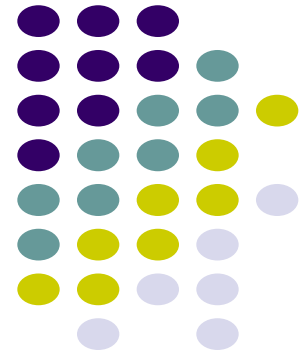


Interrupts

ECE 469, Feb 08

Aravind Machiry



Recap: OS Process API



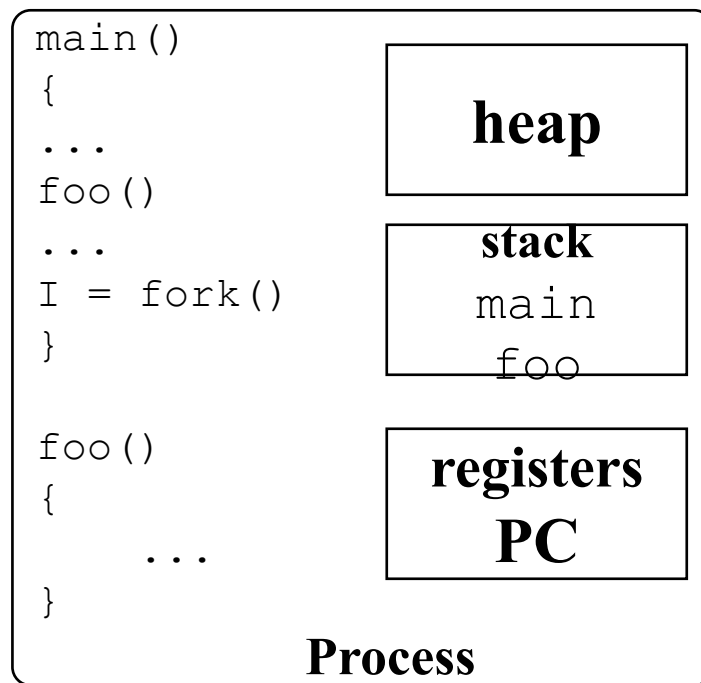
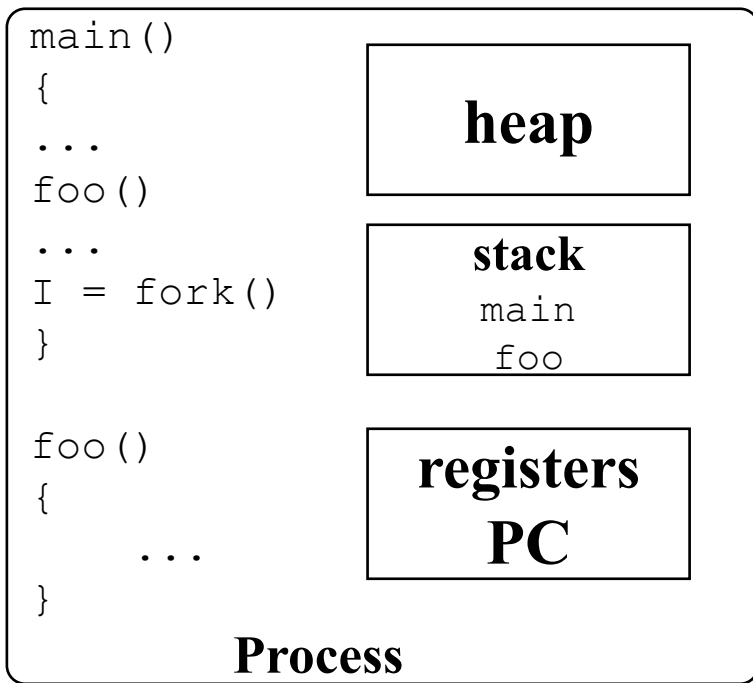
- 4 system calls related to process creation/termination:
 - Process Creation:
 - fork/clone – create a copy of this process
 - exec – replace this process with this program
 - Wait for completion:
 - wait – wait for child process to finish
 - Terminate a process:
 - kill - send a signal (to terminate) a process

Recap: fork



fork causes OS creates a copy of the calling process:

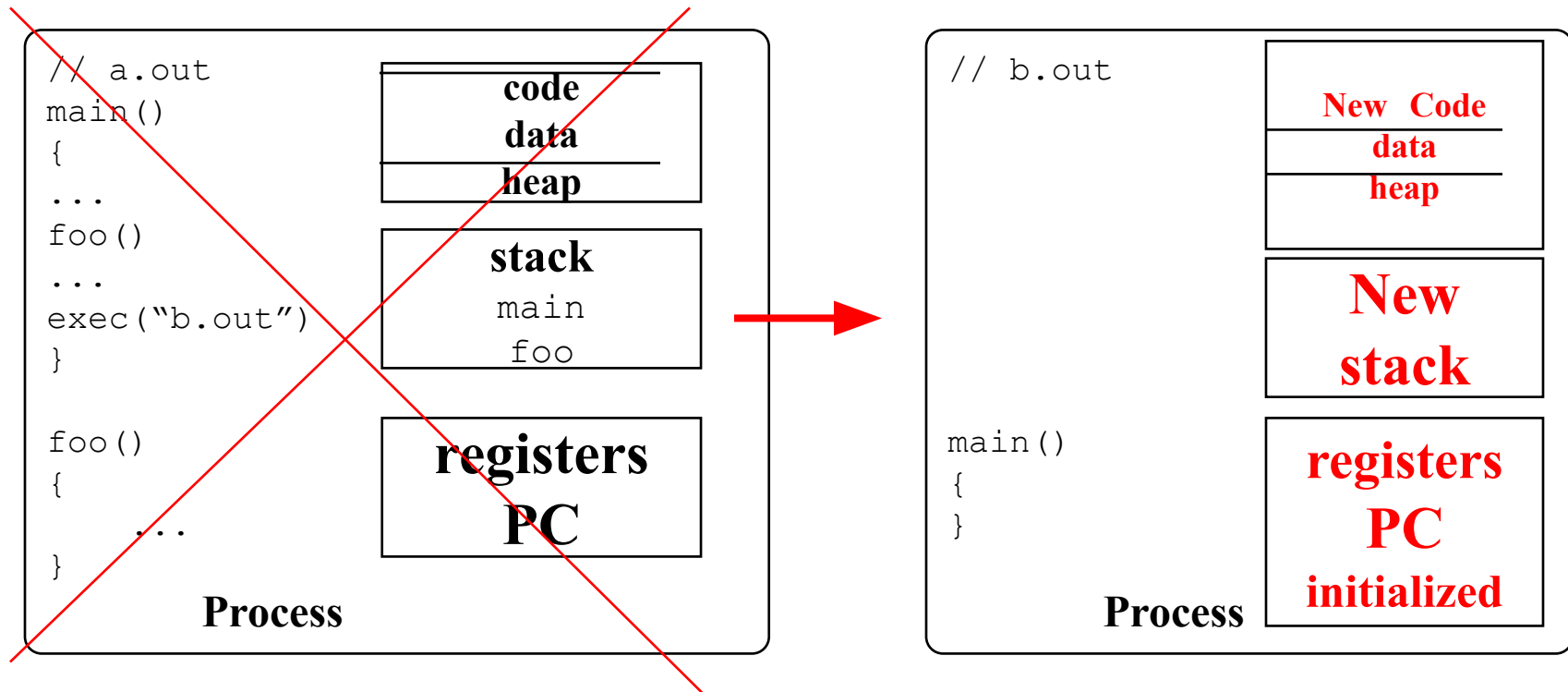
- Why?
- How can we disambiguate between new process and the calling process?



Recap: exec



Replaces current process with the content from new program.



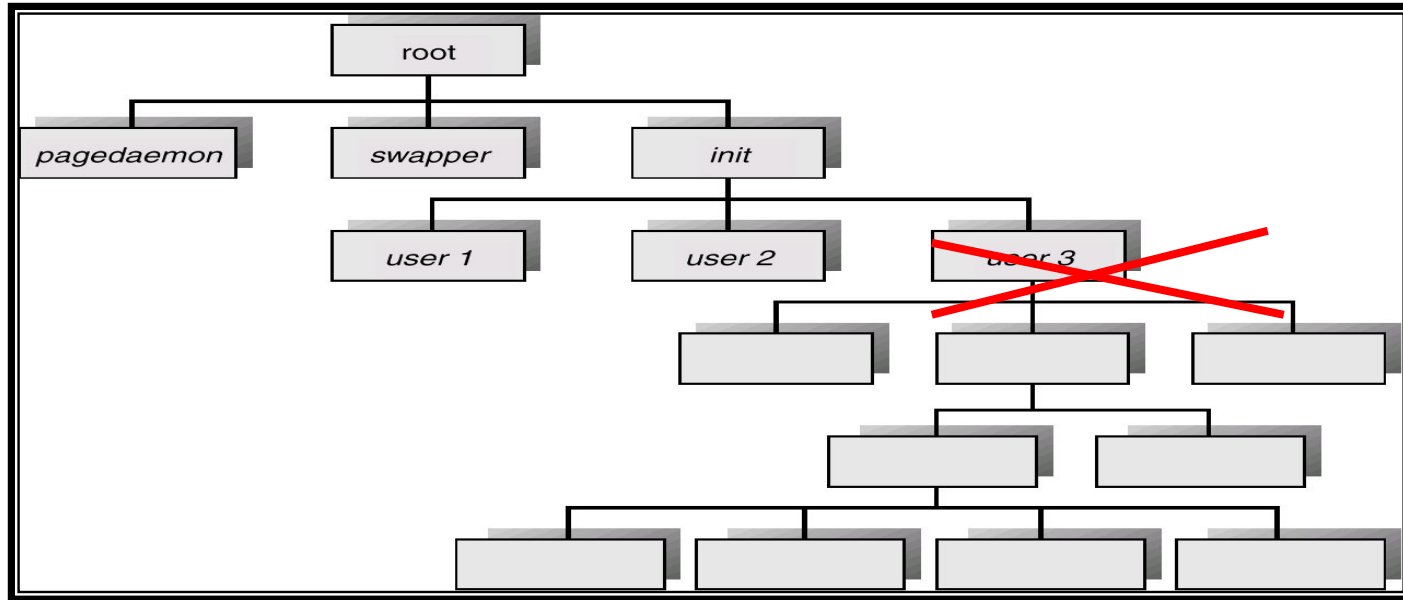
Recap: wait

wait for a child process to finish



Recap: wait

What happens when the parent process dies? what happens to child process?

