Context switch

- Lab0 feedback
- Recap - interrupt
- Context switch

Game submits

- total: 25 submits
- penalty
  - timing penalty
  - naming penalty – 10%
    - only your student number
    - no Chinese or sepcial characters
  - upload to the wrong directory – 10%
- plagiarism is found
  - no tolerance

Due Date & Hard Deadline

- Submits before Due Date will not receive timing penalty.
- Submits after Due Date will receive
  - 50% penalty for the first day
  - 60% penalty for the second day
  - 70% penalty for the third day
- Submits after Hard Deadline will receive 80% penalty.

- Lab0 feedback

Congratulations!

- Most of you have finished the first game in your life!
  - excited? fulfill?
- share them with your friends
  - http://cslab.nju.edu.cn/ics/index.php/Game

Code smell

```
 static inline void
 draw_pixel(int x, int y, int color) {
-        assert(x >= 0 && y >= 0 && x <= SCR_HEIGHT && y <= SCR_WIDTH);
+        assert(x >= 0 && y >= 0 && x < SCR_HEIGHT && y < SCR_WIDTH);
         vmem[(x << 8) + (x << 6) + y] = color;
 }
```

```
 CFLAGS = -m32 -static -MD -std=gnu89 -ggdb \
                  -fno-builtin -fno-stack-protector -fno-omit-frame-pointer \
-                 -Wall -Werror -O2 -I./include
+                 -Wall  -O2 -I./include
```

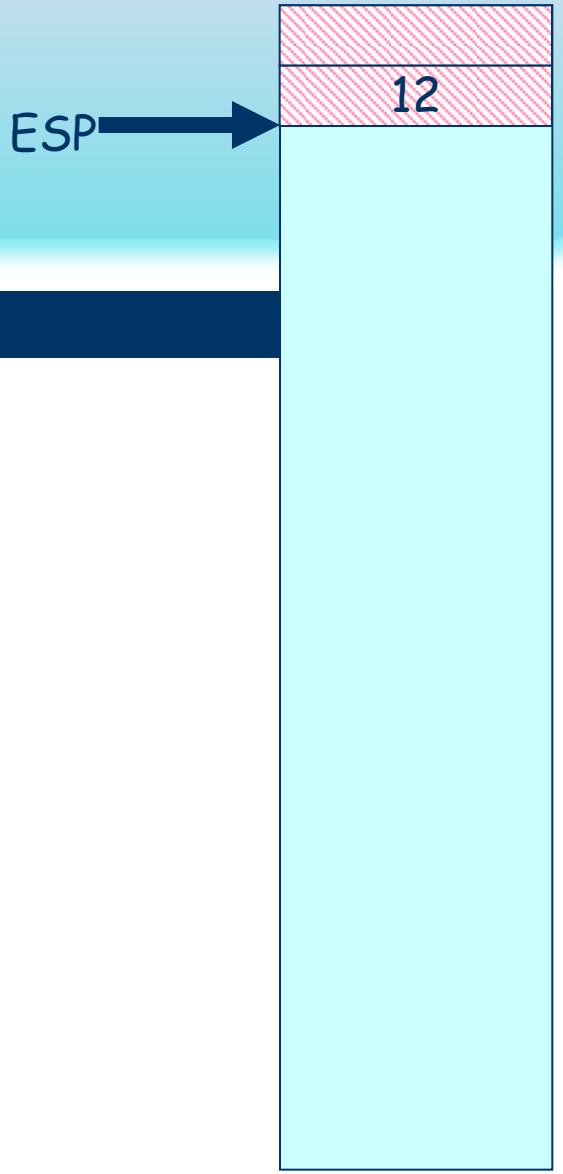
- Why use assert() and Werror?
  - capture faults & errors as soon as possible

More questions

- What are GDT, GDTR, segment descriptor, segment selector, IDT, IDTR, gate descriptor?
- How is address translation performed in Lab0?
- What happen to your game when an interrupt comes?
- How does game.img generate?

- Recap - interrupt

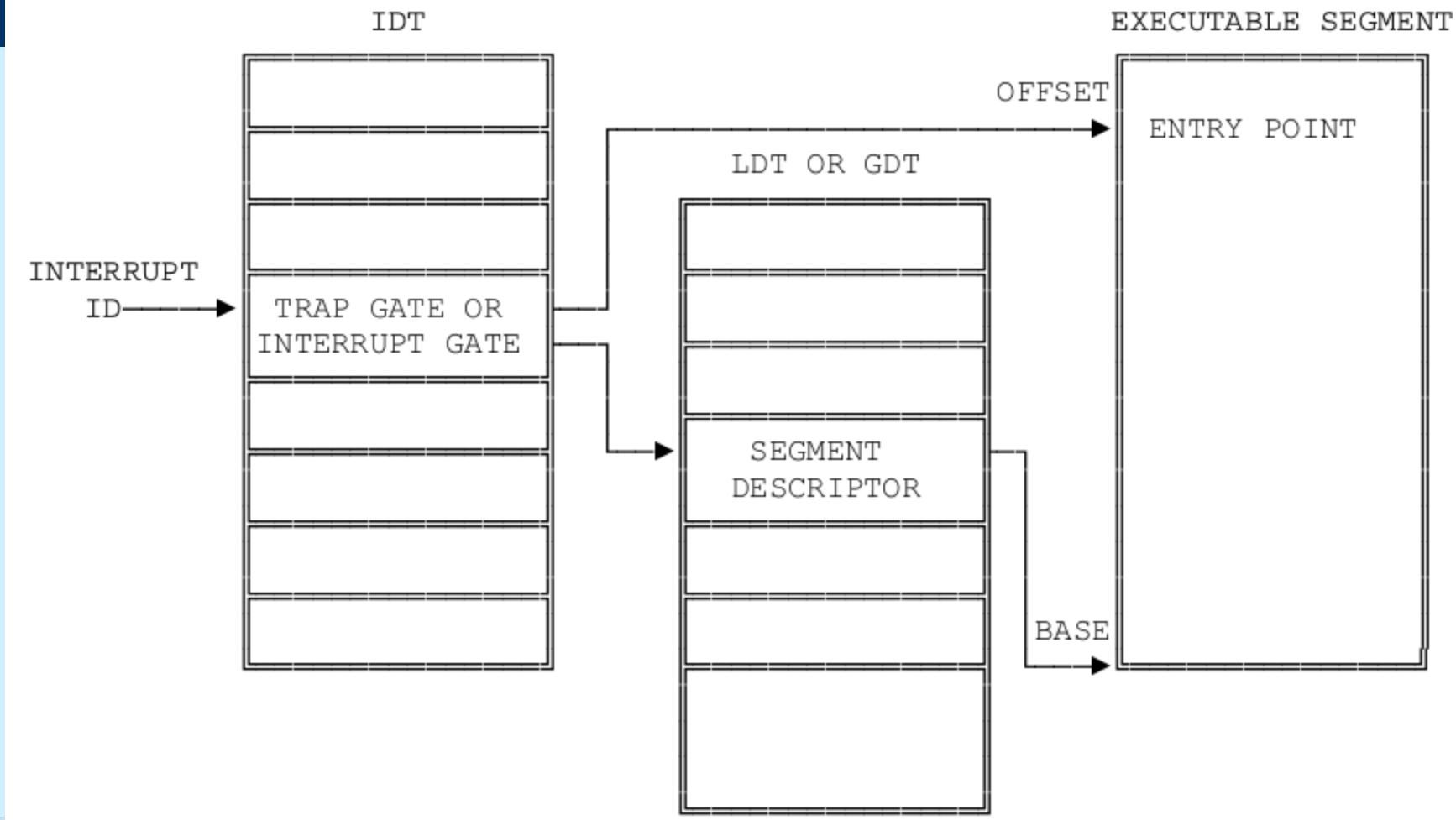
Example



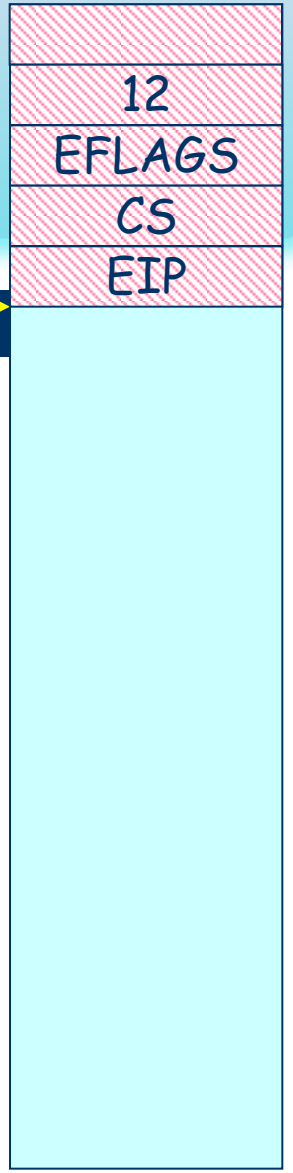
time expires

```
while(1){
        x = (x + 1) % 100;
        if(x == 0){
                printk("a");
        }
}
```

