (and system bus)
- exceptions (lecture 12 revisited)

Wed. March 3, 2016

Recall from last lecture (system bus intro)

- polling

  CPU to I/O #n: "are you ready?   are you ready? ..."

- direct memory access (DMA)

  CPU to I/O #n: "use the bus to do X,
  and tell me when you are done"

(External) Interrupts

I/O device to CPU: "stop what you are doing and do Y"

e.g. keyboard (echo/render to display, d<enter> causes action)
mouse (render to display)
printer (out of paper, render message to user)

- interrupt request (IRQ) lines are used to make requests (not system bus)

Similar general idea as with DMA.
But there are important differences.

General questions about interrupt requests

-How does an I/O device make an interrupt request ?

-How to coordinate multiple I/O devices that may *all* make interrupt requests ? Can one device interrupt another ?

-What does the CPU do when it gets an interrupt request ?

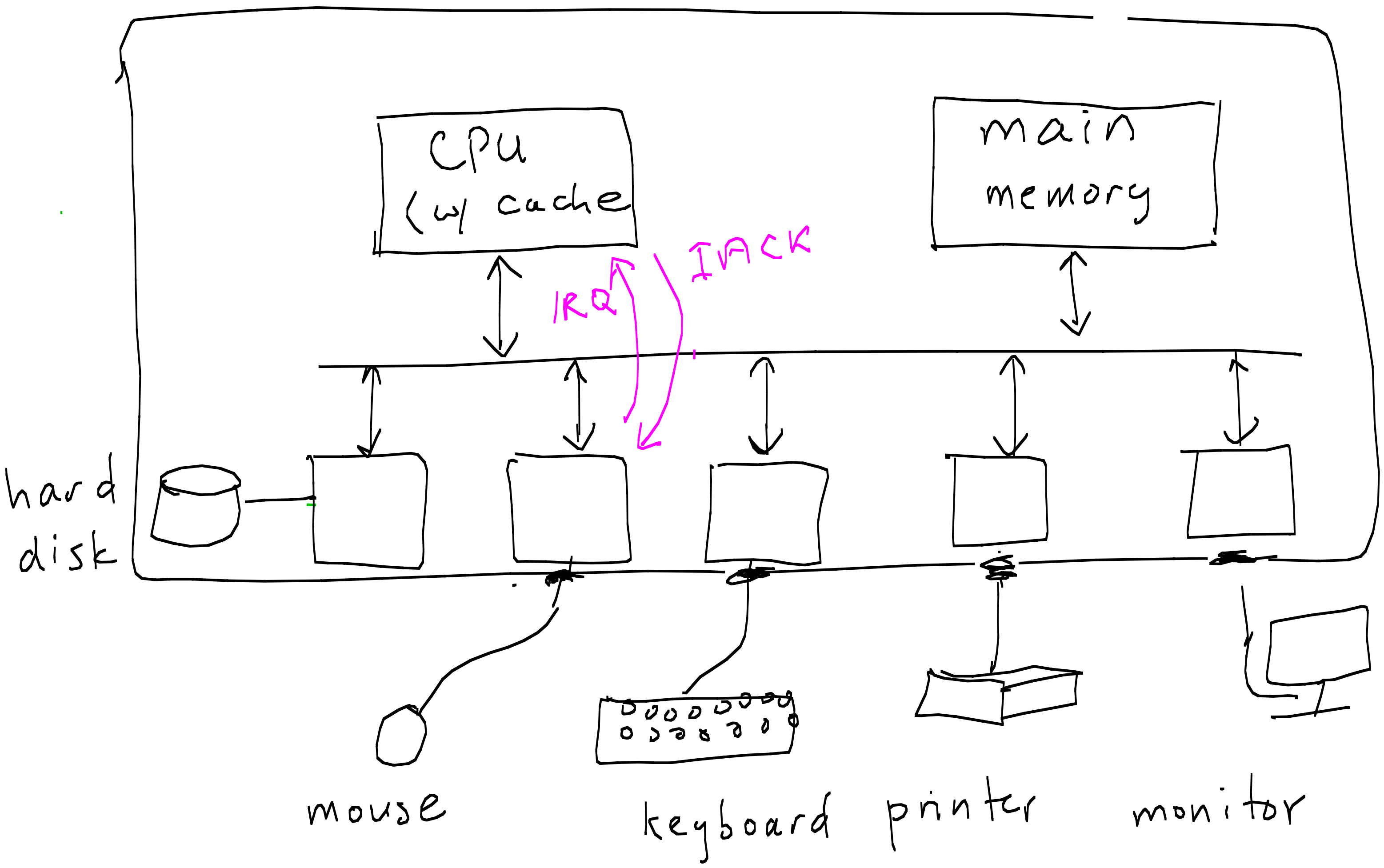
Let's examine the first two questions. I'll look at three methods.

1) each I/O device has its own IRQ, IACK lines

(similar in flavour to BR/BG in DMA)

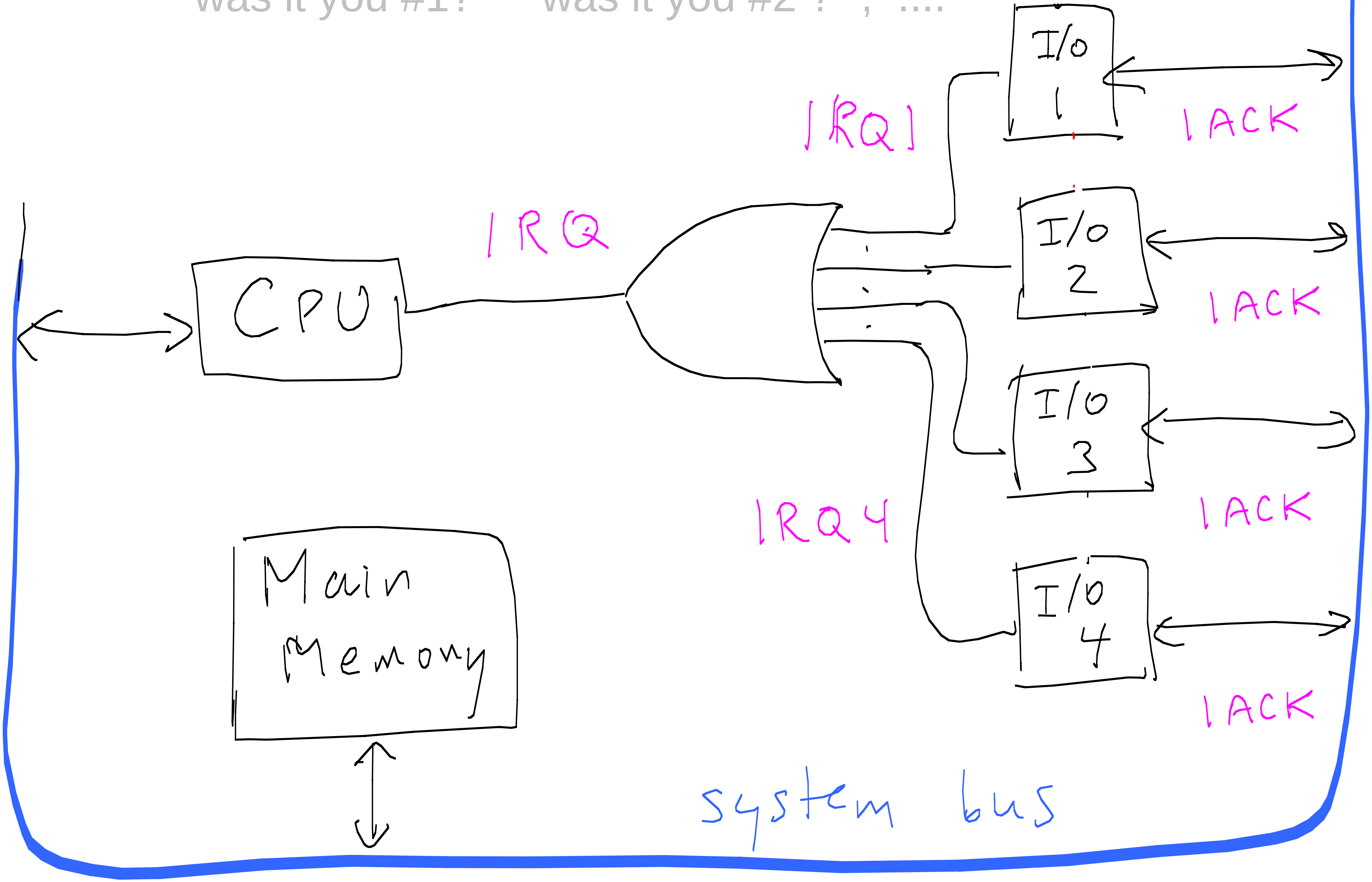
Disadvantages:

- many lines
- places burden entirely on CPU to decide on priorities

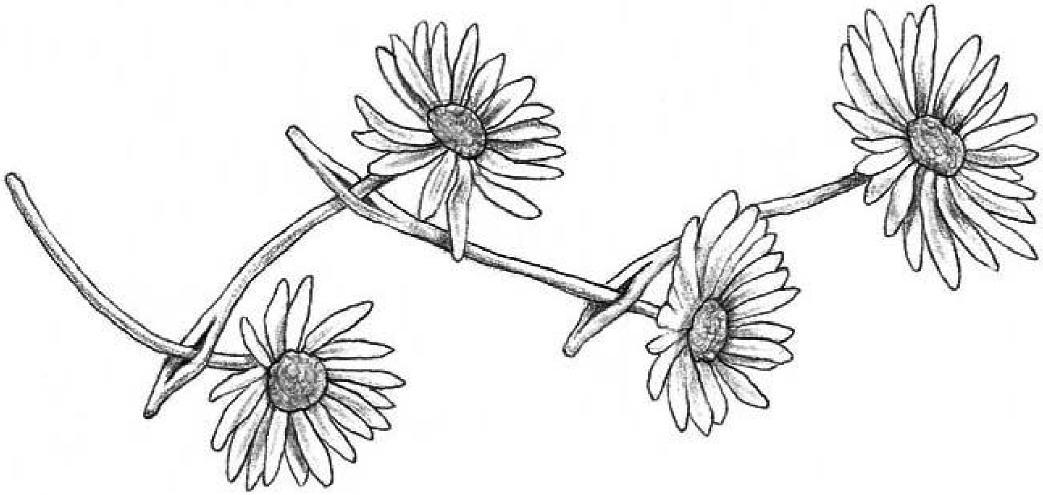


2.) shared IRQ + polling

Disadvantage: - burden still on CPU to decide whether to handle the interrupt (based on priority)
"was it you #1?" "was it you #2 ?" , ...