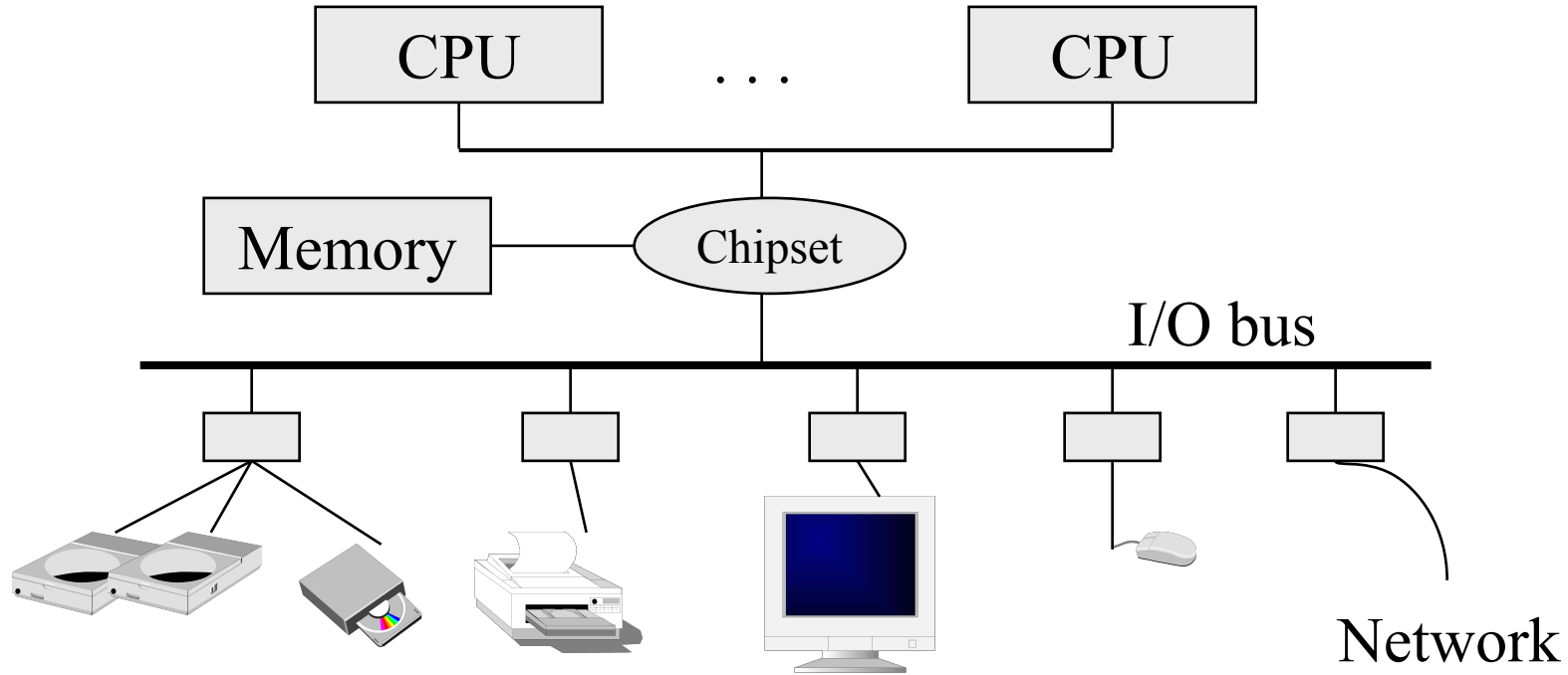


Recap: How our shell works?

- Fork/exec



Handling Hardware / unexpected events



How to handle I/O from peripherals?



- Assume mail delivery
- Poll:
 - Checking for events at regular intervals
 - Checking mailbox daily
- Interrupt
 - Get explicitly notified
 - Secretary notifying you
- Which one is better?
 - Simple (inefficient) v/s Complex (efficient)

Interrupts

- Hardware Interrupts
- Software Interrupts



Hardware Interrupts



- A way of hardware interacting with CPU
- Example: a network device
 - NIC: “Hey, CPU, I have a packet received for the OS, so please wake up the OS to handle the data”
 - CPU: call the interrupt handler for network device in ring 0 (set by the OS)
- Asynchronous (can happen at any time of execution)
 - It’s a request from a hardware, so it comes at any time of CPU’s execution
- Read
 - https://en.wikipedia.org/wiki/Intel_8259
 - https://en.wikipedia.org/wiki/Advanced_Programmable Interrupt_Controller

Software Interrupts / exceptions



- A software mean to run code in ring 0 (e.g., int \$0x30)
 - Telling CPU that "Please run the interrupt handler at 0x30"
- Synchronous (caused by running an instruction, e.g., int \$0x30)
- System call
 - int \$0x30 □ system call in JOS

Types of exceptions

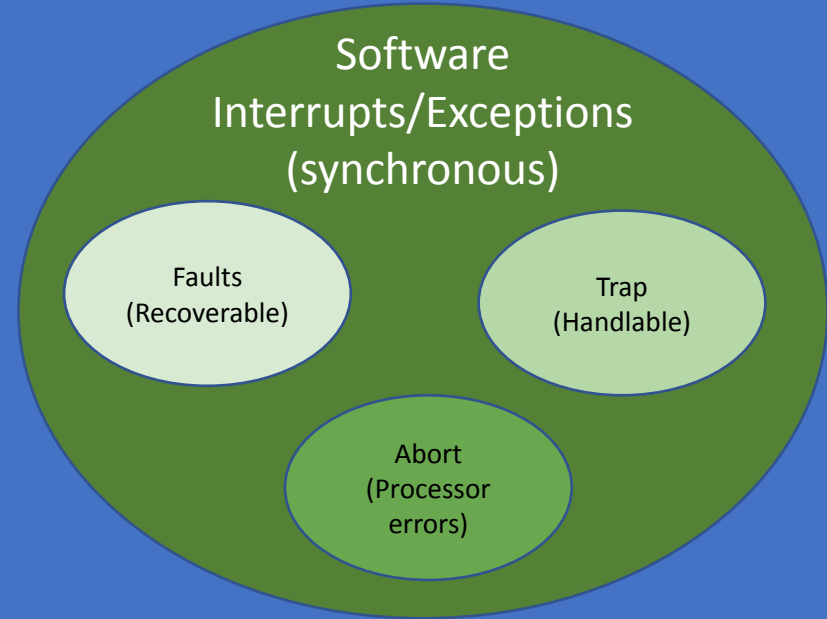


- Classification based on how they are handled:
 - Fault
 - Exception occurred but can be fixed
 - IP points to the current instruction
 - Trap
 - Exception occurred but the program could continue execution
 - IP points to next instruction
 - Abort
 - Unhandleable exception
 - Hardware failures in processor

Interrupts classification



Interrupts



Handling Interrupts



- Interrupts are numbered
- We need to define “what to do” (i.e., code to run) when an interrupt with corresponding number occurs

Handling Interrupts



- Setting an Interrupt Descriptor Table (IDT)

Interrupt Number	Code address
0 (Divide error)	0xf0130304
1 (Debug)	0xf0153333
2 (NMI, Non-maskable Interrupt)	0xf0183273
3 (Breakpoint)	0xf0223933
4 (Overflow)	0xf0333333
...	
8 (Double Fault)	0xf0222293
...	
14 (Page Fault)	0xf0133390
...	...
0x30 (syscall in JOS)	0xf0222222

Handling Interrupts



- Setting an Interrupt Descriptor Table (IDT)

Interrupt Number	Code address
0 (Divide error)	0xf0130304
1 (Debug)	0xf0153333
2 (NMI, Non-maskable Interrupt)	0xf0183273
3 (Breakpoint)	0xf0223933
4 (Overflow)	0xf0333333
...	
8 (Double Fault)	0xf0222293
...	
14 (Page Fault)	0xf0133390
...	...
0x30 (syscall in JOS)	0xf0222222

Load the base address into IDTR

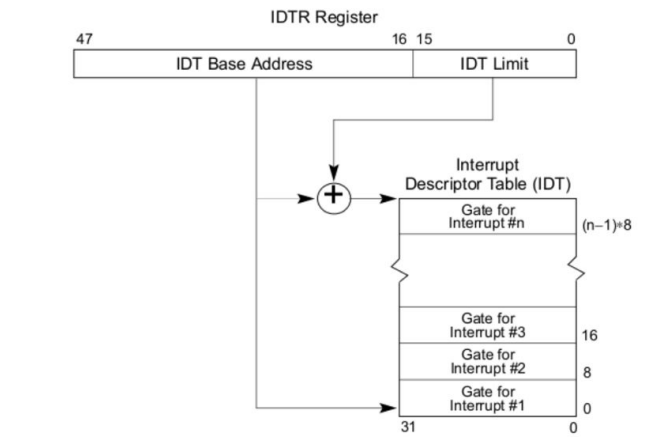


Figure 6-1. Relationship of the IDTR and IDT

Handling Interrupts



- Setting an Interrupt Descriptor Table (IDT)

Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
3 (Breakpoint)	t_brkpt
4 (Overflow)	t_oflow
...	
8 (Double Fault)	t_dblflt
...	
14 (Page Fault)	t_pgflt
...	...
0x30 (syscall in JOS)	t_syscall