Call the interrupt handler

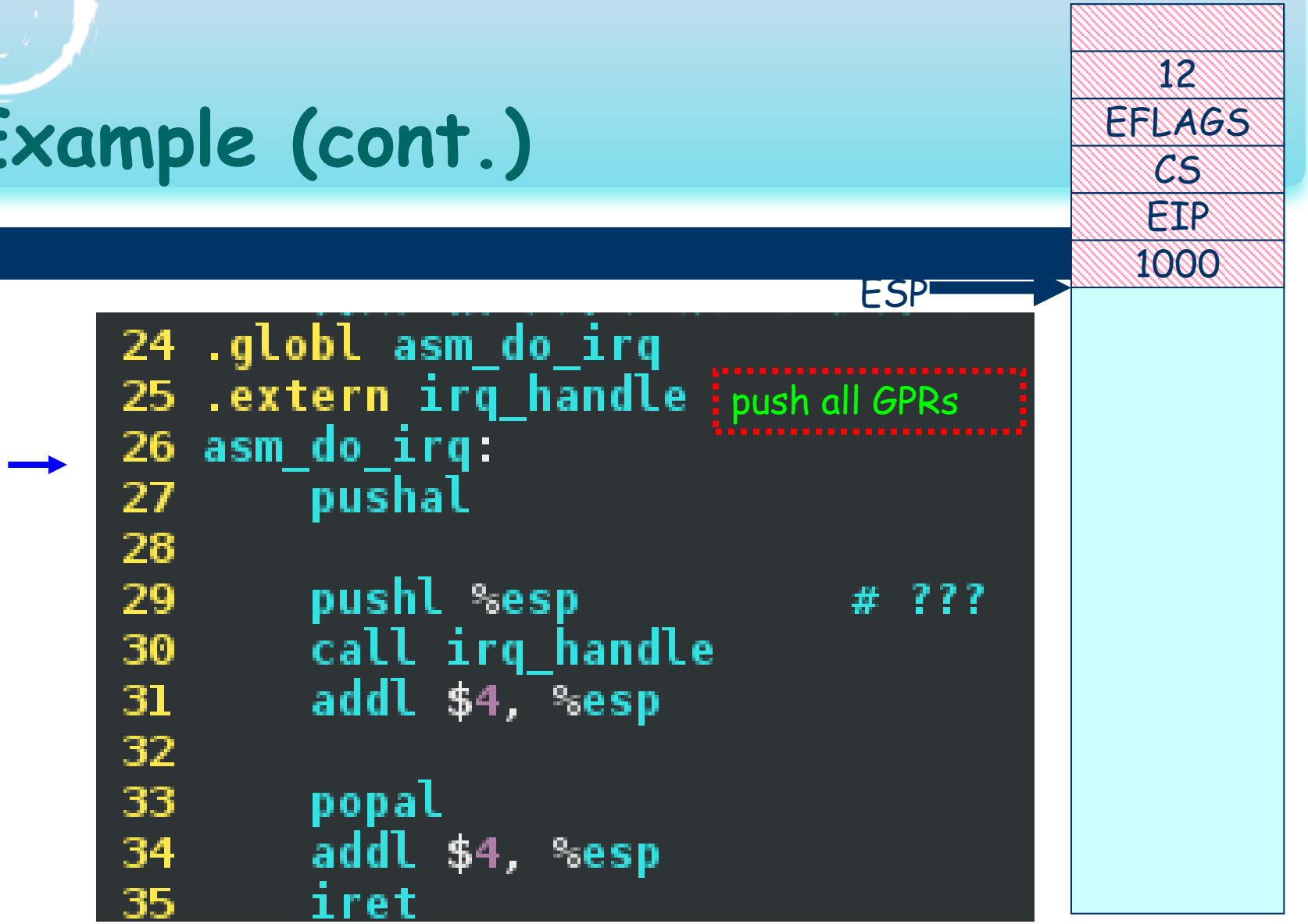


Example (cont.)




```
 1 # 中断和异常处理函数的入口
 2 # 中断处理函数会在IDT中为相应的中断/异常设置处理程序
 3 # 中断/异常的行为参见i386手册
 4 .globl vec0; vec0: pushl $0; jmp asm_do_irq
 5 .globl vec1; vec1: pushl $1; jmp asm_do_irq
 6 .globl vec2; vec2: pushl $2; jmp asm_do_irq
 7 .globl vec3; vec3: pushl $3; jmp asm_do_irq
 8 .globl vec4; vec4: pushl $4; jmp asm_do_irq
 9 .globl vec5; vec5: pushl $5; jmp asm_do_irq
10 .globl vec6; vec6: pushl $6; jmp asm_do_irq
11 .globl vec7; vec7: pushl $7; jmp asm_do_irq
12 .globl vec8; vec8: pushl $8; jmp asm_do_irq
13 .globl vec9; vec9: pushl $9; jmp asm_do_irq
14 .globl vec10; vec10: pushl $10; jmp asm_do_irq
15 .globl vec11; vec11: pushl $11; jmp asm_do_irq
16 .globl vec12; vec12: pushl $12; jmp asm_do_irq
17 .globl vec13; vec13: pushl $13; jmp asm_do_irq
18
19 .globl irq0; irq0: pushl $1000; jmp asm_do_irq
20
21 .globl irq_empty; irq_empty: pushl $-1; jmp asm_do_irq
```

Example (cont.)



Example (cont.)

```
24 .globl asm_do_irq
25 .extern irq_handle
26 asm_do_irq:
27     pushal
28
29     pushl %esp                # ???
30     call irq_handle
31     addl $4, %esp
32
33     popal
34     addl $4, %esp
35     iret
```


trap frame

ESP

Example (cont.)

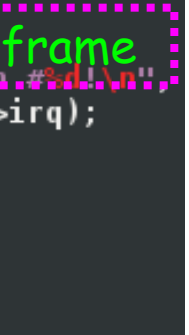
```
24 .globl asm_do_irq
25 .extern irq_handle
26 asm_do_irq:
27     pushal
28
29     pushl %esp              # ???
30     call irq_handle
31     addl $4, %esp
32
33     popal
34     addl $4, %esp
35     iret
```

trap frame

ESP

Example (cont.)

```
18 void
19 irq_handle(struct TrapFrame *tf) {
20     if(tf->irq < 1000) {
21         if(tf->irq == -1) {
22             printk("%s, %d: Unhandled exception!\n",
23                     __FUNCTION__, __LINE__);
24         }
25         else {
26             printk("%s, %d: Unexpected exception #%d!\n",
27                     __FUNCTION__, __LINE__, tf->irq);
28         }
29         assert(0);
30     }
31
32     if (tf->irq == 1000) {
33         do_timer();
34     } else if (tf->irq == 1001) {
35         uint32_t code = in_byte(0x60);
36         uint32_t val = in_byte(0x61);
37         out_byte(0x61, val | 0x80);
38         out_byte(0x61, val);
39
40         printk("%s, %d: key code = %x\n",
41                 __FUNCTION__, __LINE__, code);
42         do_keyboard(code);
43     } else {
44         assert(0);
45     }
46 }
```