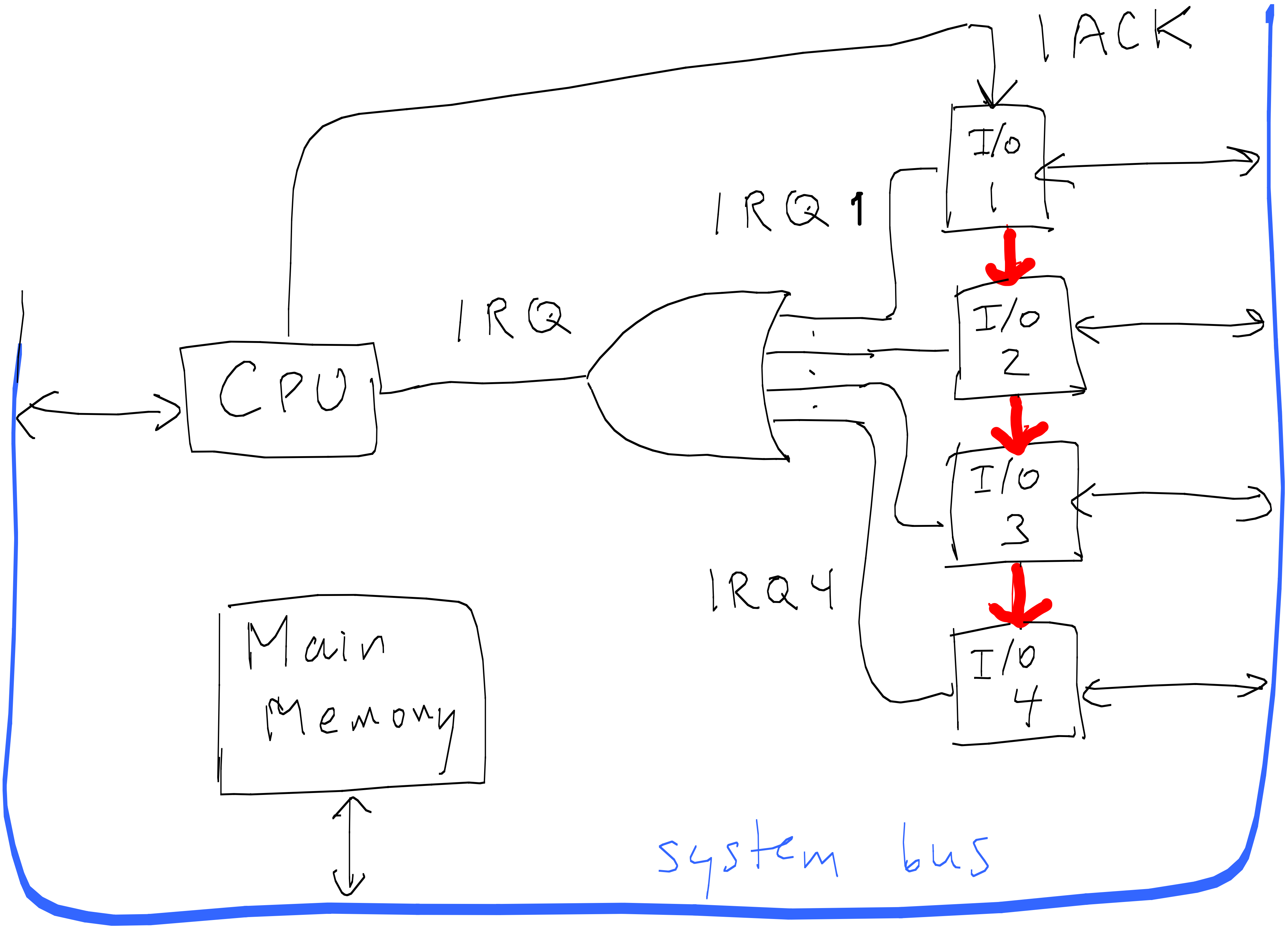


3. Daisy Chain

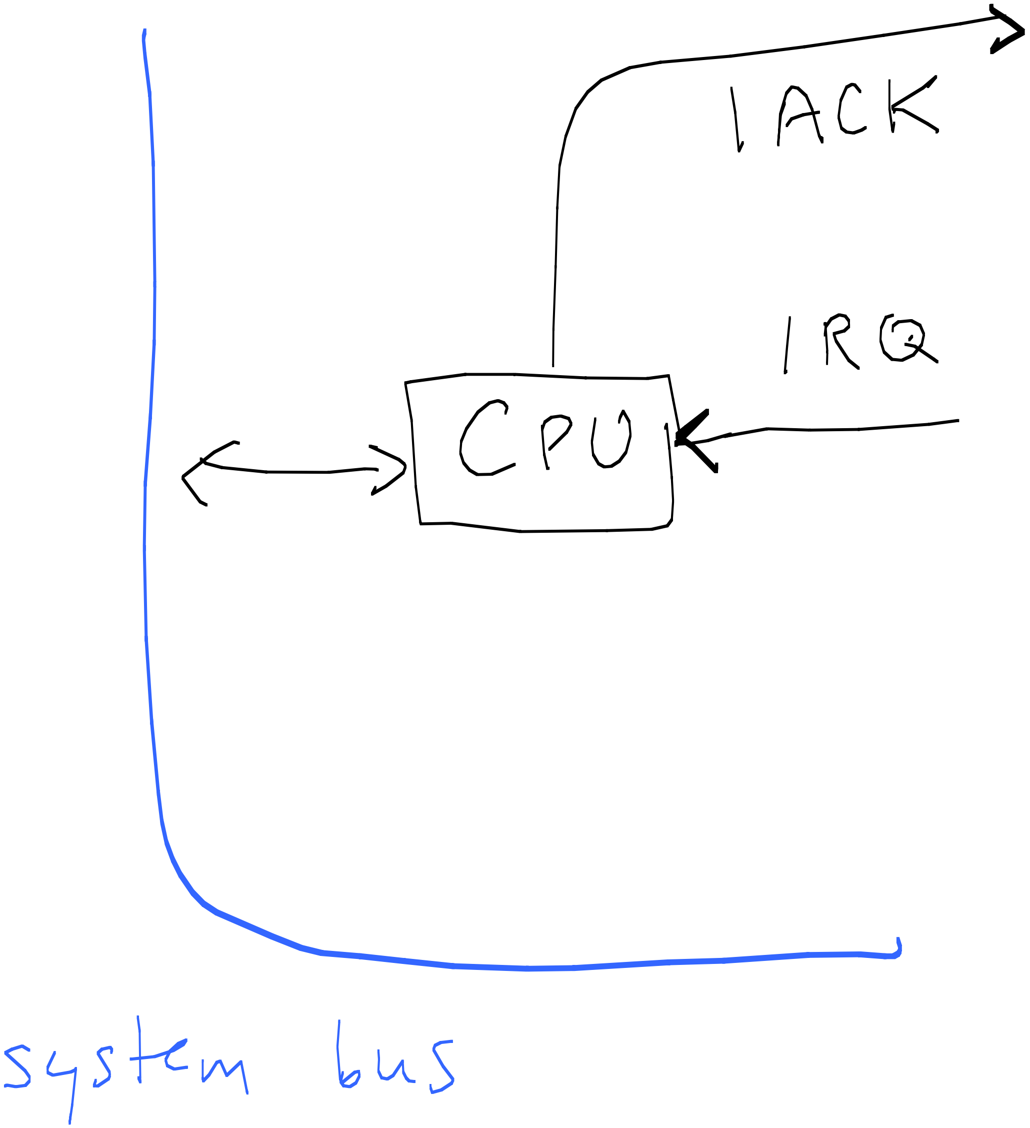


No, I don't know the historical origins of this term in bus design.

3.) Daisy Chain (priority ordering = physical ordering)

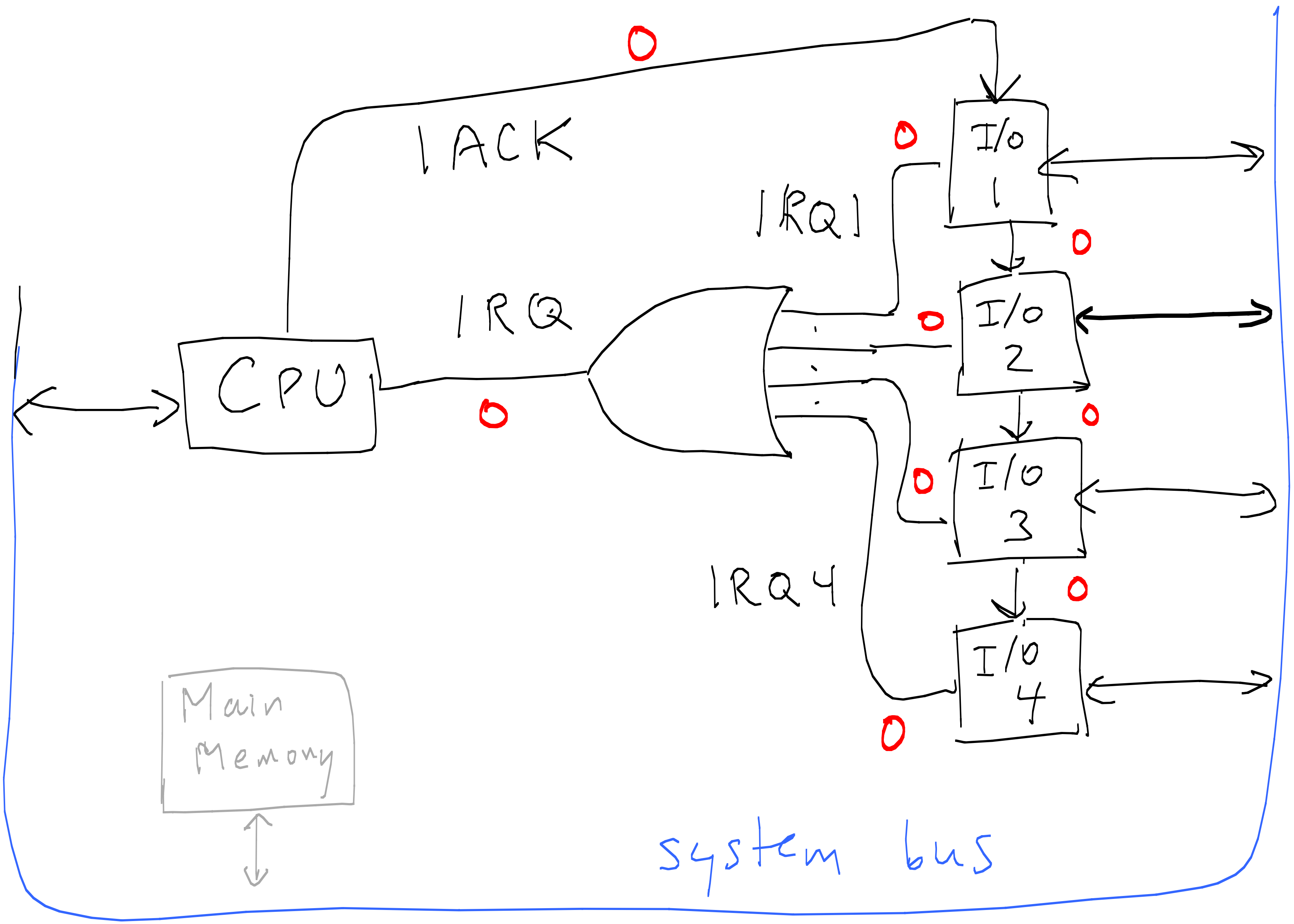


The key advantage of daisy chaining is that the I/O controllers (not the CPU) decide on priorities of interrupt requests.   HOW ?



If the CPU is available then it allows the I/O device to use the bus to make an interrupt request.