Load the base address into IDTR

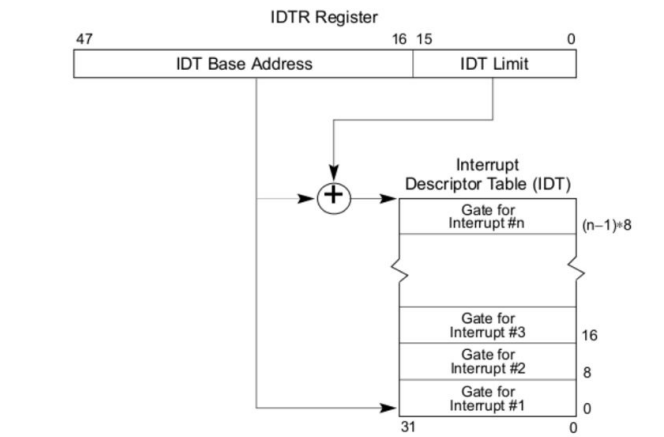


Figure 6-1. Relationship of the IDTR and IDT

Handling Interrupts



- Setting an Interrupt Descriptor Table (IDT)

Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
3 (Breakpoint)	t_brkpt
4 (Overflow)	t_oflow
...	
8 (Double Fault)	t_dblflt
...	
14 (Page Fault)	t_pgflt
...	...
0x30 (syscall in JOS)	t_syscall

```
TRAPHANDLER_NOEC(t_divide, T_DIVIDE); // 0
TRAPHANDLER_NOEC(t_debug, T_DEBUG); // 1
TRAPHANDLER_NOEC(t_nmi, T_NMI); // 2
TRAPHANDLER_NOEC(t_brkpt, T_BRKPT); // 3
TRAPHANDLER_NOEC(t_oflow, T_OFLOW); // 4
TRAPHANDLER_NOEC(t_bound, T_BOUND); // 5
TRAPHANDLER_NOEC(t_illop, T_ILLOP); // 6
TRAPHANDLER_NOEC(t_device, T_DEVICE); // 7

TRAPHANDLER(t_dblflt, T_DBLFLT); // 8

TRAPHANDLER(t_tss, T_TSS); // 10
TRAPHANDLER(t_segnp, T_SEGNP); // 11
TRAPHANDLER(t_stack, T_STACK); // 12
TRAPHANDLER(t_gpflt, T_GPFLT); // 13
TRAPHANDLER(t_pgflt, T_PGFLT); // 14

TRAPHANDLER_NOEC(t_fperr, T_FPERR); // 16

TRAPHANDLER(t_align, T_ALIGN); // 17

TRAPHANDLER_NOEC(t_mchk, T_MCHK); // 18
TRAPHANDLER_NOEC(t_sinderr, T_SIMDERR); // 19

TRAPHANDLER_NOEC(t_syscall, T_SYSCALL); // 48, 0x30
```

Handling Interrupts



Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
3 (Breakpoint)	t_brkpt
4 (Overflow)	t_oflow
...	
8 (Double Fault)	t_dblflt
...	
14 (Page Fault)	t_pgflt
...	...
0x30 (syscall in JOS)	t_syscall

Execution

Program Interrupt

Handling Interrupts



Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
3 (Breakpoint)	t_brkpt
4 (Overflow)	t_oflow
...	
8 (Double Fault)	t_dblflt
...	
14 (Page Fault)	t_pgflt
...	...
0x30 (syscall in JOS)	t_syscall

Execution

Program Interrupt



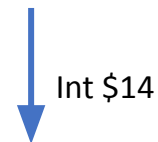
Handling Interrupts



Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
3 (Breakpoint)	t_brkpt
4 (Overflow)	t_oflow
...	
8 (Double Fault)	t_dblflt
...	
14 (Page Fault)	t_pgflt
...	...
0x30 (syscall in JOS)	t_syscall

Execution

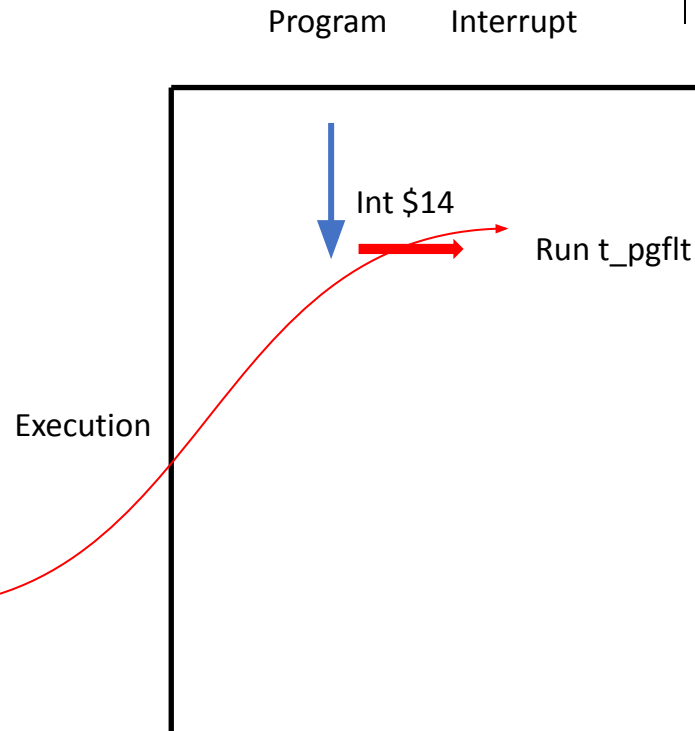
Program Interrupt



Handling Interrupts



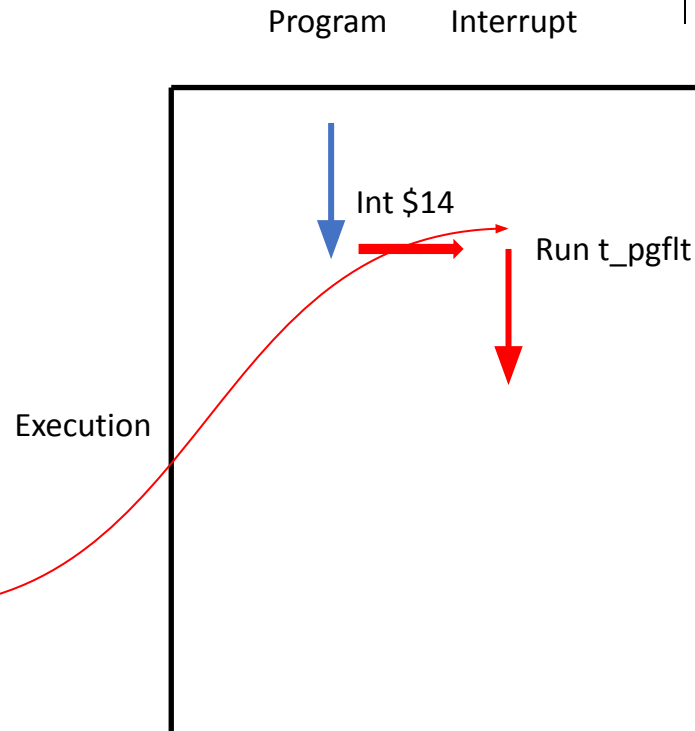
Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
3 (Breakpoint)	t_brkpt
4 (Overflow)	t_oflow
...	
8 (Double Fault)	t_dblflt
...	
14 (Page Fault)	t_pgflt
...	...
0x30 (syscall in JOS)	t_syscall



Handling Interrupts



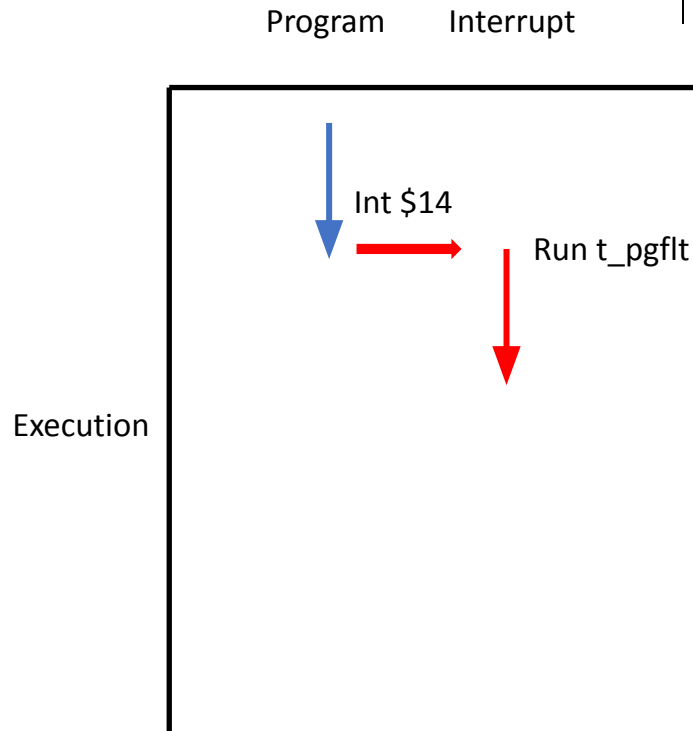
Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
3 (Breakpoint)	t_brkpt
4 (Overflow)	t_oflow
...	
8 (Double Fault)	t_dblflt
...	
14 (Page Fault)	t_pgflt
...	...
0x30 (syscall in JOS)	t_syscall



Simultaneous Interrupts



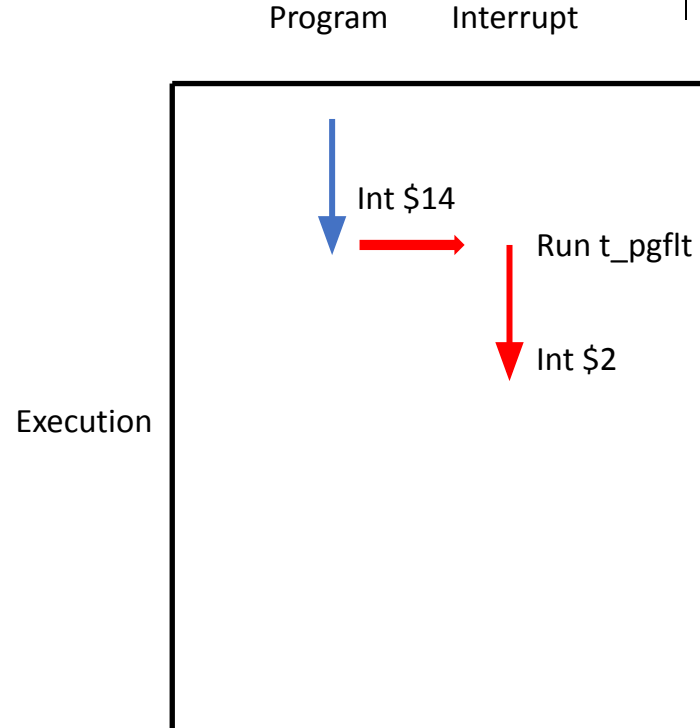
- What if another interrupt happens
 - During processing an interrupt?