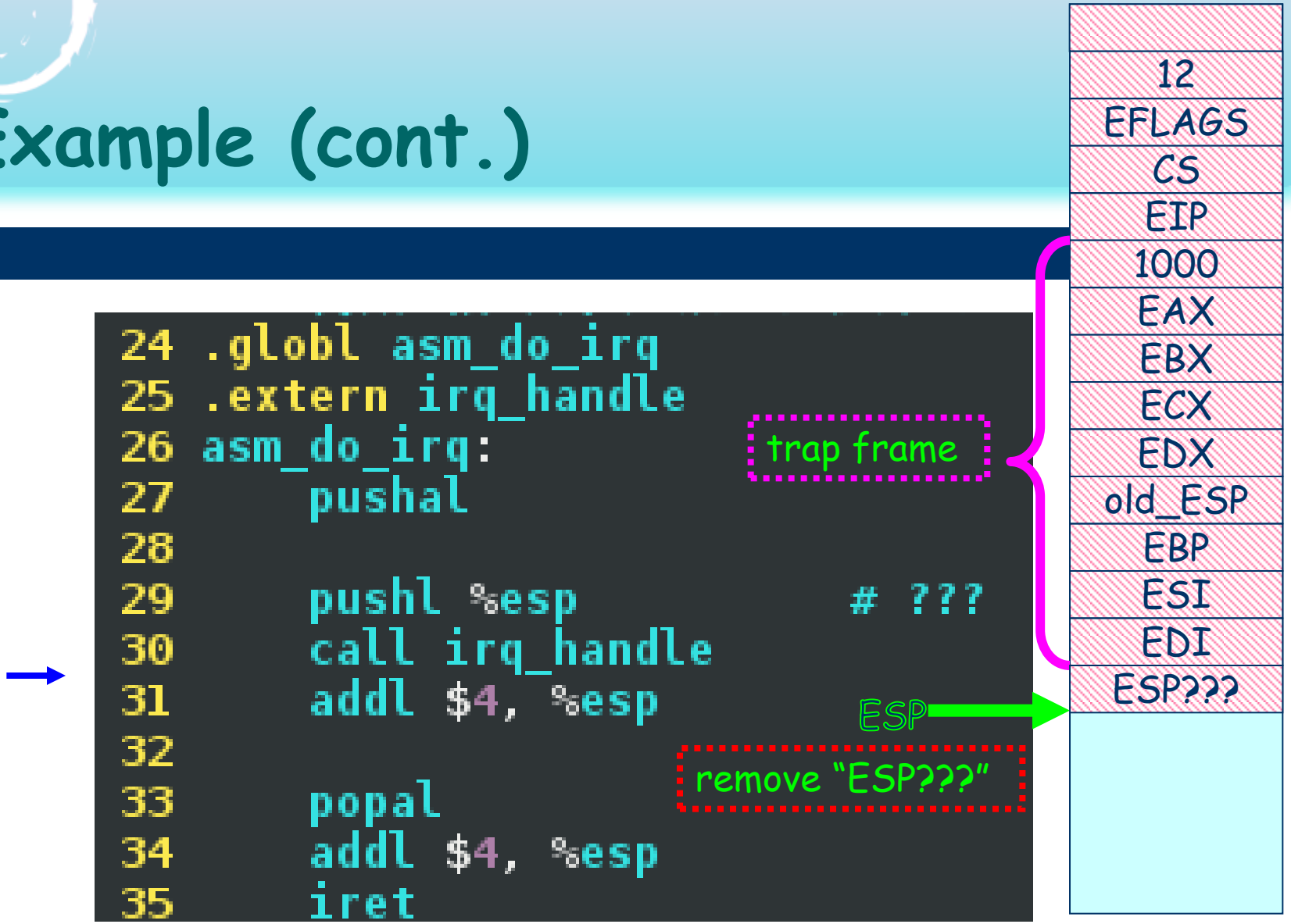
Example (cont.)

```
24 .globl asm_do_irq
25 .extern irq_handle
26 asm_do_irq:
27     pushal
28
29     pushl %esp              # ???
30     call irq_handle
31     addl $4, %esp
32
33     popal
34     addl $4, %esp
35     iret
```

trap frame

ESP

remove "ESP???"



Example (cont.)

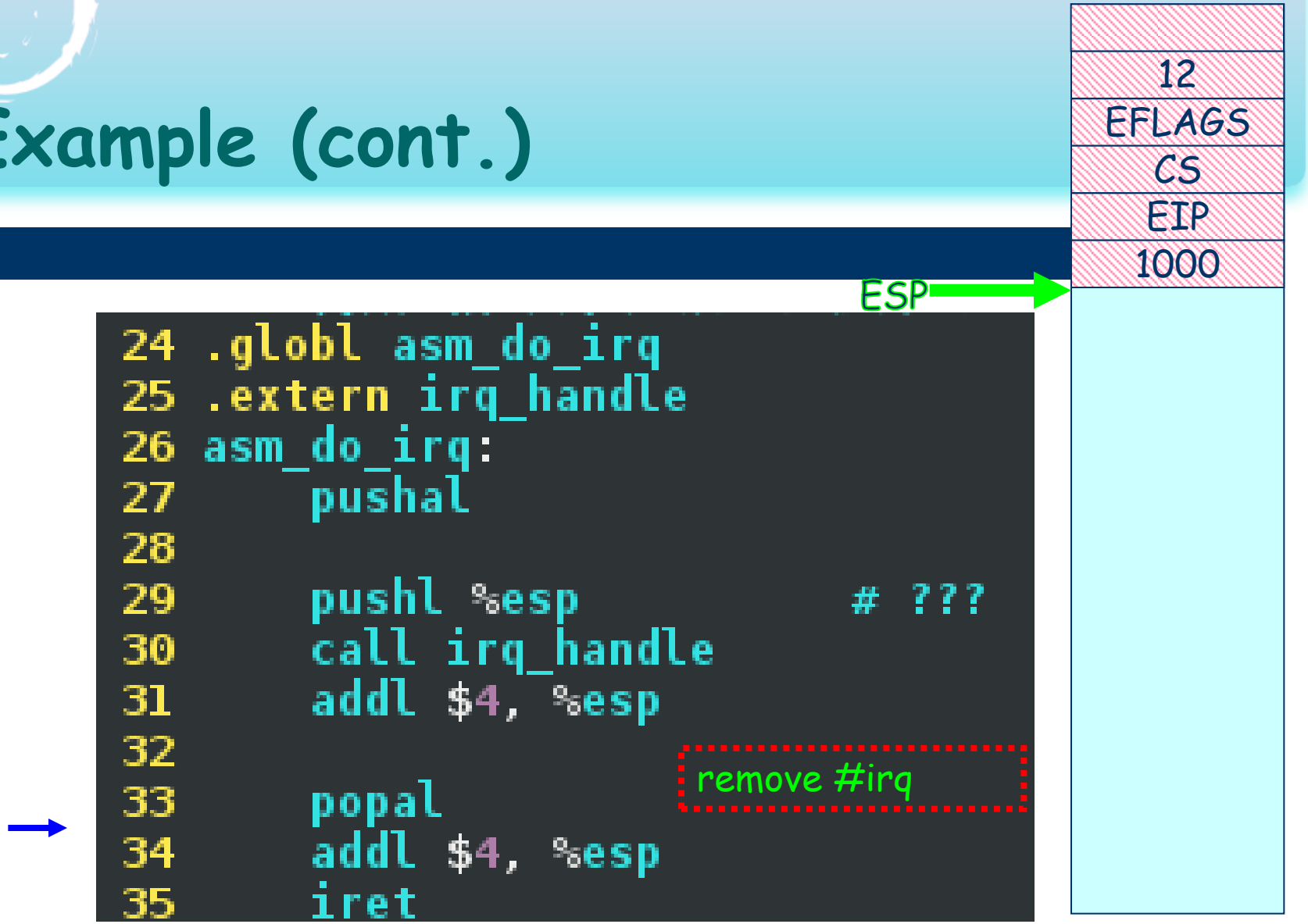
```
24 .globl asm_do_irq
25 .extern irq_handle
26 asm_do_irq:
27     pushal
28
29     pushl %esp                # ???
30     call irq_handle
31     addl $4, %esp
32
33     popal
34     addl $4, %esp
35     iret
```