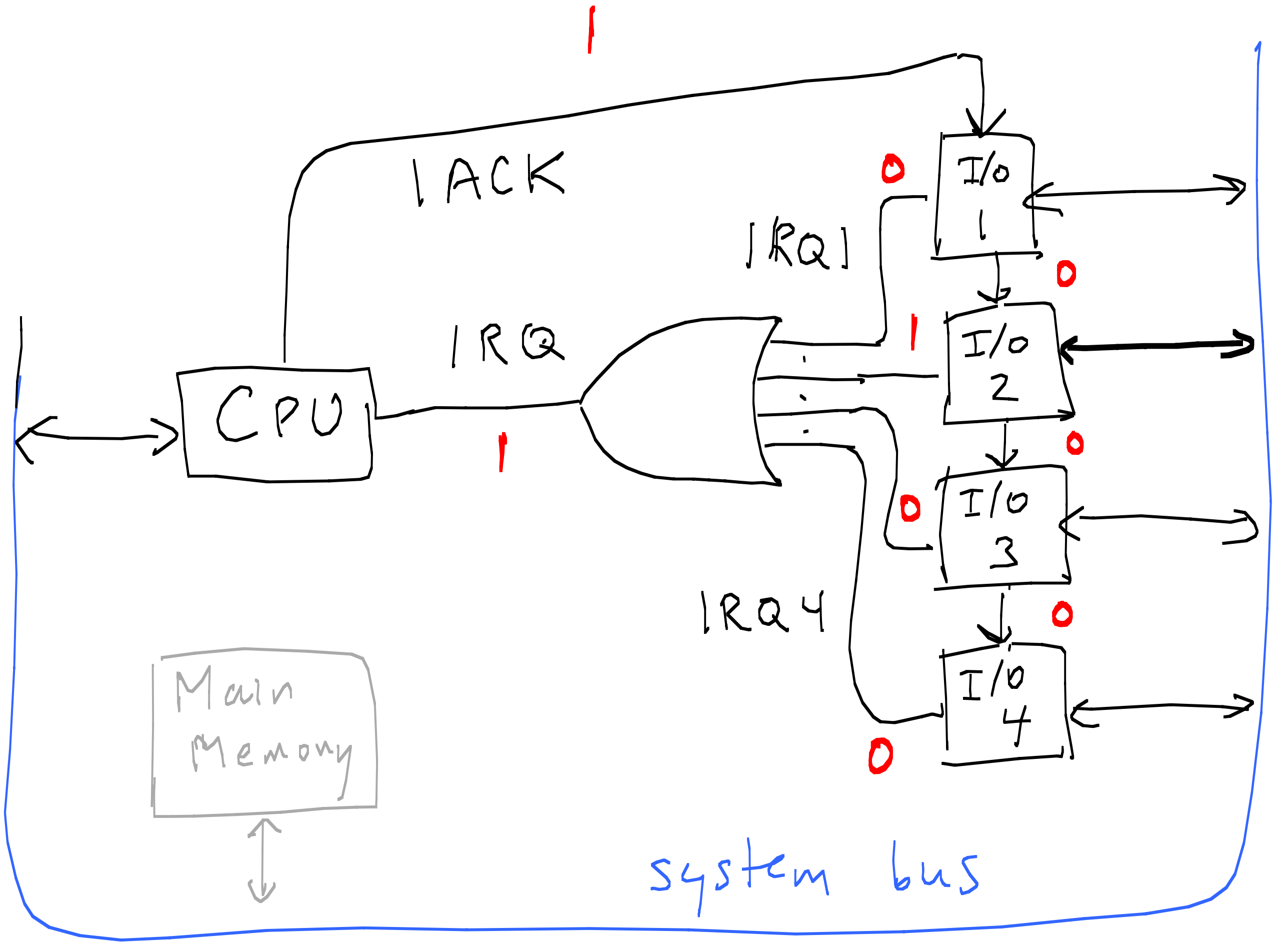
Suppose all IRQ signals are initially 0.

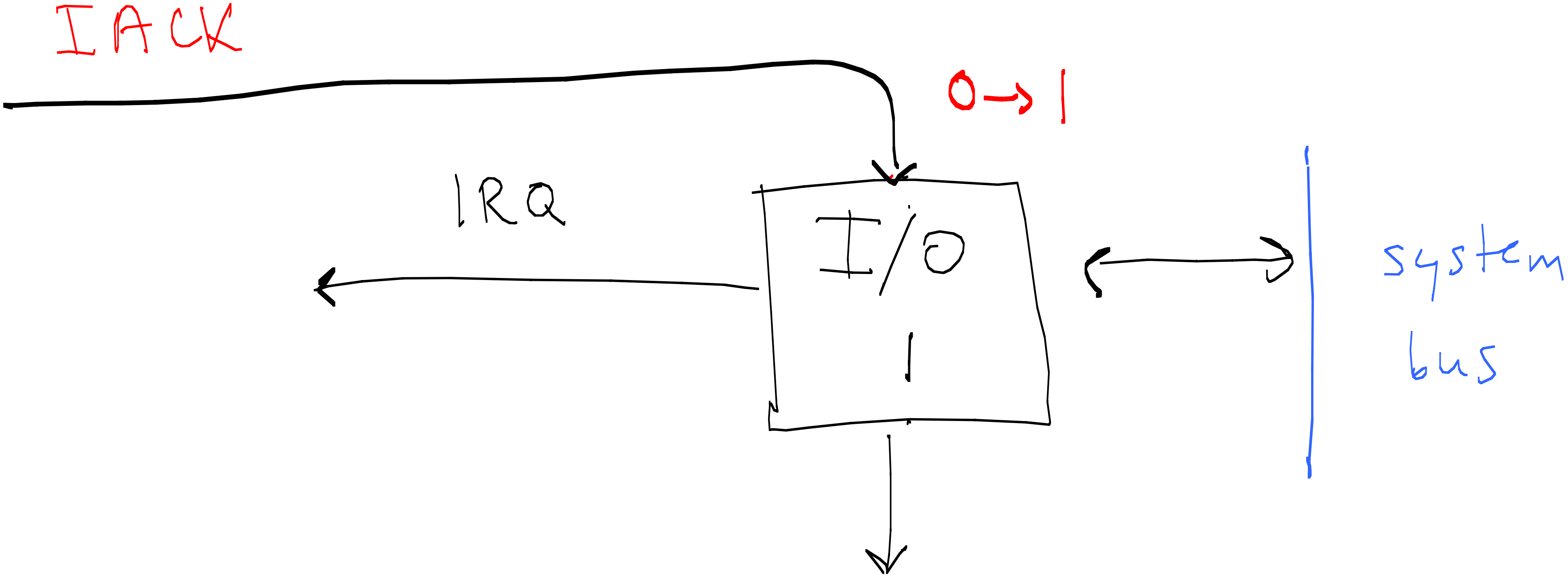


Suppose IRQ 2 makes an interrupt request

IRQ 2 = 1   :   "knock, knock"

IACK = 1 :   "who is there?"
(and CPU frees up system bus temporarily)



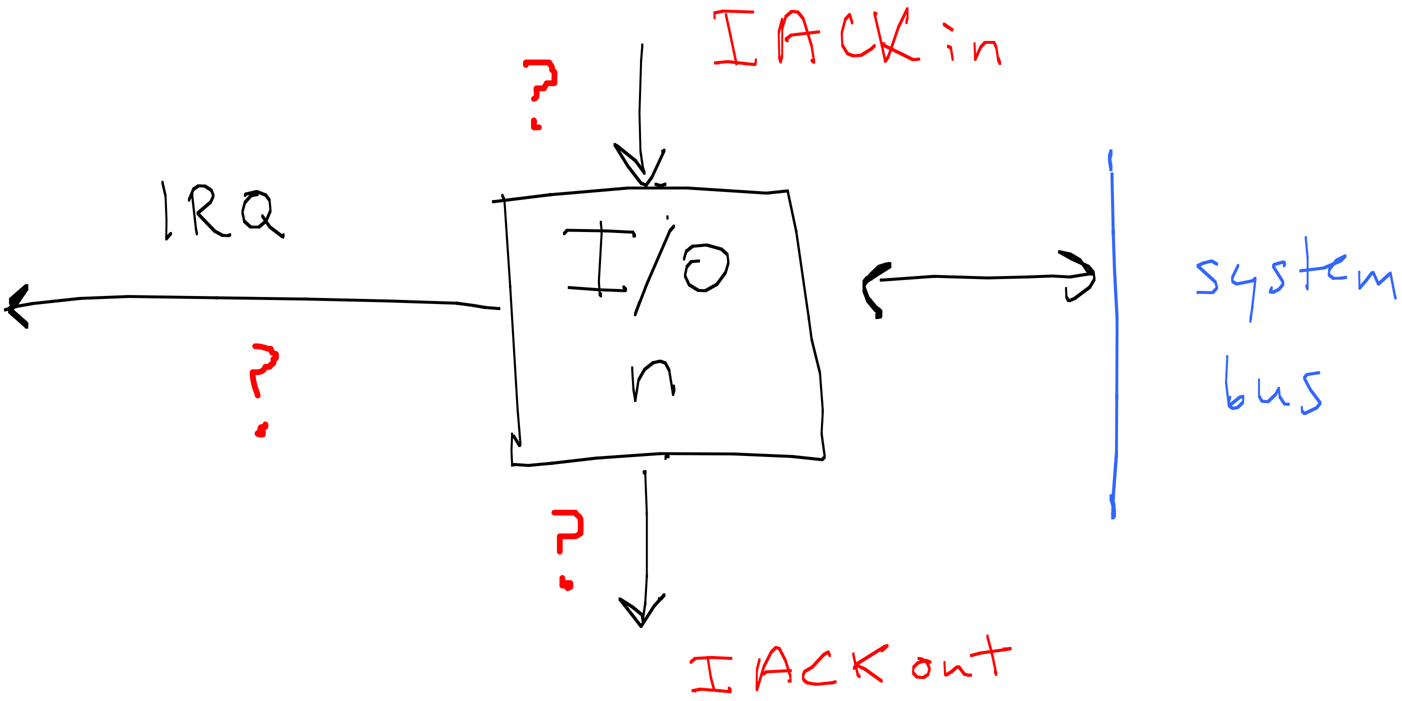


I/O device 1 observes IACK change from 0 to 1.