Simultaneous Interrupts



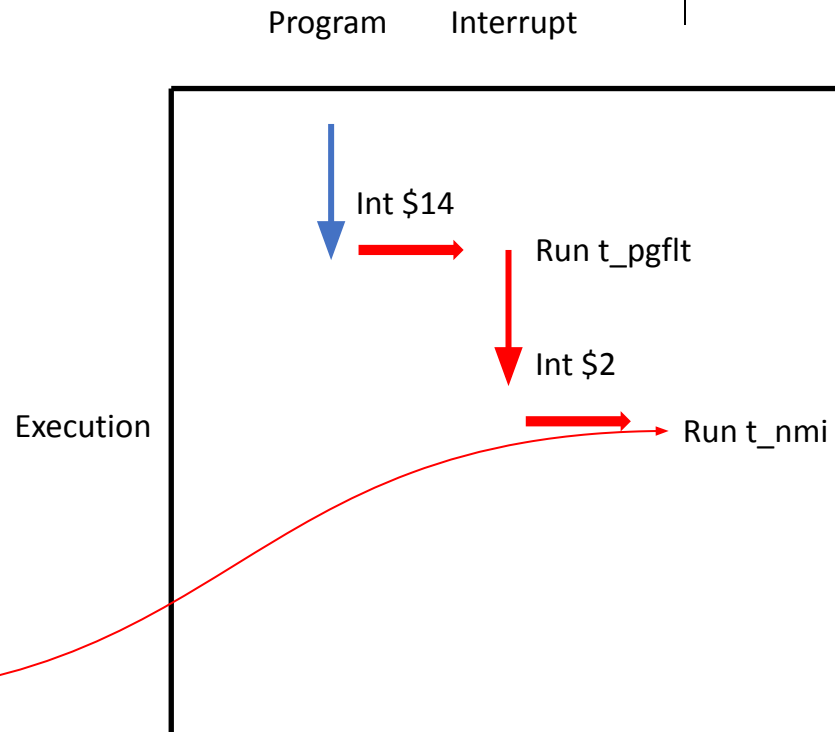
- What if another interrupt happens
 - During processing an interrupt?



Simultaneous Interrupts



- What if another interrupt happens
 - During processing an interrupt?



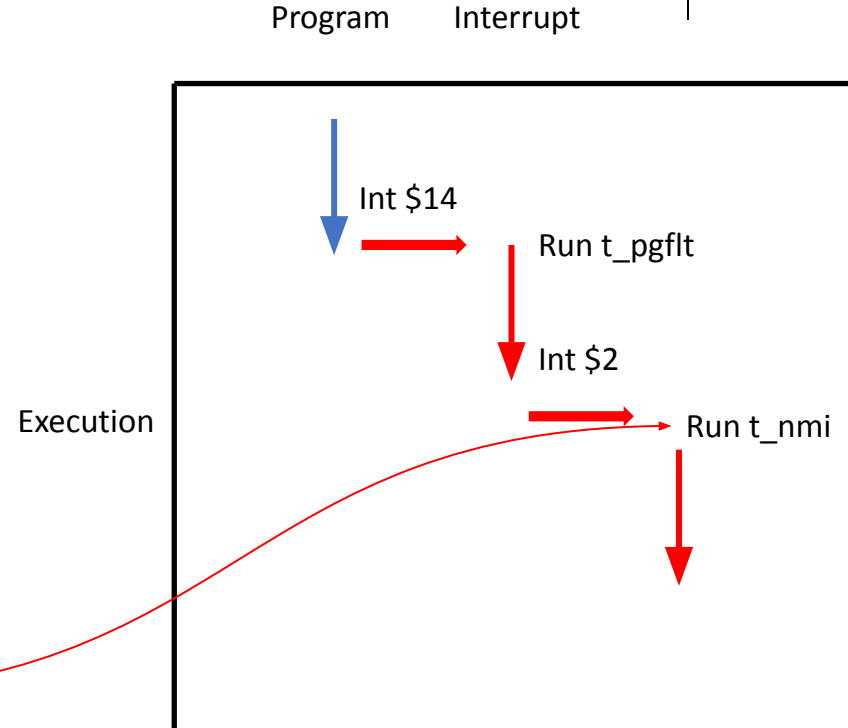
Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
...	

Simultaneous Interrupts



- What if another interrupt happens
 - During processing an interrupt?

Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
...	

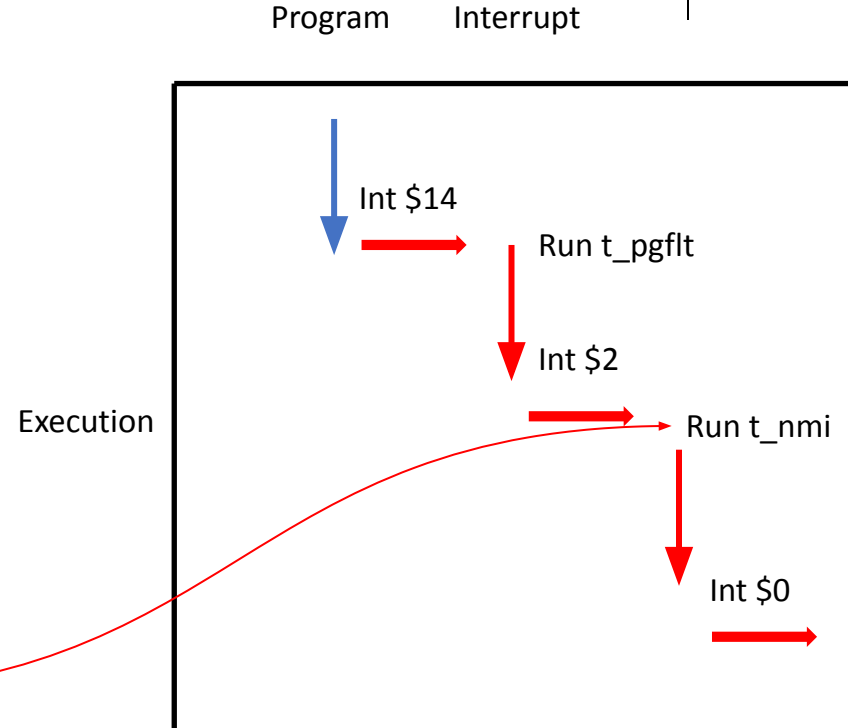


Simultaneous Interrupts



- What if another interrupt happens
 - During processing an interrupt?

Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
...	

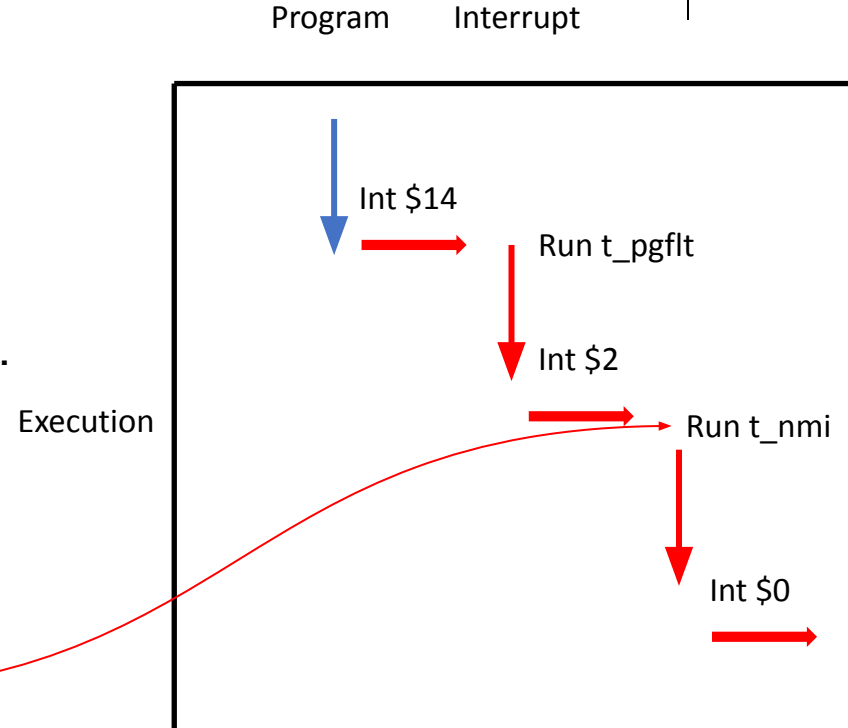


Simultaneous Interrupts



- What if another interrupt happens
 - During processing an interrupt?
- Handle interrupts indefinitely...
 - Cannot continue the program execution
 - Even cannot finish an interrupt handler...

Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
...	



Simultaneous Interrupts



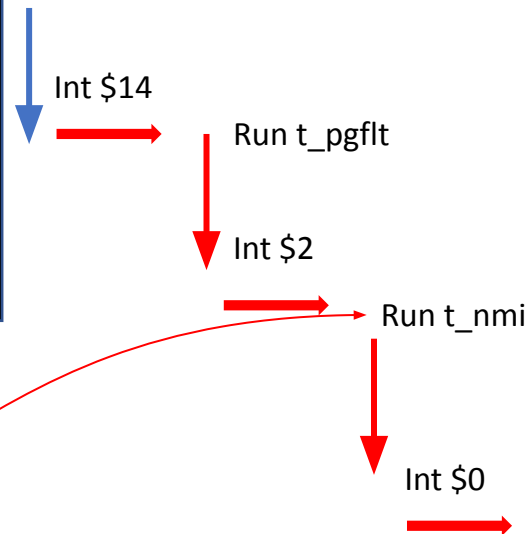
- What if another interrupt happens

- During processing an interrupt?