trap frame

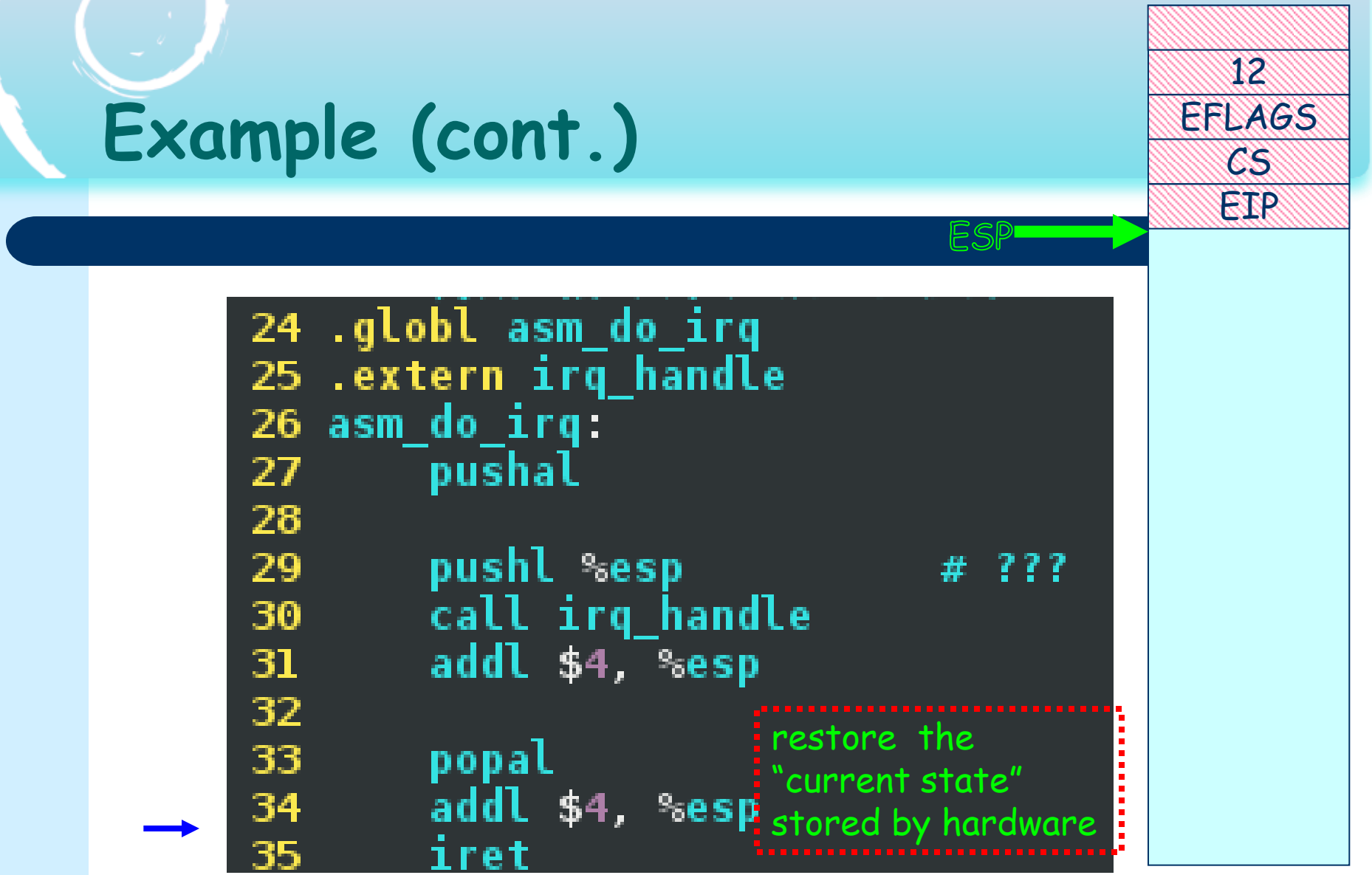
ESP

restore GPRs

Example (cont.)



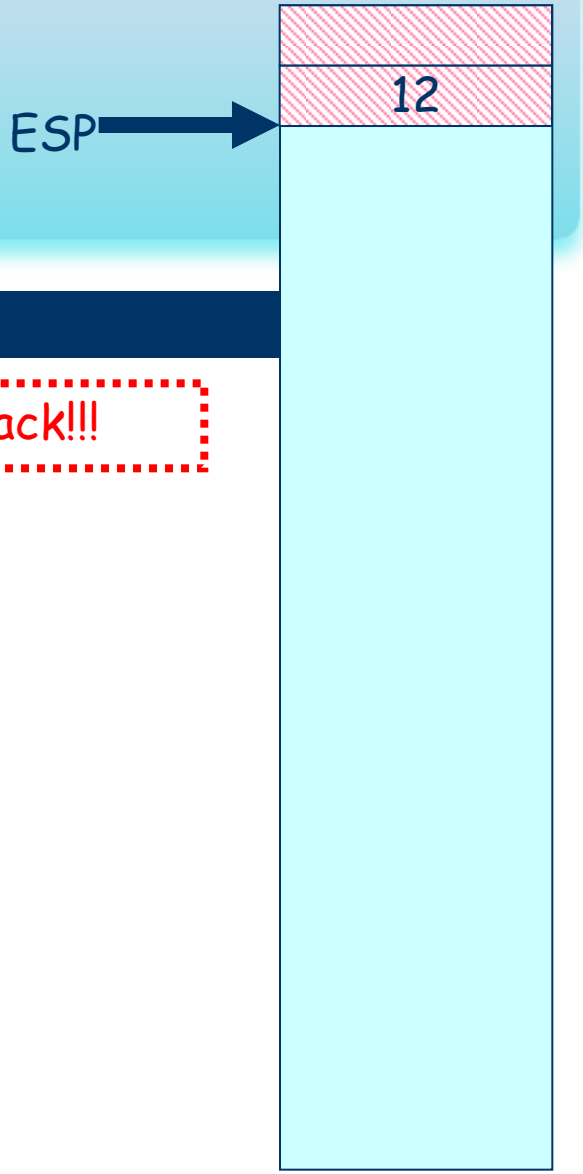
Example (cont.)




```
24 .globl asm_do_irq
25 .extern irq_handle
26 asm_do_irq:
27     pushal
28
29     pushl %esp              # ???
30     call irq_handle
31     addl $4, %esp
32
33     popal
34     addl $4, %esp
35     iret
```

restore the "current state" stored by hardware

Example (cont.)



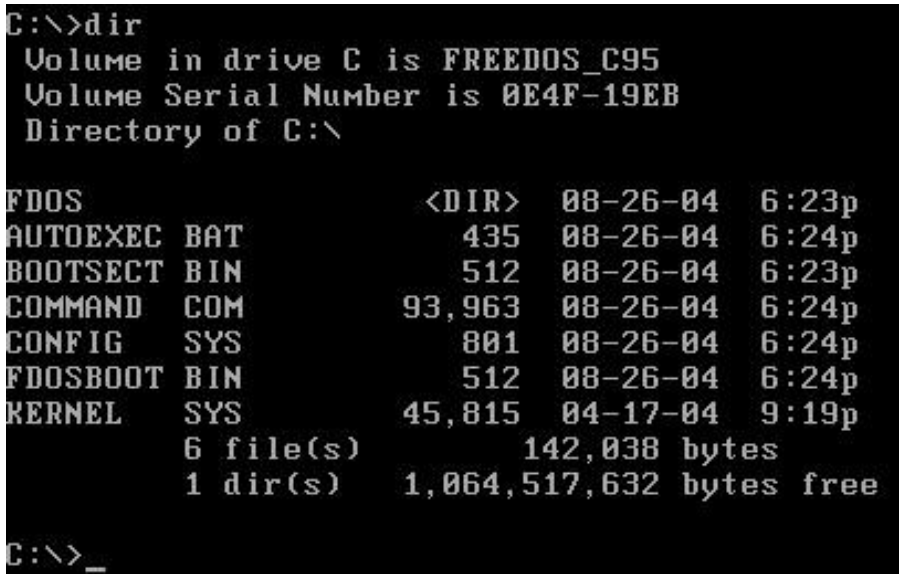
we are back!!!

```
while(1){
        x = (x + 1) % 100;
        if(x == 0){
                printk("a");
        }
}
```

- Context switch

Unitasking

- Only one program executing at a time.
  - occupies all resources (CPU, memory, I/O…)
- To let B run, A must exit first.