If device 1 made the interrupt request,
then it uses the system bus to communicate with CPU
(and it passes a 0 to the next device controller).

else it sets to 1 the signal to the next I/O device controller.

More generally...   I/O device controller n



I/O device n observes its IACK in   change from 0 to 1.

If device n made the interrupt request, then it sets its IACK out to 0, and uses the system bus to communicate with CPU.

Otherwise it sets its IACK out to 1.

IRQ = 1 : "knock, knock" (recall OR gate)

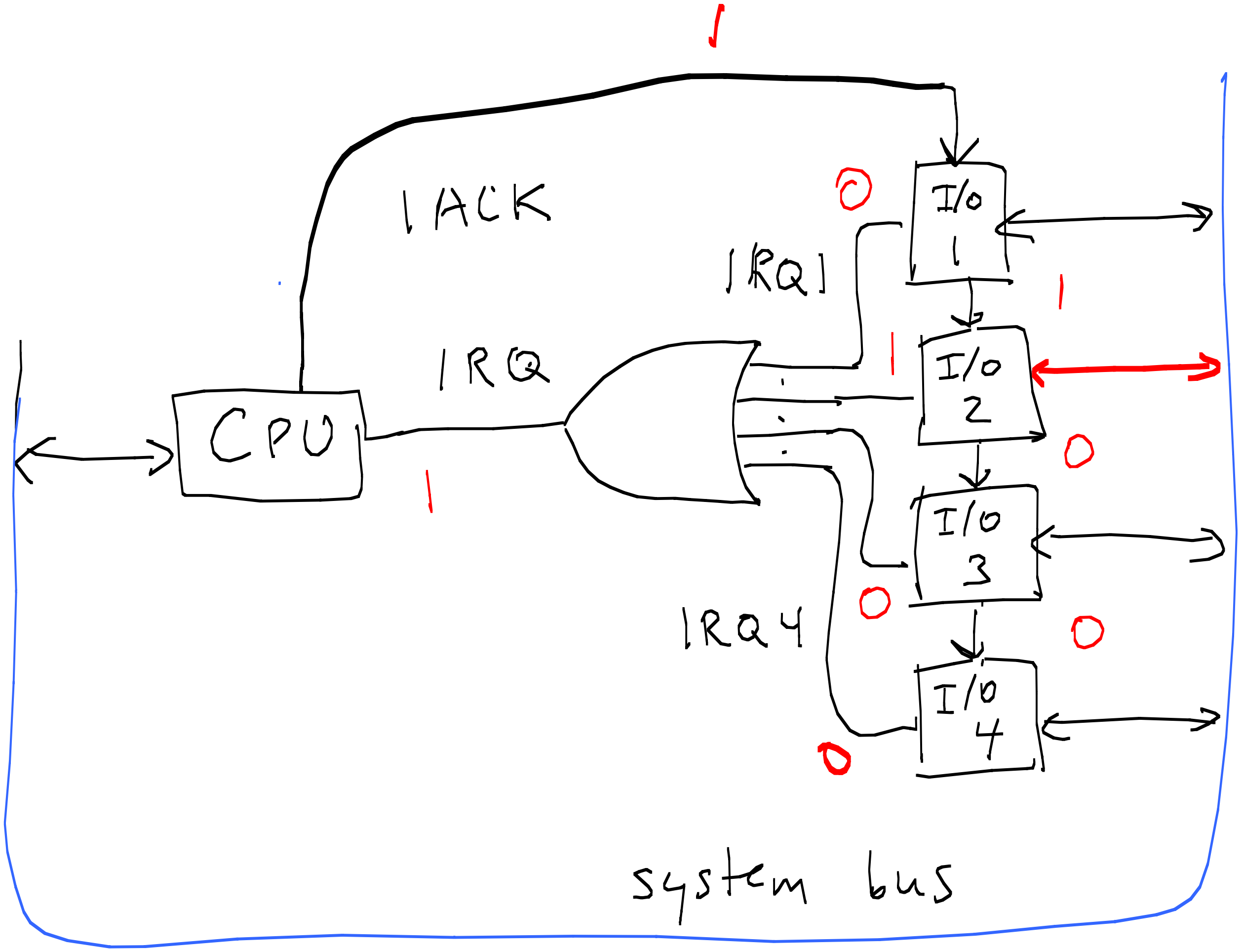
IACK = 1 (CPU) : "who is there?"
(and CPU frees up system bus)

IACK gets passed down to highest priority device that initiated IRQ.

I/O device n : "its me, device n" (on system bus)

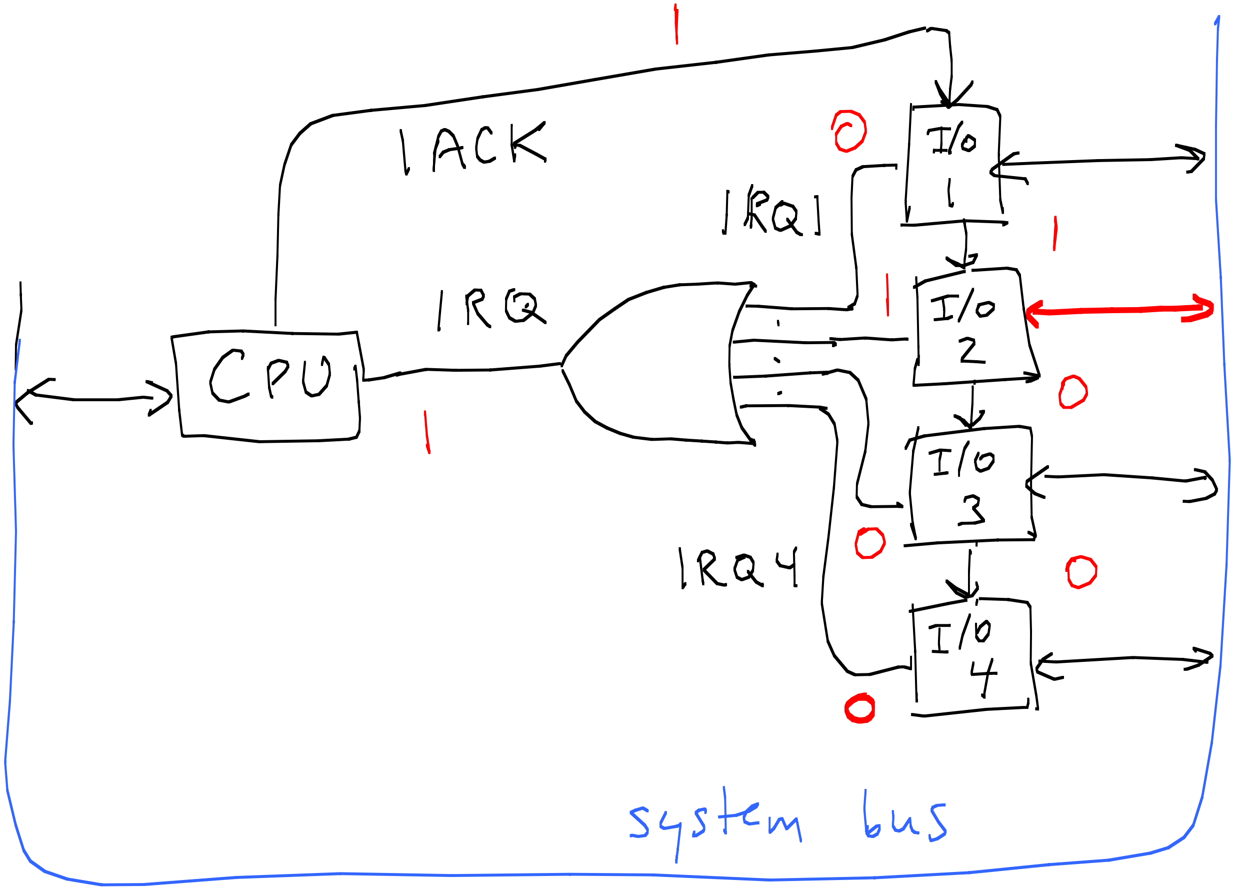
$$\text{IACK (CPU)} = \begin{cases} 0, & \text{"go away, you are not important enough"} \\ 1, & \text{"yes, what can I do for you?"} \end{cases}$$