Interrupt request coming **during handling** an interrupt request could make our interrupt handling **never finish!**

To avoid such an **'infinite' interrupt**,
We **disable interrupt** while handling interrupt...

Program Interrupt



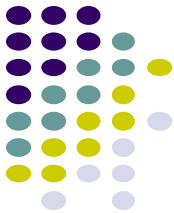
Interrupt Number	Code address
0 (Divide error)	t_divide
1 (Debug)	t_debug
2 (NMI, Non-maskable Interrupt)	t_nmi
...	

Controlling Interrupts

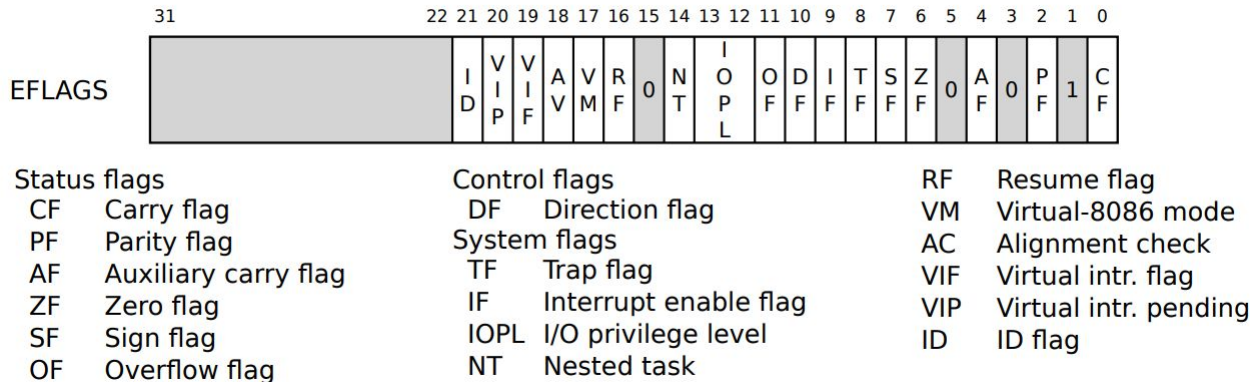
- Enabled/disabled by OS



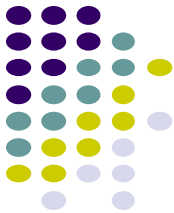
Controlling Interrupts



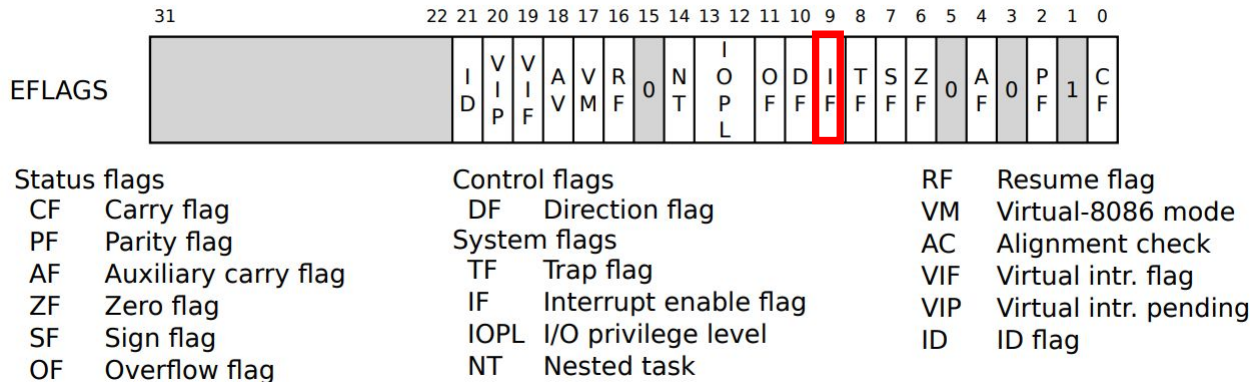
- Enabled/disabled by OS
- IF flag in EFLAGS indicates this
 - `sti` (set interrupt flag, turn on)
 - `cli` (clear interrupt flag, turn off)



Controlling Interrupts



- Enabled/disabled by OS
- IF flag in EFLAGS indicates this
 - `sti` (set interrupt flag, turn on)
 - `cli` (clear interrupt flag, turn off)



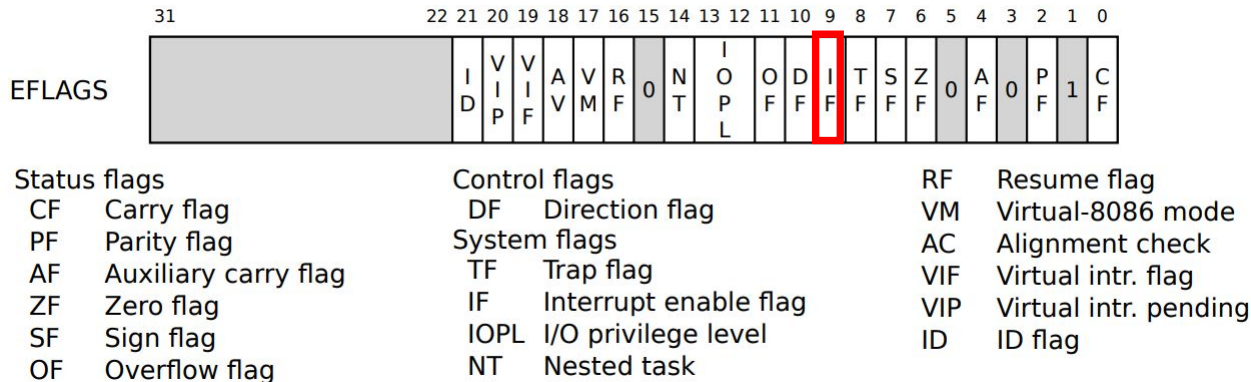
Controlling Interrupts



- Enabled/disabled by OS
- IF flag in EFLAGS indicates this
 - sti (set interrupt flag, turn on)
 - cli (clear interrupt flag, turn off)

```
.globl start
start:
.code16
cli
```

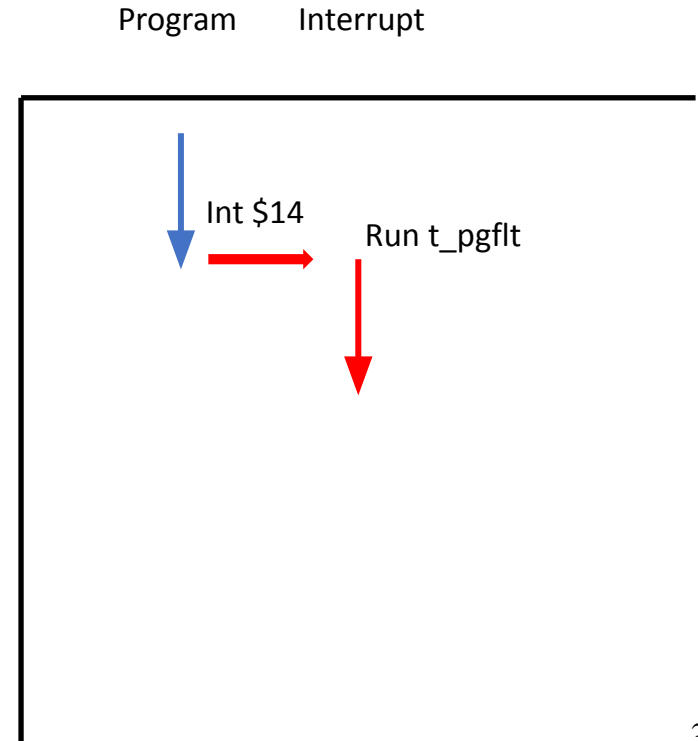
```
# Assemble for 16-bit mode
# Disable interrupts
```



Executing interrupt handlers



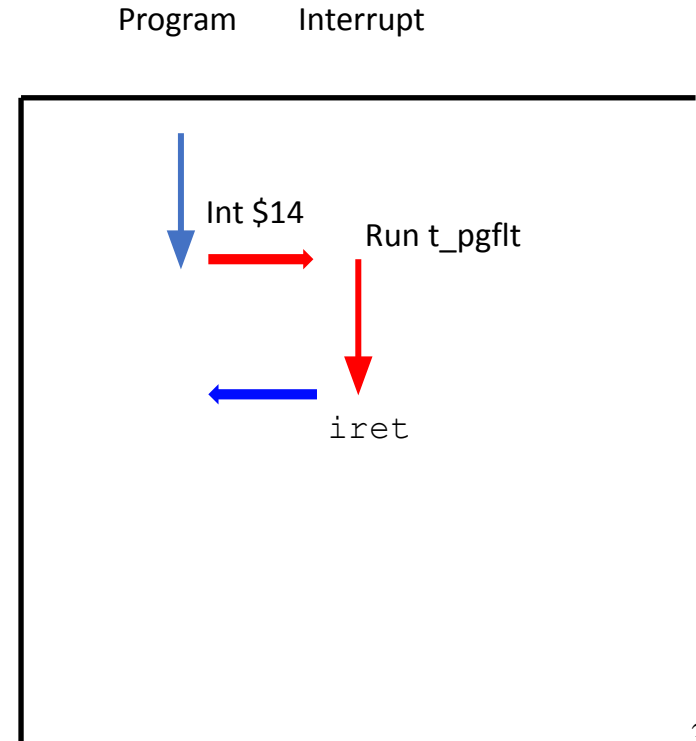
- We would like to handle the interrupt/exceptions at the kernel