Multitasking

- Two or more programs can run "simultaneously"
  - resources are shared
- process: a running program

- fact: for unicore CPU, only one process can run at any point in time

Solution

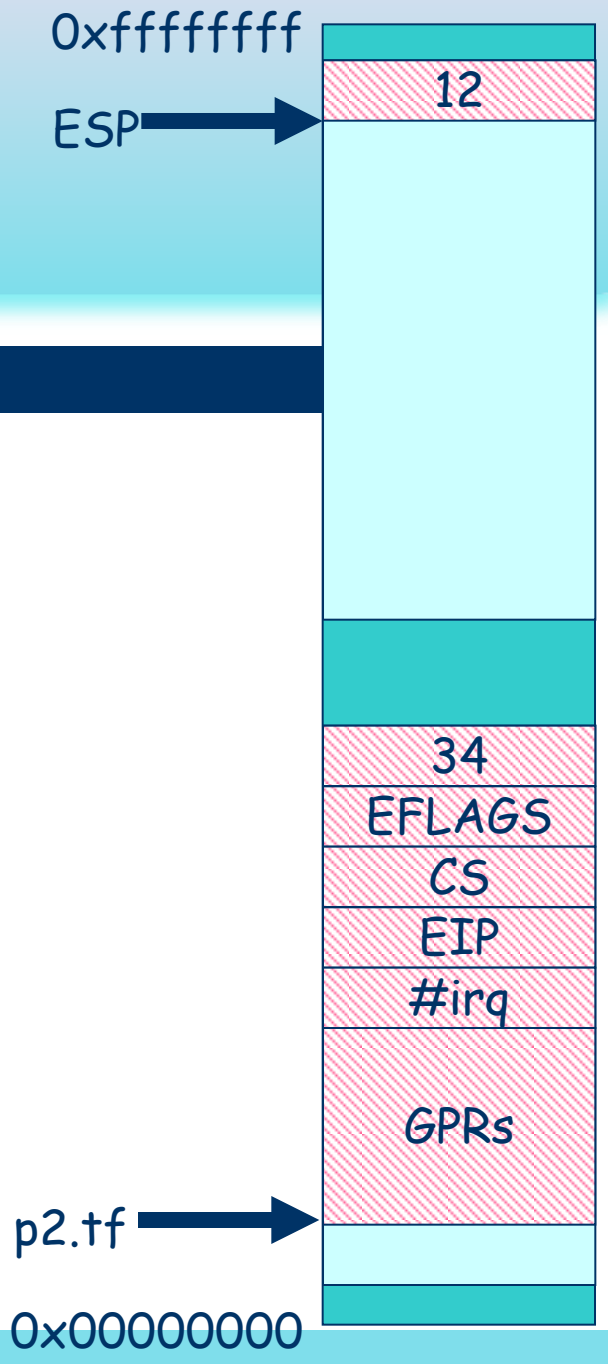
- Allow different processes use CPU in turn.
- A running process may relinquish CPU
  - actively
    - wait for I/O
    - sleep
  - passively
    - time's up

Context switch

- the action of swapping the current process out of CPU, and let the next process run
  - save current state
  - choose another process
  - switch to the new process
  - restore current state
- Cuttent state is saved and restored by the interrupt procedure!!!
  - we do not need to manually implement them

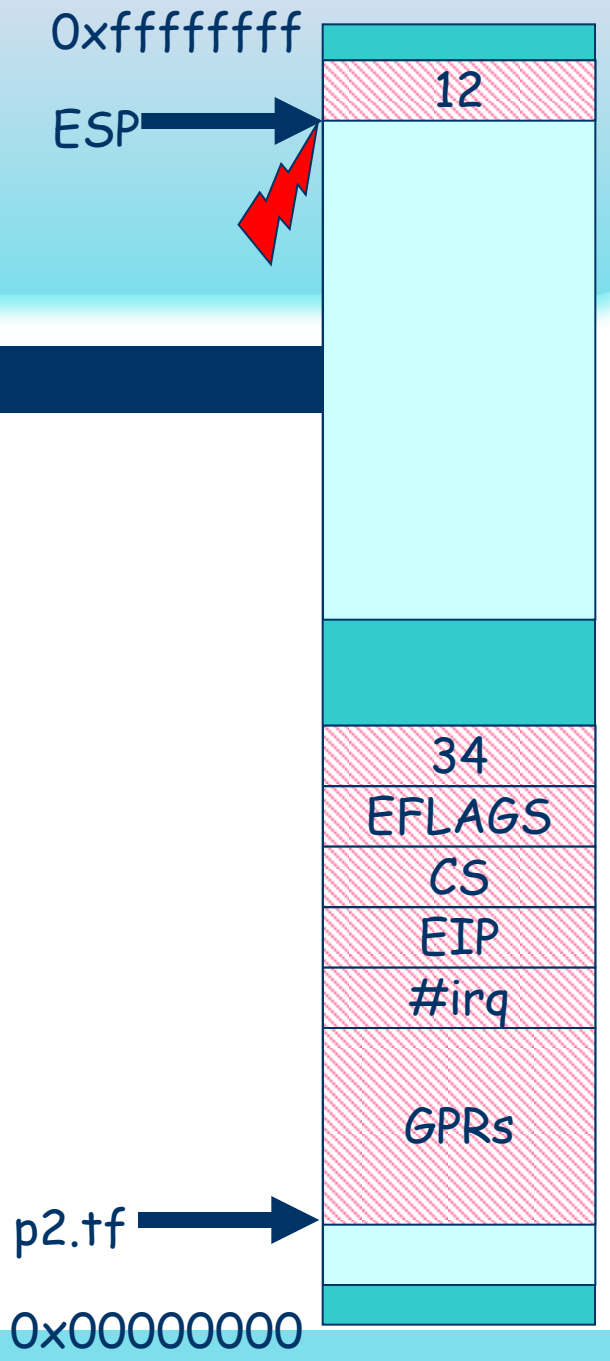
Context switch (cont.)

- process #1 is running…



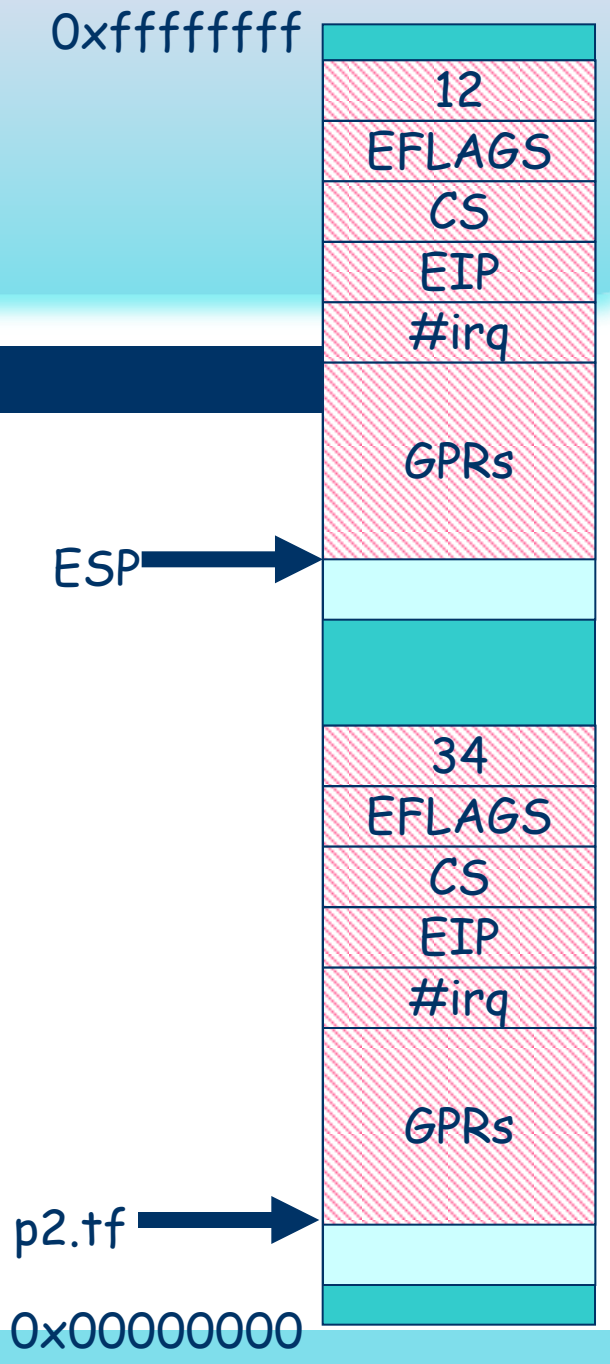
Context switch (cont.)