A higher priority device can interrupt a lower priority device.
How ?



Q: How does I/O device 1 interrupt I/O device 2 ?
A: I/0 device 1 sets its IACK out to 0
I/0 device 2 wraps up and sets IRQ2 to 0. Tries again

A lower priority device cannot interrupt a lower priority device. Why not ?



Q: Why can't I/O device 3 interrupt I/O device 2 ?

A: IRQ3 goes through the same OR gate as IRQ2.
I/O is not allowed to write onto system bus until it sees
IACK in go from 0 to 1 (indicating bus has been freed up)

General questions about interrupt requests

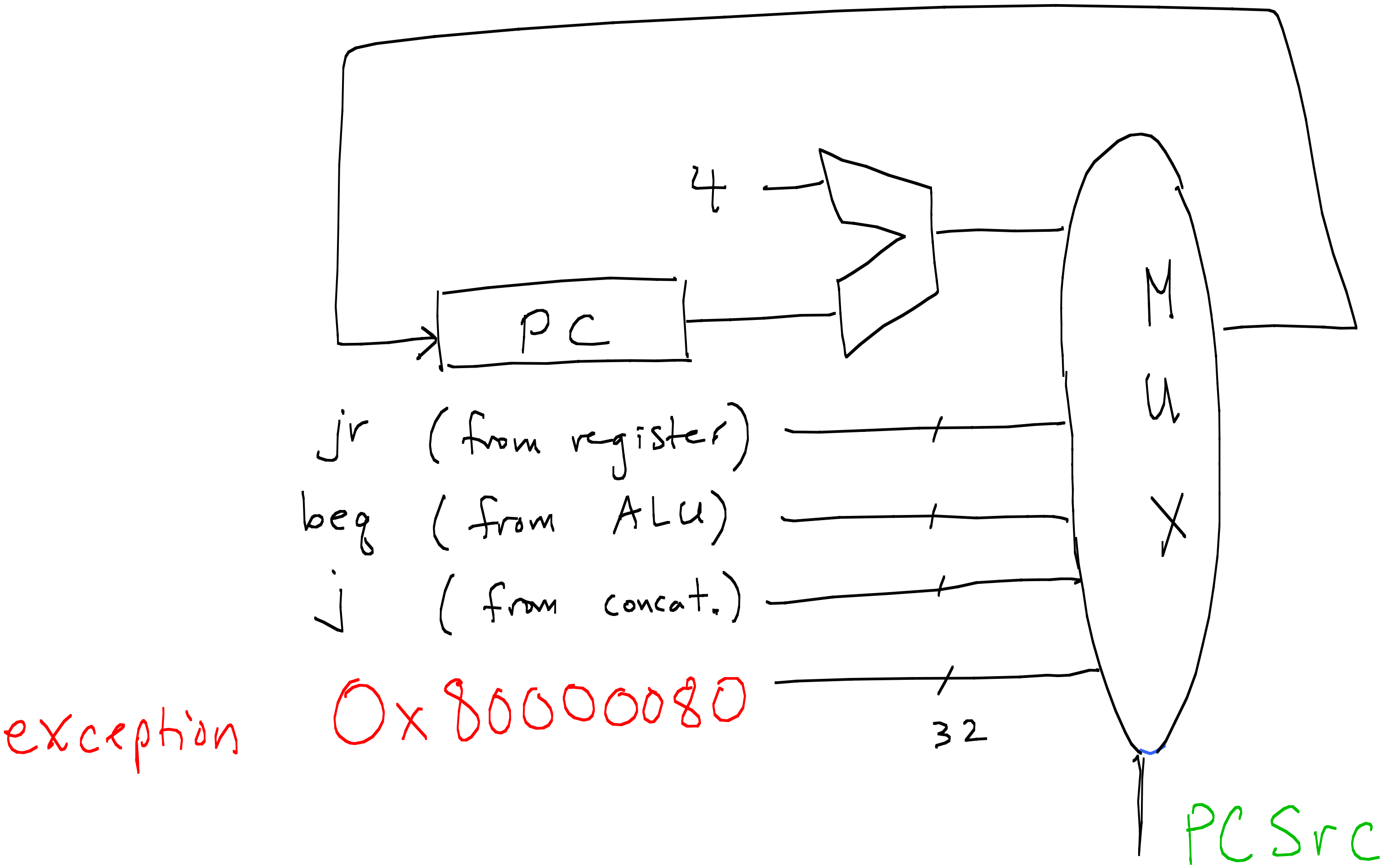
How does an I/O device make a request ?

How to coordinate multiple I/O devices that may all make interrupt requests ? Can one device interrupt another ?

What does the CPU do when it gets an interrupt request ?

- now think MIPS

- interrupt request may be from an I/O device (external interrupt), or from a CPU exception (internal interrupt).



Program jumps to exception handler in the kernel.

Here we have a more general notion of interrupt, which can include both external interrupts (I/O, just discussed) and internal interrupts ("exceptions" such as overflow, division by 0, bad Address, timer = 0).

Interrupt handler

Decide if the interrupt should be handled (sketched in a few slides)

If yes,

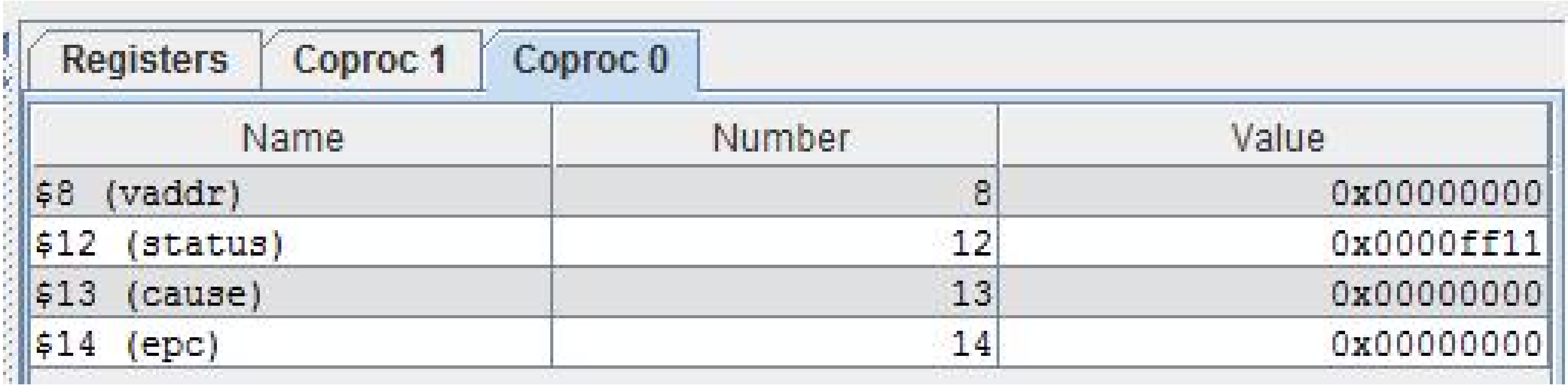
- disable *all* interrupts
- save state of current process
  (store register values in kernel memory)
- enable higher priority interrupts only
- service the interrupt
- possibly run other processes
- restore the state of the process and return to it