Executing interrupt handlers



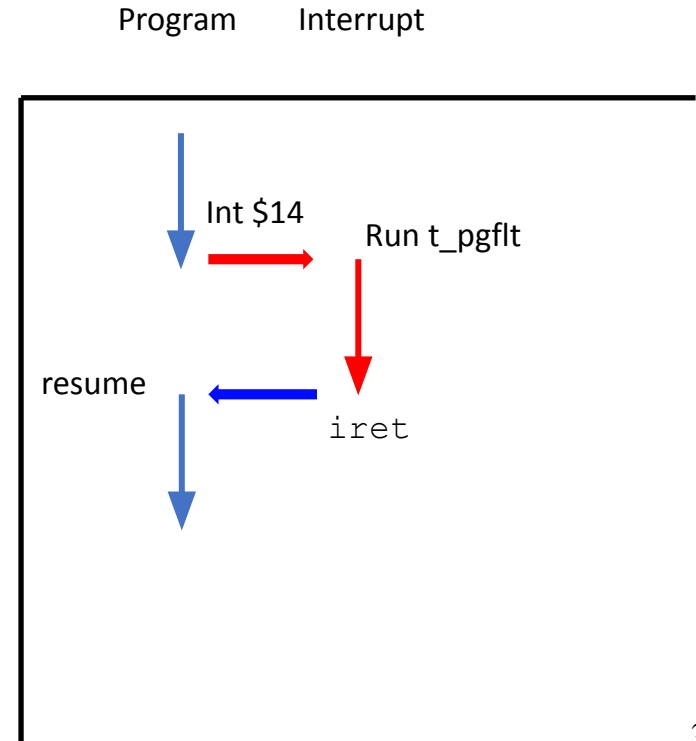
- We would like to handle the interrupt/exceptions at the kernel
- After handing that, we would like to go back to the previous execution



Executing interrupt handlers



- We would like to handle the interrupt/exceptions at the kernel
- After handing that, we would like to go back to the previous execution
- How?
 - Store an execution context



Execution Context



```
int global_value; // don't know the value

int main() {

    int i = 3;
    int j = 5;

    int sum = i;

    sum += global_value;

    sum += j;

    return 0;
}
```

Program Interrupt

Execution Context




```
int global_value; // don't know the value

int main() {
    int i = 3;
    int j = 5;

    int sum = i;
    sum += global_value;
    sum += j;
    return 0;
}
```

Execute



Program Interrupt



Execution Context



```
int global_value; // don't know the value

int main() {

    int i = 3;
    int j = 5;

    int sum = i;
    sum += global_value;
    sum += j;
    return 0;
}
```

Execute

Accessing
a global
variable,
Page fault!

Program Interrupt



Execution Context



```
int global_value; // don't know the value

int main() {

    int i = 3;
    int j = 5;

    int sum = i;
    sum += global_value;
    sum += j;
    return 0;
}
```

Execute

Accessing
a global
variable,
Page fault!

Program Interrupt



Execution Context



```
int global_value; // don't know the value

int main() {

    int i = 3;
    int j = 5;

    int sum = i;

    sum += global_value;

    sum += j;

    return 0;
}
```

return addr
Saved EBP
???
???
???
var i : 3
var j: 5
var sum: i

Program
Stack

Execution Context



```
int global_value; // don't know the value

int main() {

    int i = 3;
    int j = 5;

    int sum = i;

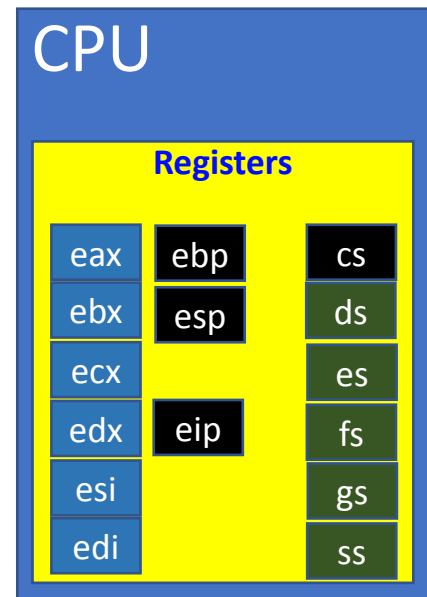
    sum += global_value;

    sum += j;

    return 0;
}
```



Program
Stack



Execution Context



```
int global_value; // don't know the value

int main() {

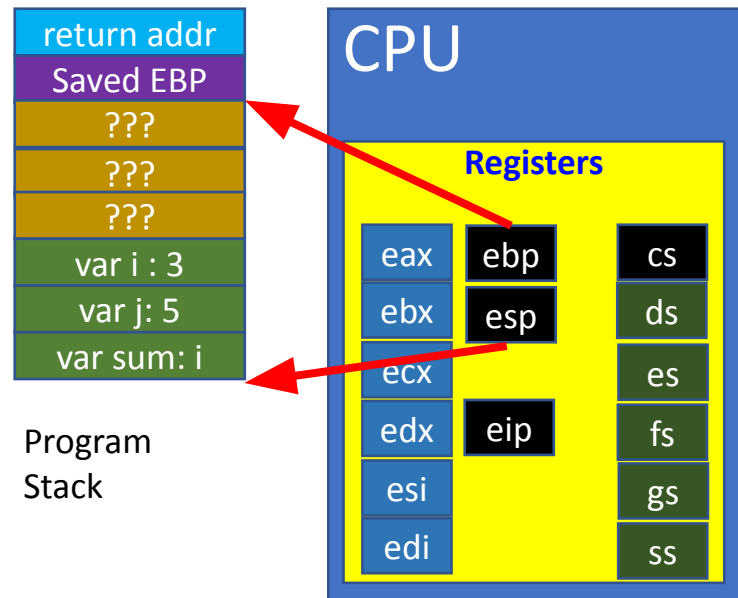
    int i = 3;
    int j = 5;

    int sum = i;

    sum += global_value;

    sum += j;

    return 0;
}
```



Execution Context



```
int global_value; // don't know the value

int main() {

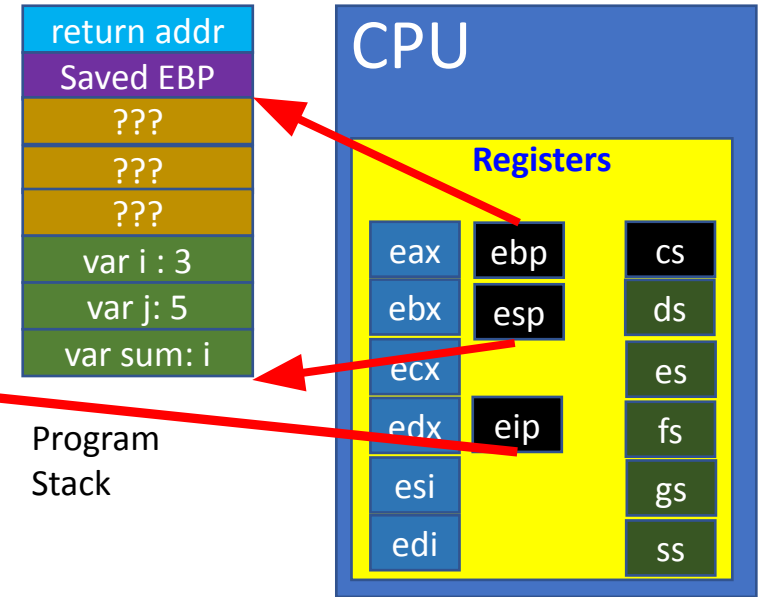
    int i = 3;
    int j = 5;

    int sum = i;

    sum += global_value;

    sum += j;

    return 0;
}
```



Execution Context



```
int global_value; // don't know the value

int main() {

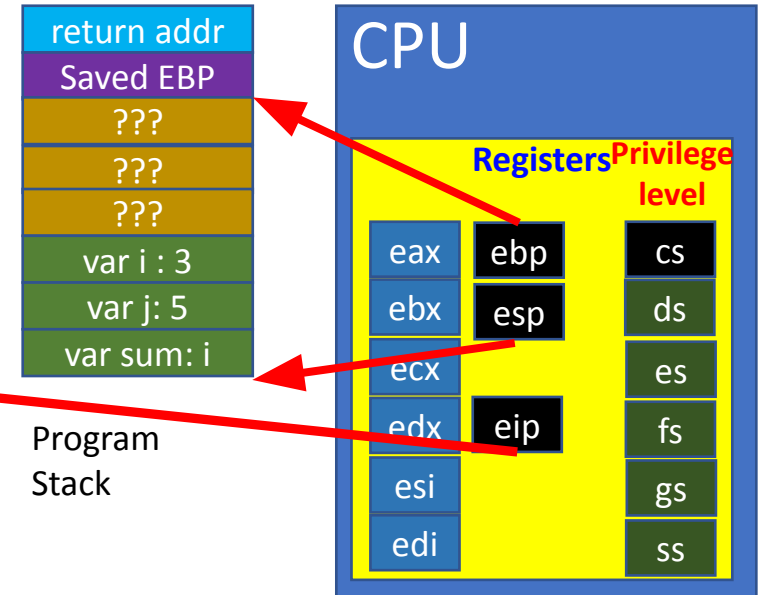
    int i = 3;
    int j = 5;

    int sum = i;

    sum += global_value;

    sum += j;

    return 0;
}
```



Storing an Execution Context



- CPU uses registers and memory (stack) for maintaining an execution context
- Let's store them
 - Stack (%ebp, %esp)
 - Program counter (where our current execution is, %eip)
 - All general purpose registers (%eax, %edx, %ecx, %ebx, %esi, %edi)
 - EFLAGS
 - CS register (why? CPL!)

Storing an Execution Context



- CPU uses registers and memory (stack) for maintaining an execution context

CPU stores some of them for us.

**But, CPU only stores
esp, eip, EFLAGS, ss, cs**

What about the others?

+-----+ KSTACKTOP			
0x000000 old SS		" - 4	
old ESP		" - 8	
old EFLAGS		" - 12	
0x000000 old CS		" - 16	
old EIP		" - 20 <---- ESP	
+-----+			

- Let's store them
 - Stack (%ebp, %esp)
 - Program counter (where our current execution is, %eip)
 - All general purpose registers (%eax, %edx, %ecx, %ebx, %esi, %edi)
 - EFLAGS
 - CS register (why? CPL!)