- process #1 is running…
- time's up



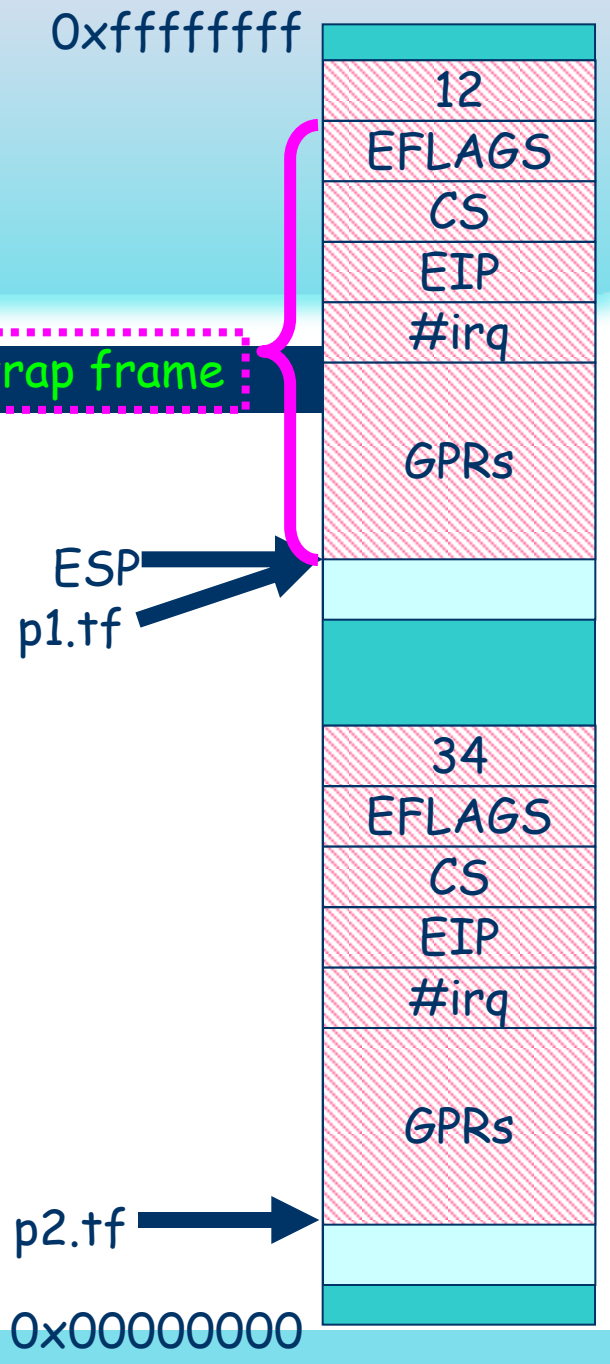
Context switch (cont.)

- process #1 is running…
- time's up
- save current state



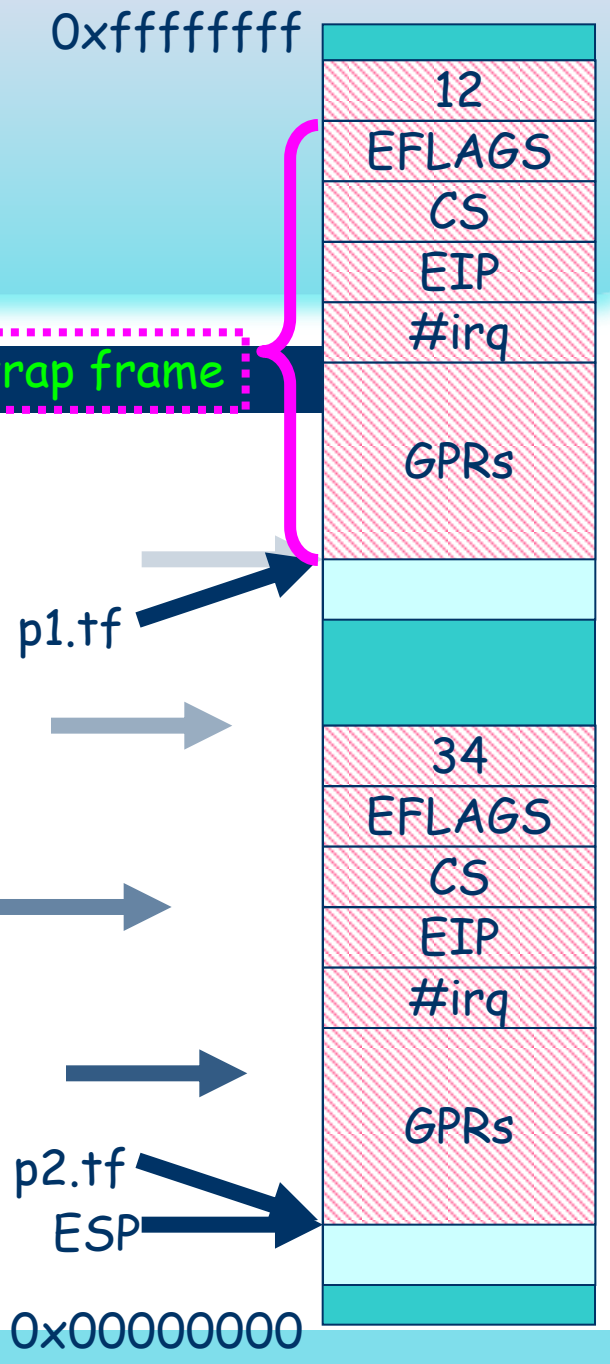
Context switch (cont.)

- process #1 is running…
- time's up
- save current state
- save the top of #1's stack



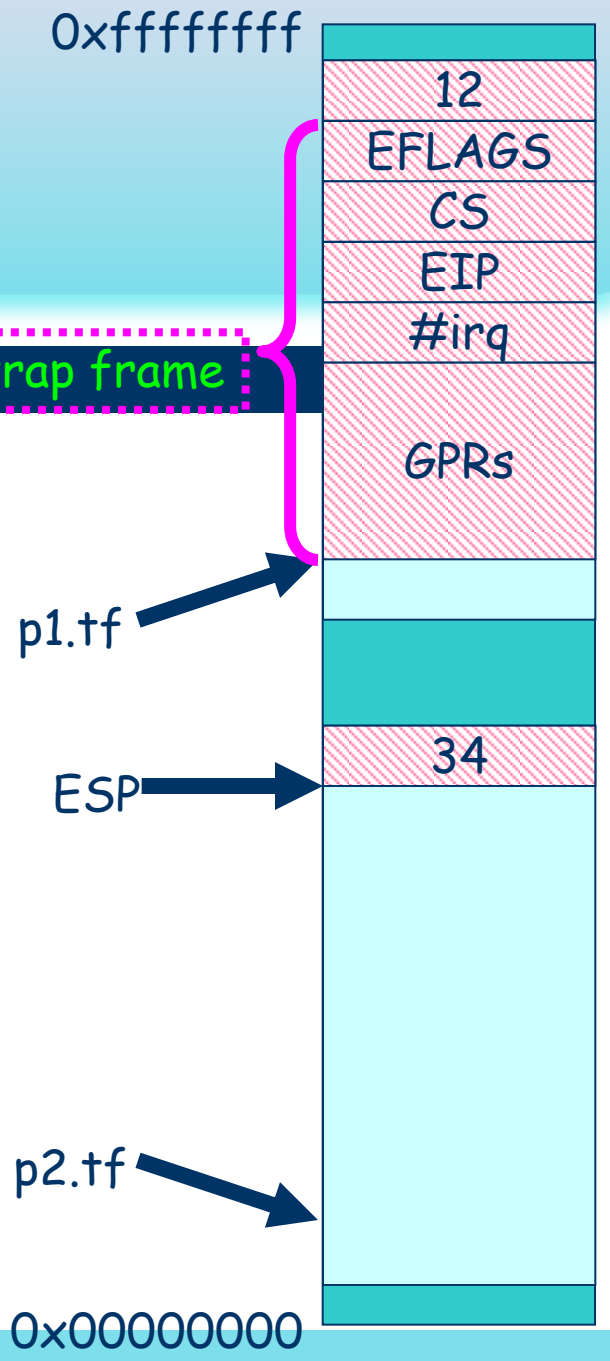
Context switch (cont.)