Note there are TWO reasons why an interrupt might be *ignored*:

- the CPU is currently handling a higher priority interrupt

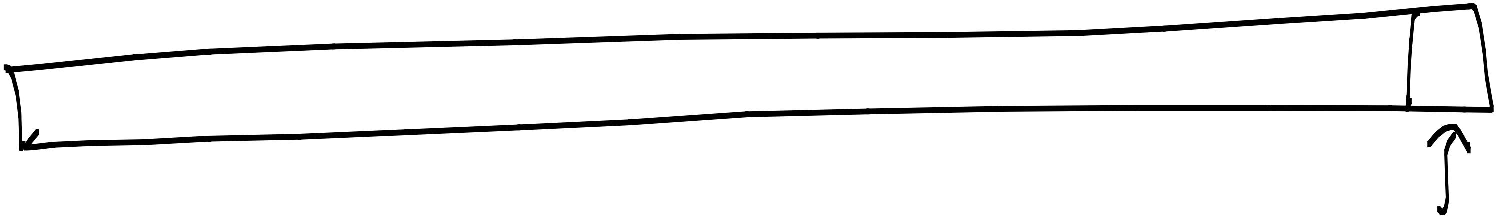
- the CPU is handling a lower priority interrupt BUT it needs to finish saving state information, before that interrupt itself can be interrupted.

Recall coprocessor 0 (lecture 12)



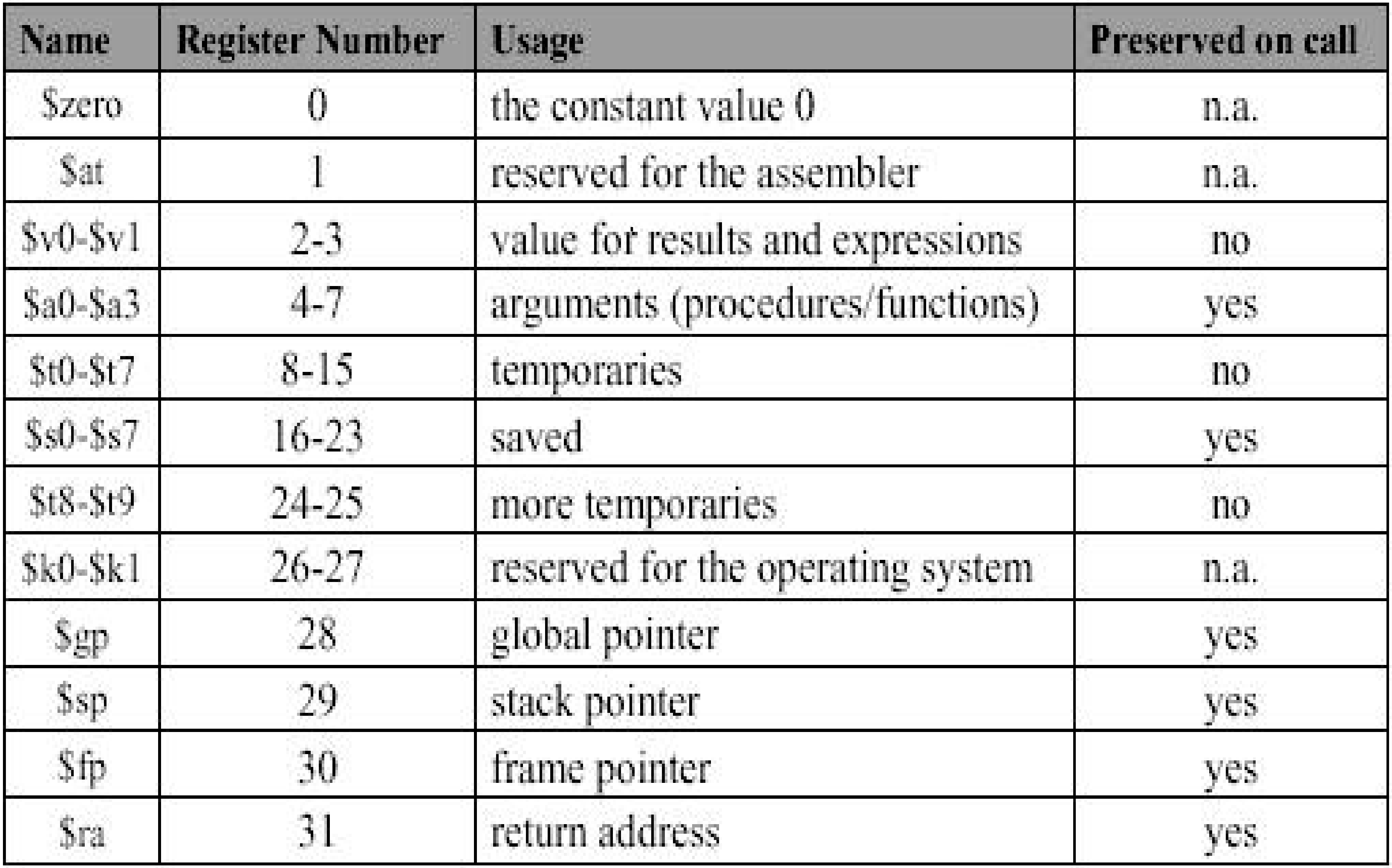
Name	Number	Value
$8 (vaddr)	8	0x00000000
$12 (status)	12	0x0000ff11
$13 (cause)	13	0x00000000
$14 (epc)	14	0x00000000

In particular, $12 is the status register



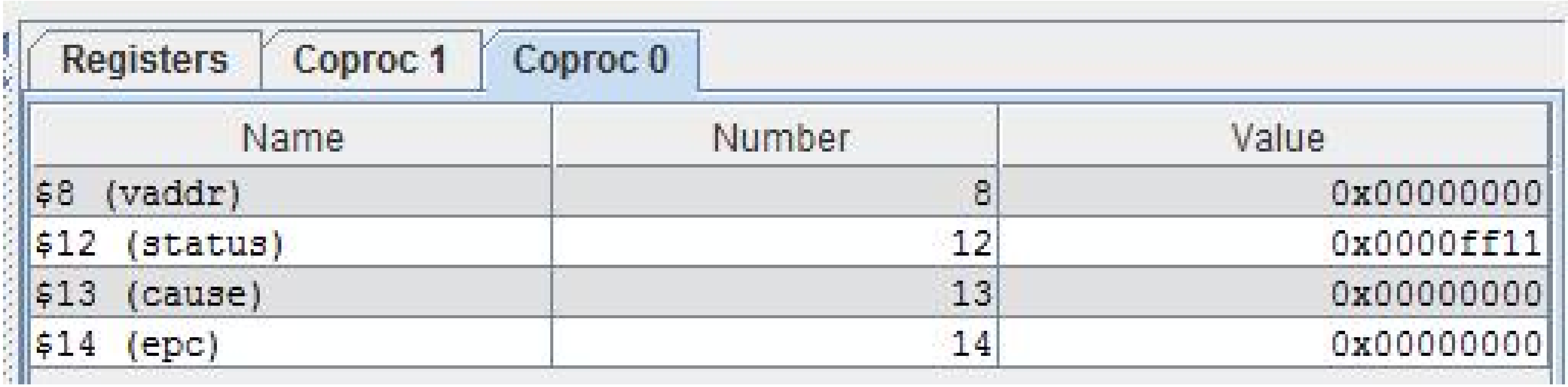
bit 0 of $12 is
"interrupt enable"