TrapFrame structure in JOS



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

```
struct PushRegs {
    /* registers as pushed by pusha */
    uint32_t reg_edi;
    uint32_t reg_esi;
    uint32_t reg_ebp;
    uint32_t reg_oesp; /* Useless */
    uint32_t reg_ebx;
    uint32_t reg_edx;
    uint32_t reg_ecx;
    uint32_t reg_eax;
} __attribute__((packed));
```

**JOS stores additional information as
Struct Trapframe**

TrapFrame structure in JOS



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

2 byte padding because cs is 16-bit

2 byte padding because ss is 16-bit

```
struct PushRegs {
    /* registers as pushed by pusha */
    uint32_t reg_edi;
    uint32_t reg_esi;
    uint32_t reg_ebp;
    uint32_t reg_oesp; /* Useless */
    uint32_t reg_ebx;
    uint32_t reg_edx;
    uint32_t reg_ecx;
    uint32_t reg_eax;
} __attribute__((packed));
```

JOS stores additional information as Struct Trapframe

+-----+ KSTACKTOP		
0x000000	old SS	" - 4
	old ESP	" - 8
	old EFLAGS	" - 12
0x000000	old CS	" - 16
	old EIP	" - 20 <---- ESP
+-----+		

TrapFrame structure in JOS



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

2 byte padding because cs is 16-bit

2 byte padding because ss is 16-bit

```
struct PushRegs {
    /* registers as pushed by pusha */
    uint32_t reg_edi;
    uint32_t reg_esi;
    uint32_t reg_ebp;
    uint32_t reg_oesp; /* Useless */
    uint32_t reg_ebx;
    uint32_t reg_edx;
    uint32_t reg_ecx;
    uint32_t reg_eax;
} __attribute__((packed));
```

JOS stores additional information as Struct Trapframe

+-----+ KSTACKTOP	
0x000000 old SS	" - 4
old ESP	" - 8
old EFLAGS	" - 12
0x000000 old CS	" - 16
old EIP	" - 20 <---- ESP
+-----+	

TrapFrame structure in JOS



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

2 byte padding because cs is 16-bit

2 byte padding because ss is 16-bit

```
struct PushRegs {
    /* registers as pushed by pusha */
    uint32_t reg_edi;
    uint32_t reg_esi;
    uint32_t reg_ebp;
    uint32_t reg_oesp; /* Useless */
    uint32_t reg_ebx;
    uint32_t reg_edx;
    uint32_t reg_ecx;
    uint32_t reg_eax;
} __attribute__((packed));
```

JOS stores additional information as Struct Trapframe

+-----+ KSTACKTOP			
0x000000	old SS		" - 4
	old ESP		" - 8
	old EFLAGS		" - 12
0x000000	old CS		" - 16
	old EIP		" - 20 <---- ESP
+-----+			

TrapFrame structure in JOS



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

2 byte padding because cs is 16-bit

2 byte padding because ss is 16-bit

```
struct PushRegs {
    /* registers as pushed by pusha */
    uint32_t reg_edi;
    uint32_t reg_esi;
    uint32_t reg_ebp;
    uint32_t reg_oesp; /* Useless */
    uint32_t reg_ebx;
    uint32_t reg_edx;
    uint32_t reg_ecx;
    uint32_t reg_eax;
} __attribute__((packed));
```

JOS stores additional information as Struct Trapframe

+-----+ KSTACKTOP		
0x000000	old SS	" - 4
	old ESP	" - 8
	old EFLAGS	" - 12
0x000000	old CS	" - 16
	old EIP	" - 20 <---- ESP
+-----+		

TrapFrame structure in JOS



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

2 byte padding because cs is 16-bit

2 byte padding because ss is 16-bit

```
struct PushRegs {
    /* registers as pushed by pusha */
    uint32_t reg_edi;
    uint32_t reg_esi;
    uint32_t reg_ebp;
    uint32_t reg_oesp; /* Useless */
    uint32_t reg_ebx;
    uint32_t reg_edx;
    uint32_t reg_ecx;
    uint32_t reg_eax;
} __attribute__((packed));
```

JOS stores additional information as Struct Trapframe

+-----+ KSTACKTOP		
0x000000	old SS	" - 4
	old ESP	" - 8
	old EFLAGS	" - 12
0x000000	old CS	" - 16
	old EIP	" - 20 <---- ESP
+-----+		

TrapFrame structure in JOS



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

2 byte padding because cs is 16-bit

2 byte padding because ss is 16-bit

```
struct PushRegs {
    /* registers as pushed by pusha */
    uint32_t reg_edi;
    uint32_t reg_esi;
    uint32_t reg_ebp;
    uint32_t reg_oesp; /* Useless */
    uint32_t reg_ebx;
    uint32_t reg_edx;
    uint32_t reg_ecx;
    uint32_t reg_eax;
} __attribute__((packed));
```

JOS stores additional information as Struct Trapframe

+-----+ KSTACKTOP			
0x000000	old SS		" - 4
	old ESP		" - 8
	old EFLAGS		" - 12
0x000000	old CS		" - 16
	old EIP		" - 20 <---- ESP

Setting up interrupt handlers



- You will define an interrupt gate per each interrupt/exception

- Using MACROs defined in trapentry.S

- TRAPHANDLER(name, num)
- TRAPHANDLER_NOEC(name, num)

- Gate generated by this macro should call

- trap() in kern/trap.c
- Implement _alltraps:

```
TRAPHANDLER_NOEC(t_divide, T_DIVIDE); // 0
TRAPHANDLER_NOEC(t_debug, T_DEBUG); // 1
TRAPHANDLER_NOEC(t_nmi, T_NMI); // 2
TRAPHANDLER_NOEC(t_brkpt, T_BRKPT); // 3
TRAPHANDLER_NOEC(t_oflow, T_OFLOW); // 4
TRAPHANDLER_NOEC(t_bound, T_BOUND); // 5
TRAPHANDLER_NOEC(t_illop, T_ILLOP); // 6
TRAPHANDLER_NOEC(t_device, T_DEVICE); // 7

TRAPHANDLER(t_dblflt, T_DBLFLT); // 8

TRAPHANDLER(t_tss, T_TSS); // 10
TRAPHANDLER(t_segnp, T_SEGNP); // 11
TRAPHANDLER(t_stack, T_STACK); // 12
TRAPHANDLER(t_gpflt, T_GPFLT); // 13
TRAPHANDLER(t_pgflt, T_PGFLT); // 14

TRAPHANDLER_NOEC(t_fperr, T_FPERR); // 16

TRAPHANDLER(t_align, T_ALIGN); // 17

TRAPHANDLER_NOEC(t_mchk, T_MCHK); // 18
TRAPHANDLER_NOEC(t_simderr, T_SIMDERR); // 19

TRAPHANDLER_NOEC(t_syscall, T_SYSCALL); // 48, 0x30
```

```
#define TRAPHANDLER(name, num) \
    .globl name; /* define global symbol for 'name' */ \
    .type name, @function; /* symbol type is function */ \
    .align 2; /* align function definition */ \
    name: /* function starts here */ \
    pushl $(num); \
    jmp _alltraps
```



- Using MACROs defined in trapentry.S
 - TRAPHANDLER(name, num)
 - TRAPHANDLER_NOEC(name, num)
- Gate generated by this macro should call
 - trap() in kern/trap.c
 - Implement _alltraps:

```
TRAPHANDLER_NOEC(t_divide, T_DIVIDE); // 0
TRAPHANDLER_NOEC(t_debug, T_DEBUG); // 1
TRAPHANDLER_NOEC(t_nmi, T_NMI); // 2
TRAPHANDLER_NOEC(t_brkpt, T_BRKPT); // 3
TRAPHANDLER_NOEC(t_oflow, T_OFLOW); // 4
TRAPHANDLER_NOEC(t_bound, T_BOUND); // 5
TRAPHANDLER_NOEC(t_illop, T_ILLOP); // 6
TRAPHANDLER_NOEC(t_device, T_DEVICE); // 7

TRAPHANDLER(t_dblflt, T_DBLFLT); // 8

TRAPHANDLER(t_tss, T_TSS); // 10
TRAPHANDLER(t_segnp, T_SEGNP); // 11
TRAPHANDLER(t_stack, T_STACK); // 12
TRAPHANDLER(t_gpflt, T_GPFLT); // 13
TRAPHANDLER(t_pgflt, T_PGFLT); // 14

TRAPHANDLER_NOEC(t_fperr, T_FPERR); // 16

TRAPHANDLER(t_align, T_ALIGN); // 17

TRAPHANDLER_NOEC(t_mchk, T_MCHK); // 18
TRAPHANDLER_NOEC(t_simderr, T_SIMDERR); // 19

TRAPHANDLER_NOEC(t_syscall, T_SYSCALL); // 48, 0x30
```

Which interrupts has EC?



- Intel Manual

- https://purs3lab.github.io/ee469/static_files/read/ia32/IA32-3A.pdf (page 186)

Table 6-1. Protected-Mode Exceptions and Interrupts

Vector	Mnemonic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	Debug Exception	Fault/ Trap	No	Instruction, data, and I/O breakpoints; single-step; and others.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT 3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD2 instruction or reserved opcode. ¹
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction. ²
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.

Processor handling of EC/NOEC interrupts



```
+-----+ KSTACKTOP
| 0x00000 | old SS   | " - 4
|   old ESP   | " - 8
|   old EFLAGS | " - 12
| 0x00000 | old CS   | " - 16
|   old EIP   | " - 20 <---- ESP
+-----+
```

Interrupt context (on the stack)
When there is no error code

```
+-----+ KSTACKTOP
| 0x00000 | old SS   | " - 4
|   old ESP   | " - 8
|   old EFLAGS | " - 12
| 0x00000 | old CS   | " - 16
|   old EIP   | " - 20
|   error code | " - 24 <---- ESP
+-----+
```

Interrupt context (on the stack)
When there is an error code

Handling TrapFrame for EC/NOEC



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

Processor pushes the error code for EC interrupts

+-----+ KSTACKTOP		
0x00000	old SS	" - 4
	old ESP	" - 8
	old EFLAGS	" - 12
0x00000	old CS	" - 16
	old EIP	" - 20
	error code	" - 24 <---- ESP
+-----+		

Handling TrapFrame for EC/NOEC



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

What about NOEC interrupts?