- process #1 is running…
- time's up
- save current state
- save the top of #1's stack
- switch %ESP to process #2



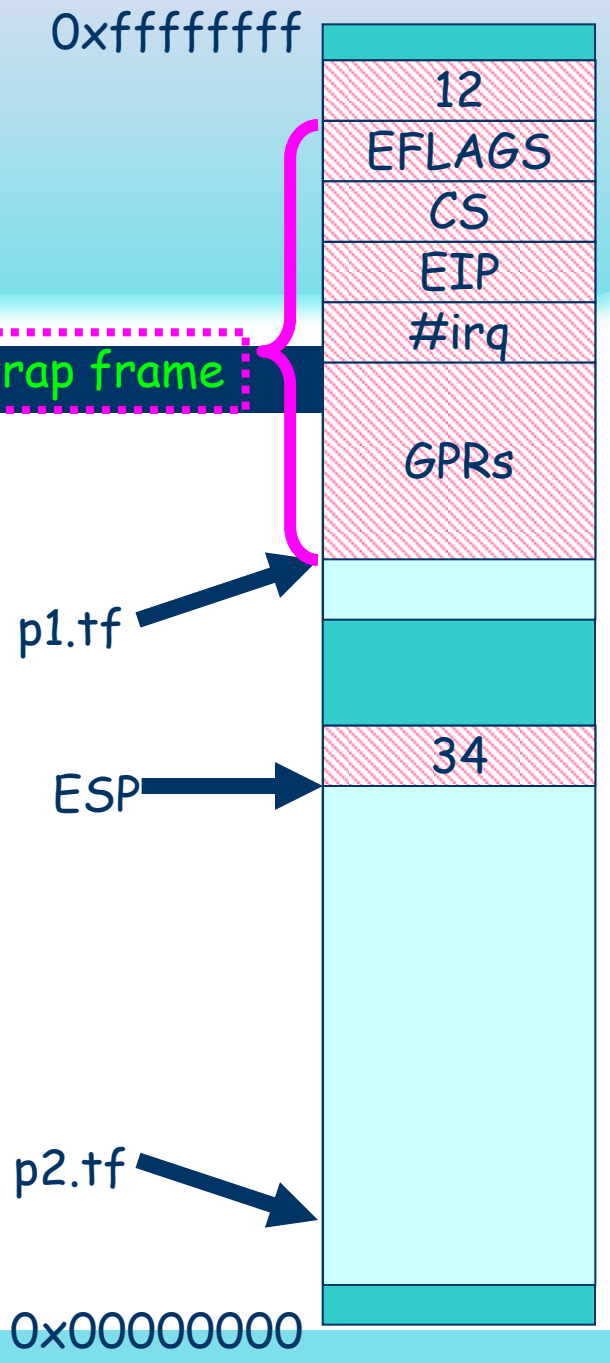
Context switch (cont.)

- process #1 is running…
- time's up
- save current state
- save the top of #1's stack
- switch %ESP to process #2
- restore current state



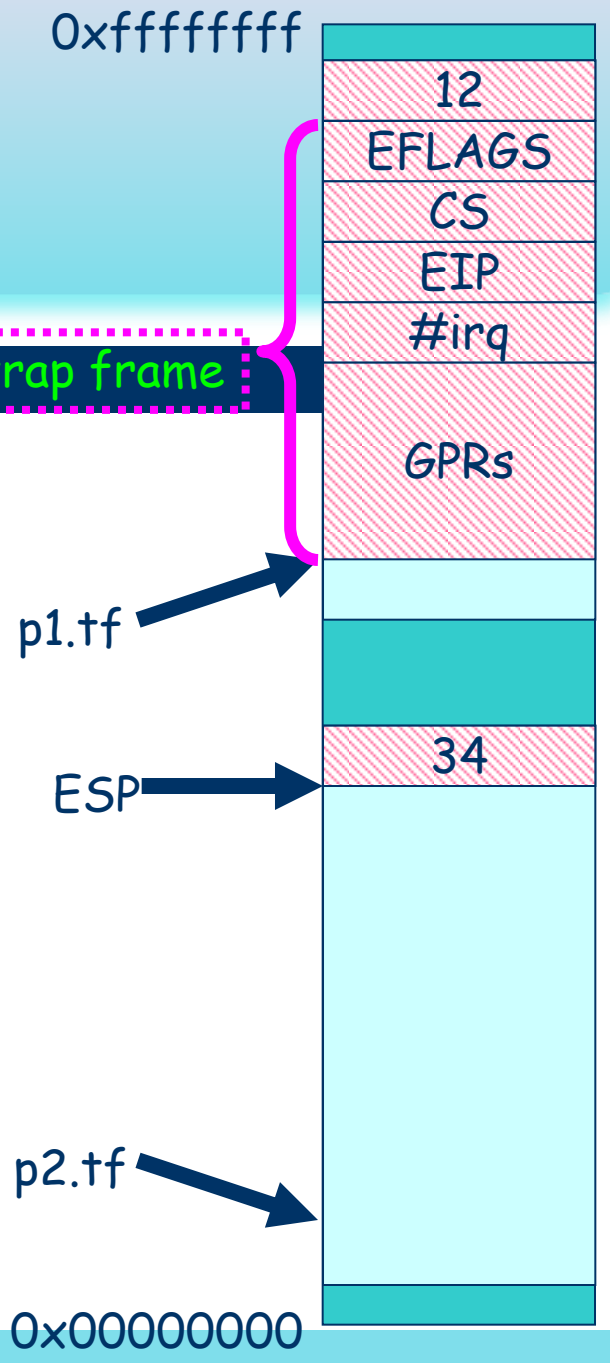
Context switch (cont.)

- process #1 is running…
- time's up
- save current state
- save the top of #1's stack
- switch %ESP to process #2
- restore current state
- process #2 is running…



Context switch (cont.)

- process #1 is running…
- time's up
- save current state
- save the top of #1's stack
- switch %ESP to process #2
- restore current state
- process #2 is running…



How to present a process?

- process control block
  - contains information for process management
- Currently, two members are sufficient.
  - stack
  - context information

```
#define KSTACK_SIZE 4096
struct PCB {
    void *tf;
    uint8_t kstack[KSTACK_SIZE];
} *current;
```

- *current* points to the running process

Remaining problems

- How to choose the next process?
  - schedule policy
    - round robin, multi-level queue, ...
- How to switch to the new process?
  - hint: all information needed is in PCB
- How to create new process?

Process?

- A process has its own resource.
  - address space
    - code, data, stack
  - semaphore, file, ...
- By now we can only create kernel threads.
  - share codes and data with kernel
  - has it own stack
- Current information in PCB is not enough to identify different address spaces.
  - You will implement real processes in the future.