MIPS registers



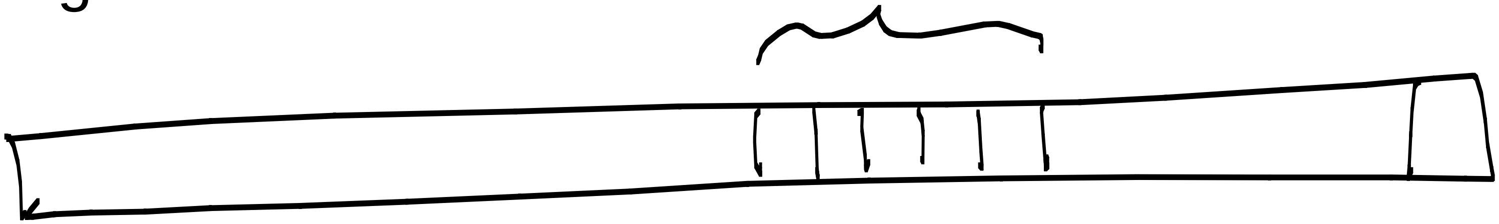
Name	Register Number	Usage	Preserved on call
$zero	0	the constant value 0	n.a.
$at	1	reserved for the assembler	n.a.
$v0-$v1	2-3	value for results and expressions	no
$a0-$a3	4-7	arguments (procedures/functions)	yes
$t0-$t7	8-15	temporaries	no
$s0-$s7	16-23	saved	yes
$t8-$t9	24-25	more temporaries	no
$k0-$k1	26-27	reserved for the operating system	n.a.
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes

```
# enable interrupts

mfco    $k0,   $12     #      $12  is Status (on coprocessor 0)
ori      $k0,   $k0,   0x01    # turn on only LSB
mtc0     $12,  $k0


# disable interrupt

mfco    $k0,   $12
lui      $1,    0xffff
ori      $1,    $1,    0xfffe     (turn off LSB of $12)
and      $k0,  $k0,  $1
mtc0     $12,  $k0
```



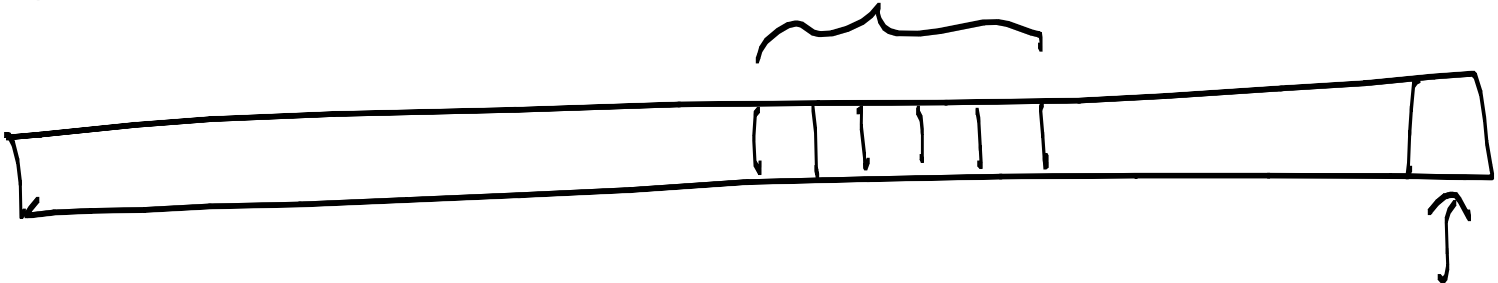
Name	Number	Value
$8 (vaddr)	8	0x00000000
$12 (status)	12	0x0000ff11
$13 (cause)	13	0x00000000
$14 (epc)	14	0x00000000

status register



enables/disables different priority interrupts

cause register



specifies which priority interrupt has occured

I am omitting many nasty details here.

The main idea is that the kernel program in MIPS needs to compare bits in the cause and status registers of c0 to determine whether the interrupt should be handled or not.

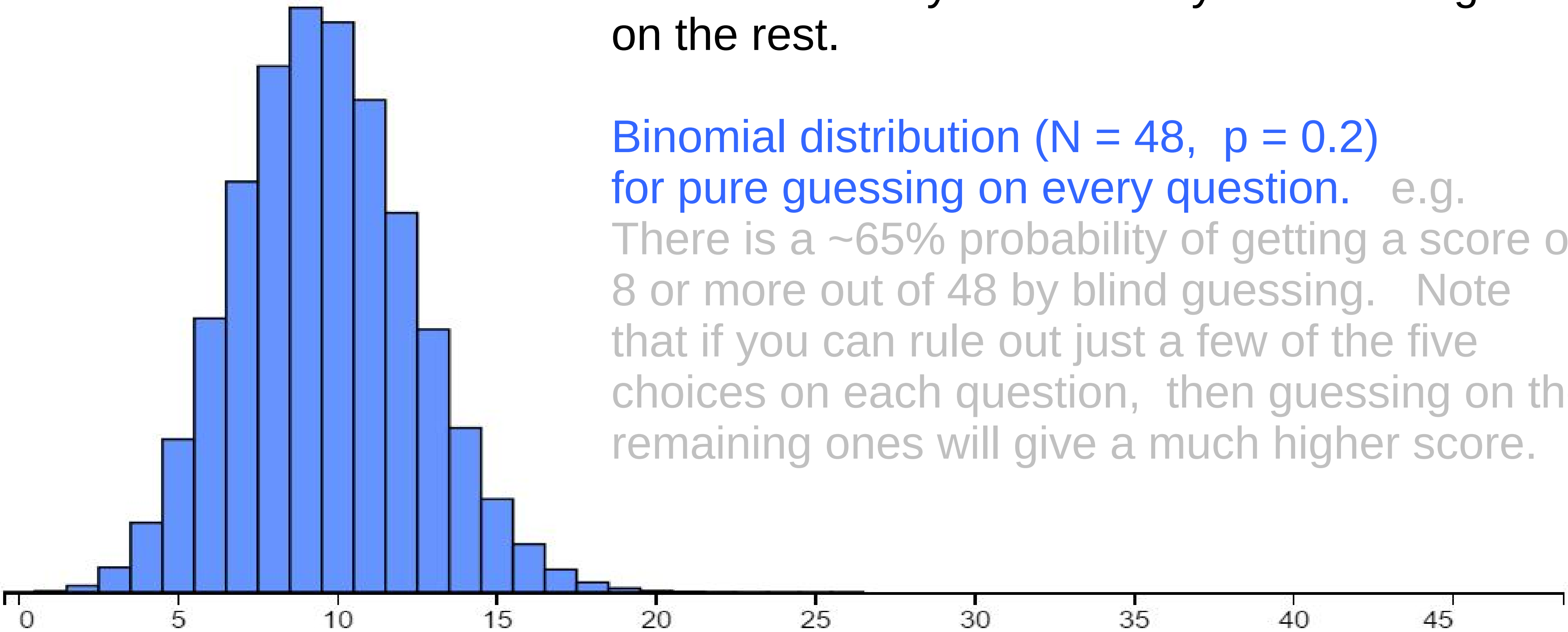
Announcements

- A3 should be graded by weekend

- Final exam is *multiple choice (48 questions).*

Answer all questions. If you are unsure, then rule out as many choices as you can and guess on the rest.

Binomial distribution (N = 48, p = 0.2) for pure guessing on every question. e.g. There is a ~65% probability of getting a score of 8 or more out of 48 by blind guessing. Note that if you can rule out just a few of the five choices on each question, then guessing on the remaining ones will give a much higher score.



I will attempt to use the following method to calculate your exam score. (The rescaling is to compensate for blind guessing.)

$$\text{your score} = \frac{\text{number correct} - 8}{40} * 50$$

e.g. 28 correct answers out of the 48 questions would give a score of 25 / 50 (50%).

If this method gives scores overall that are too low, then I will replace the constant 8 by a smaller one, perhaps even as small as 0 if that's what I need to do to get a roughly B- to B grade average.