Handling TrapFrame for EC/NOEC



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

What about NOEC interrupts?

```
+-----+ KSTACKTOP
| 0x000000 | old SS | " - 4
|      old ESP      | " - 8
|      old EFLAGS    | " - 12
| 0x000000 | old CS | " - 16
|      old EIP       | " - 20 <---- ESP
+-----+
```


Handling TrapFrame for EC/NOEC



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

What about NOEC interrupts?

```
+-----+ KSTACKTOP
| 0x000000 | old SS | " - 4
|      old ESP      | " - 8
|      old EFLAGS    | " - 12
| 0x000000 | old CS | " - 16
|      old EIP       | " - 20 <---- ESP
+-----+
```

Push 0 as a dummy error code

Handling TrapFrame for EC/NOEC



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uint32_t tf_err_code;
};

#define TRAPHANDLER_NOEC(name, num)
    .globl name;
    .type name, @function;
    .align 2;
    name:
    pushl $0;
    pushl $(num);
    jmp _alltraps
}
```

What about NOEC interrupts?

```
+-----+ KSTACKTOP
| 0x000000 | old SS | " - 4
|         | old ESP | " - 8
|         | old EFLAGS | " - 12
| 0x000000 | old CS | " - 16
|         | old EIP | " - 20 <---- ESP
+-----+
```

Push 0 as a dummy error code

Handling Trap number



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

Handling Trap number



```
#define TRAPHANDLER(name, num) \
    .globl name; /* define global symbol for 'name' */ \
    .type name, @function; /* symbol type is function */ \
    .align 2; /* align function definition */ \
    name: /* function starts here */ \
    pushl $(num); Pushes the interrupt \
    jmp _alltraps number!
```

```
#define TRAPHANDLER_NOEC(name, num) \
    .globl name; \
    .type name, @function; \
    .align 2; \
    name: \
    pushl $0; \
    pushl $(num); Pushes the interrupt \
    jmp _alltraps number!
```

Handling Trap number



```
#define TRAPHANDLER(name, num)
    .globl name; /* define (
    .type name, @function; /* sym
    .align 2; /* align functi
    name: /* function sta
    pushl $(num); Pushes the interrupt
    jmp _alltraps number!
```

```
#define TRAPHANDLER_NOEC(name, num)
    .globl name;
    .type name, @function;
    .align 2;
    name:
    pushl $0;
    pushl $(num); Pushes the interrupt
    jmp _alltraps number!
```

```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

Setting up other parts of TrapFrame



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

+-----+ KSTACKTOP		
0x000000 old SS		" - 4
old ESP		" - 8
old EFLAGS		" - 12
0x000000 old CS		" - 16
old EIP		" - 20
error code		" - 24 <---- ESP
+-----+		

Interrupt number!

Setting up other parts of TrapFrame



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

```
/*
 * Lab 3: Your code here for _alltraps
 */
```

```
_alltraps:
    pushl %ds
    pushl %es
    pushal
```

```
+-----+ KSTACKTOP
| 0x00000 | old SS   | " - 4
|         | old ESP | " - 8
|         | old EFLAGS | " - 12
| 0x00000 | old CS   | " - 16
|         | old EIP  | " - 20
|         | error code | " - 24 <---- ESP
+-----+
```

Interrupt number!

Setting up other parts of TrapFrame



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

```
/*
 * Lab 3: Your code here for _alltraps
 */
```

```
_alltraps:
    pushl %ds
    pushl %es
    pushal
```

+-----+ KSTACKTOP		
0x000000 old SS		" - 4
old ESP		" - 8
old EFLAGS		" - 12
0x000000 old CS		" - 16
old EIP		" - 20
error code		" - 24 <---- ESP
+-----+		

Interrupt number!

Setting up other parts of TrapFrame



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

```
/*
 * Lab 3: Your code here for _alltraps
 */
```

```
_alltraps:
    pushl %ds
    pushl %es
    pushal
```

```
+-----+ KSTACKTOP
| 0x000000 | old SS | " - 4
|         | old ESP | " - 8
|         | old EFLAGS | " - 12
| 0x000000 | old CS | " - 16
|         | old EIP | " - 20
|         | error code | " - 24 <---- ESP
+-----+
```

Interrupt number!

Setting up other parts of TrapFrame



```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

```
/*
 * Lab 3: Your code here for _alltraps
 */
```

_alltraps:

pushl %ds

pushl %es

pushal

**You need to write
more code than
this!**

+-----+ KSTACKTOP		
0x000000 old SS		" - 4
old ESP		" - 8
old EFLAGS		" - 12
0x000000 old CS		" - 16
old EIP		" - 20
error code		" - 24 <---- ESP
+-----+		

Interrupt number!

JOS Interrupt Handling

- Setup the IDT at trap_init() in kern/trap.c

```
void
trap_init(void)
{
    extern struct Segdesc gdt[];

    // LAB 3: Your code here.
    SETGATE(idt[T_DIVIDE], 0, GD_KT, t_divide, 0);
    SETGATE(idt[T_DEBUG], 0, GD_KT, t_debug, 0);
}
```

JOS Interrupt Handling

- Setup the IDT at trap_init() in kern/trap.c
- Interrupt arrives to CPU!
- Call interrupt handler in IDT
- Call _alltraps (in kern/trapentry.S)

```
void
trap_init(void)
{
    extern struct Segdesc gdt[];

    // LAB 3: Your code here.
    SETGATE(idt[T_DIVIDE], 0, GD_KT, t_divide, 0);
    SETGATE(idt[T_DEBUG], 0, GD_KT, t_debug, 0);
}

#define TRAPHANDLER_NOEC(name, num)
    .globl name;
    .type name, @function;
    .align 2;
    name:
    pushl $0;
    pushl $(num);
    jmp _alltraps
```

JOS Interrupt Handling

- Setup the IDT at trap_init() in kern/trap.c
- Interrupt arrives to CPU!
- Call interrupt handler in IDT
- Call _alltraps (in kern/trapentry.S)
- Call trap() in kern/trap.c

```
void
trap_init(void)
{
    extern struct Segdesc gdt[];

    // LAB 3: Your code here.
    SETGATE(idt[T_DIVIDE], 0, GD_KT, t_divide, 0);
    SETGATE(idt[T_DEBUG], 0, GD_KT, t_debug, 0);
}
```

```
#define TRAPHANDLER_NOEC(name, num)
    .globl name;
    .type name, @function;
    .align 2;
    name:
    pushl $0;
    pushl $(num);
    jmp _alltraps
```

```
/*
 * Lab 3: Your code here for _alltraps
 */

_alltraps:
    pushl %ds
    pushl %es
    pushal
```

**Build a
Trapframe!**

JOS Interrupt Handling

```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86 hardware */
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when crossing rings, such as from user to kernel */
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
} __attribute__((packed));
```

```
void
trap_init(void)
{
    extern struct Segdesc gdt[];

    // LAB 3: Your code here.
    SETGATE(idt[T_DIVIDE], 0, GD_KT, t_divide, 0);
    SETGATE(idt[T_DEBUG], 0, GD_KT, t_debug, 0);
}
```

```
#define TRAPHANDLER_NOEC(name, num)
    .globl name;
    .type name, @function;
    .align 2;
    name:
    pushl $0;
    pushl $(num);
    jmp _alltraps
```

```
/*
 * Lab 3: Your code here for _alltraps
 */

_alltraps:
    pushl %ds
    pushl %es
    pushal
```

**Build a
Trapframe!**

JOS Interrupt Handling

- Setup the IDT at trap_init() in kern/trap.c
- Interrupt arrives to CPU!
- Call interrupt handler in IDT
- Call _alltraps (in kern/trapentry.S)
- Call trap() in kern/trap.c

```
void
trap_init(void)
{
    extern struct Segdesc gdt[];

    // LAB 3: Your code here.
    SETGATE(idt[T_DIVIDE], 0, GD_KT, t_divide, 0);
    SETGATE(idt[T_DEBUG], 0, GD_KT, t_debug, 0);
}
```

```
#define TRAPHANDLER_NOEC(name, num)
    .globl name;
    .type name, @function;
    .align 2;
    name:
    pushl $0;
    pushl $(num);
    jmp _alltraps
```

```
/*
 * Lab 3: Your code here for _alltraps
 */

_alltraps:
    pushl %ds    Build a
    pushl %es    Trapframe!
    pushal
```

```
void
trap(struct Trapframe *tf)
{
```

JOS Interrupt Handling

- Setup the IDT at trap_init() in kern/trap.c
- Interrupt arrives to CPU!
- Call interrupt handler in IDT
- Call _alltraps (in kern/trapentry.S)
- Call trap() in kern/trap.c
- Call trap_dispatch() in kern/trap.c

```
static void
trap_dispatch(struct Trapframe *tf)
{
    // Handle processor exceptions.
    // LAB 3: Your code here.
```

```
void
trap_init(void)
{
    extern struct Segdesc gdt[];

    // LAB 3: Your code here.
    SETGATE(idt[T_DIVIDE], 0, GD_KT, t_divide, 0);
    SETGATE(idt[T_DEBUG], 0, GD_KT, t_debug, 0);

#define TRAPHANDLER_NOEC(name, num)
    .globl name;
    .type name, @function;
    .align 2;
    name:
    pushl $0;
    pushl $(num);
    jmp _alltraps
```

```
/*
 * Lab 3: Your code here for _alltraps
 */

_alltraps:
    pushl %ds    Build a
    pushl %es    Trapframe!
    pushal
```

```
void
trap(struct Trapframe *tf)
{
```