Create a kernel thread

- write a test function
- allocate a PCB
- initialize the PCB
  - stack
  - context information
    - you should initialize the "current state" correctly
    - pay attention to the IF bit of EFLAGS
- add the thread into the ready queue
  - feel free to use ListHead

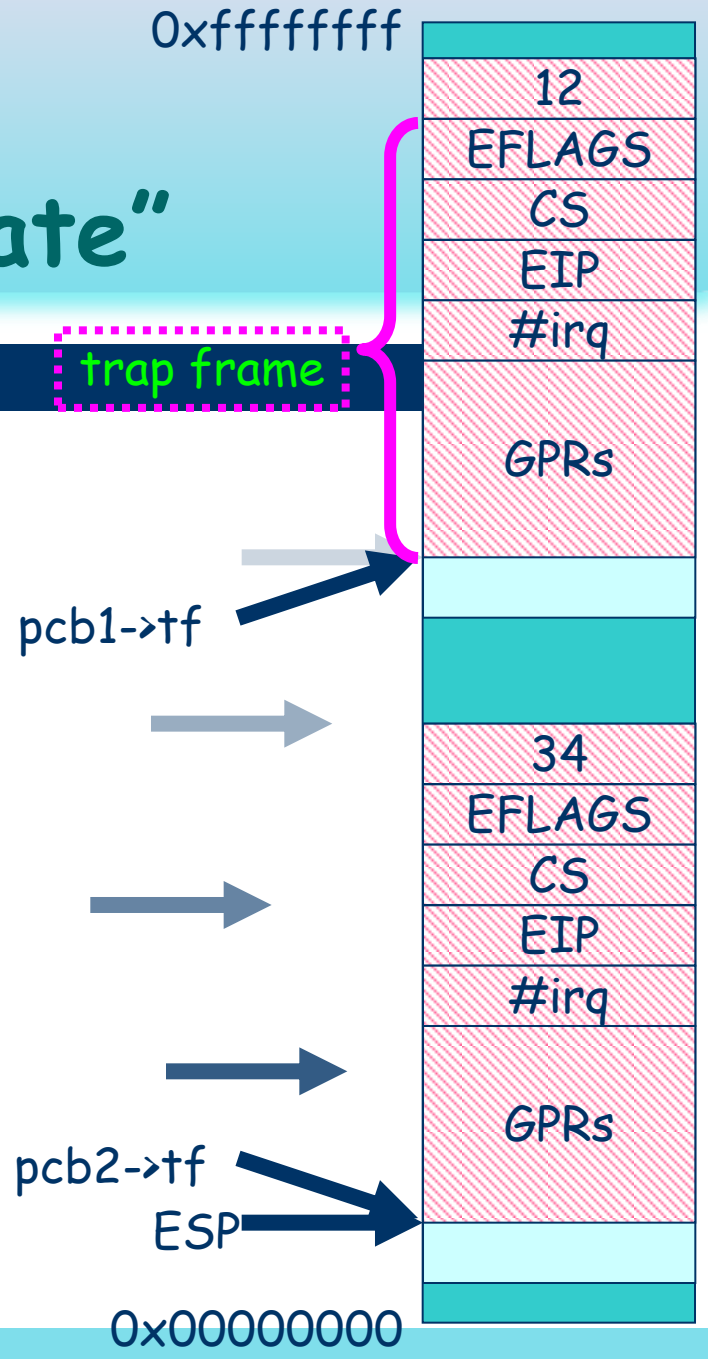
Test functions

```
void A () {
    int x = 0;
    while(1) {
        if(x % 100000 == 0) {printk("a");}
        x ++;
    }
}
void B () {
    int x = 0;
    while(1) {
        if(x % 100000 == 0) {printk("b");}
        x ++;
    }
}
```

Initialize "current state"

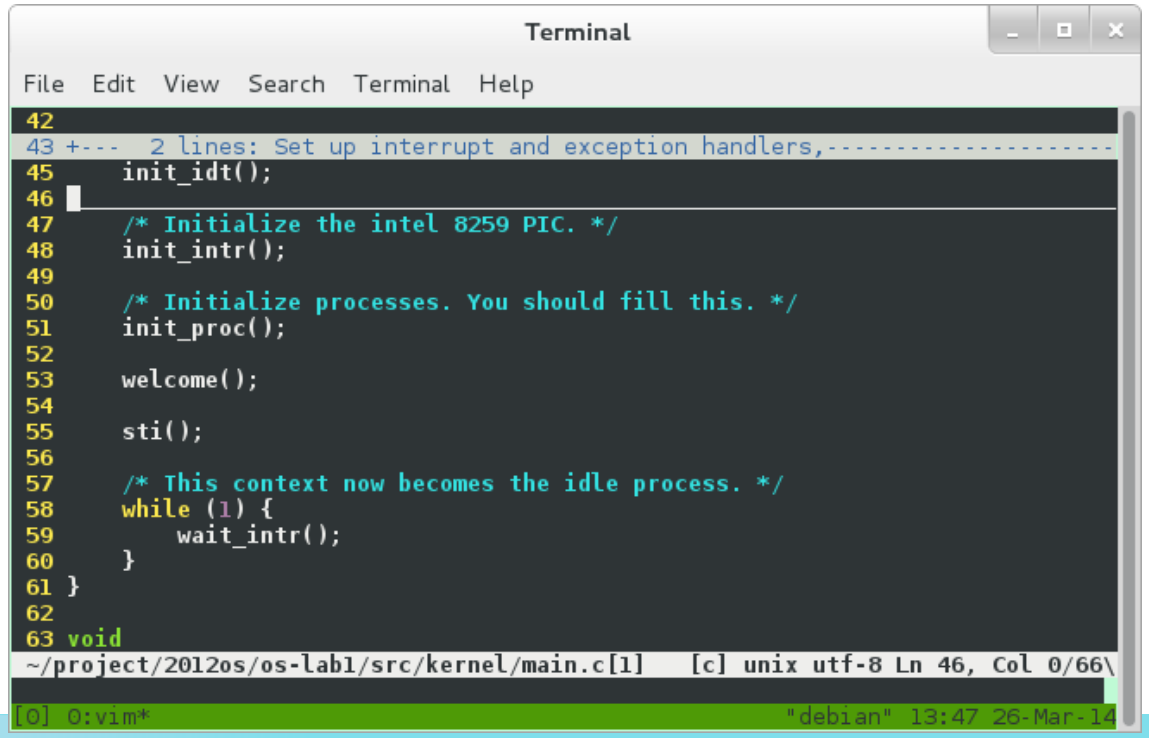
- GPRs = ?
  - GPRs.eps = ?
- #irq = ?
- EIP = ?
- CS = ?
- EFLAGS = ?

- new fields in Lab1
  - error code = ?
  - DS, ES, GS, FS = ?



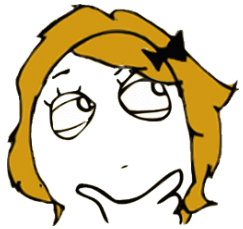
IDLE thread

- allocate a PCB to it
  - it will be scheduled when no other thread is ready



NOTICE

- you can not return from a thread
  - or you will get an exception like



Terminal

File Edit View Search Terminal Help

```
gcc -c -m32 start.S
gcc -c -m32 -O1 main.c
ld -e start -Ttext 0x7C00 -m elf_i386 start.o main.o
objcopy -S -O binary -j .text a.out bootblock
perl genboot.pl bootblock
OK: boot block is 437 bytes (max 510)
make[1]: Leaving directory `/home/yzh/project/2012os/os-lab1/boot'
cat boot/bootblock kernel > disk.img
qemu-system-i386 -serial stdio disk.img
Hello, OS World!

system panic in function "irq_handle", line 49, file "src/kernel/irq/irq_handle.
c":
Unexpected exception #14
Hint: The machine is always right! For more details about exception #14, see
 _____   ___   ___    __ __                          _
(_)___ \ / _ \ / _ \  |  \/  |                        | |
  ___) | (_) | (_) | | \  / | __ _ _ __ _ __ _   _  __| |
 |__ < > _ <| | | | | |\/| |/ _` | '__| '__| | | |/ _` |
 |___) | (_) | (_) | | |  | | (_| | |  | |  | |_| | (_| |
 |____/ \___/ \___/  |_|  |_|\__,_|_|  |_|  \__,_|\__,_|
```

[0] 0:make* "debian" 13:44 26-Mar-14

Summary

- context switch = interrupt driven stack switch
- new problems
  - thread management
    - create, destroy, sleep, wakeup
    - schedule
  - resource management
    - memory, files…
  - memory protection
  - critical area
  - IPC

Lab1 is partially out!

- the first stage
  - implement context switch
  - create 2 kernel thread to print “a” and “b” in turn
  - this is what Linux kernel did at the beginning
- Have fun!