The K Project

Introduction

LSE Team

EPITA

septembre 27, 2017

Introduction

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

Figure: K running ‘skate’

Assignment

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

- Serial driver
- Segmentation
- Events
- Keyboard
- Timer
- Syscalls
- VGA driver
- File System
- Binary loading
- Bonus: Sound driver, Console driver, ...

Intermediate handout

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

- Basic serial driver
- Segmentation initialization

What is a kernel ?

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

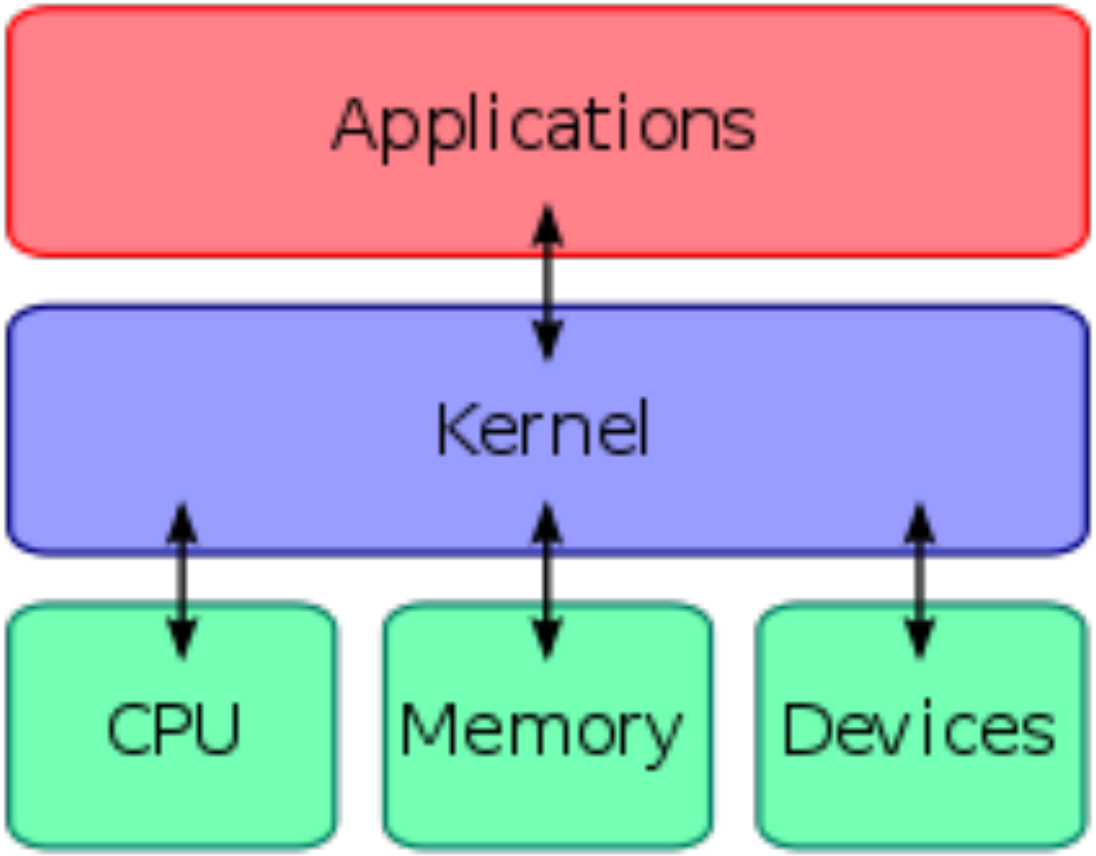

Figure: Operating system layout

Git Repository

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

```
git clone git://git.lse.epita.fr/k.git
```

Source Tree

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

```
|-- config.mk
|-- k
|   |-- compiler.h
|   |-- crt0.S
|   |-- elf.h
|   \-- include
|       \-- k
|   |-- io.h
|   |-- k.lds
|   |-- k.c
|   |-- libvga.c
|   |-- libvga.h
|   |-- Makefile
|   \-- multiboot.h
|-- Makefile
|-- roms
|-- libs
|   |-- libc
|   \-- libk
\-- tools
    |-- create-iso.sh
    |-- mkkfs
    \-- mkksf
```

First step

```
#include "multiboot.h"
#include "kstd.h"

void    k_main(unsigned long                magic,
               multiboot_info_t*            info)
{
    (void) magic;
    (void) info;

    unsigned int i = 0;
    char star[4] = { '|', '/', '-', '\\' };
    char* fb = (void*) 0xb8000;

    while (1)
        *fb = star[i++ % 4];
}
```

Documentation

The K Project

LSE Team

Introduction

Debugging

x86 Architecture

Serial

Conclusion

- http://k.lse.epita.fr/
- http://intel.com/products/processor/manuals/

DEBUG

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

Launch your K

```
$ qemu-system-x86_64 -cdrom k.iso [ -enable-kvm ]
```

Have QEMU wait for your debugger to hook:

- Add "`-s -S`" to QEMU options

Launch gdb and hook to QEMU

```
$> gdb k/k
$(gdb)> target remote localhost:1234
```

Set your breakpoint and continue

Don't forget to build with debug options !

x86 Registers:

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

- General purpose registers
- Segment registers
- Flags
- Control & Memory registers
- Tons of others (XMM0-7...)

General purpose registers

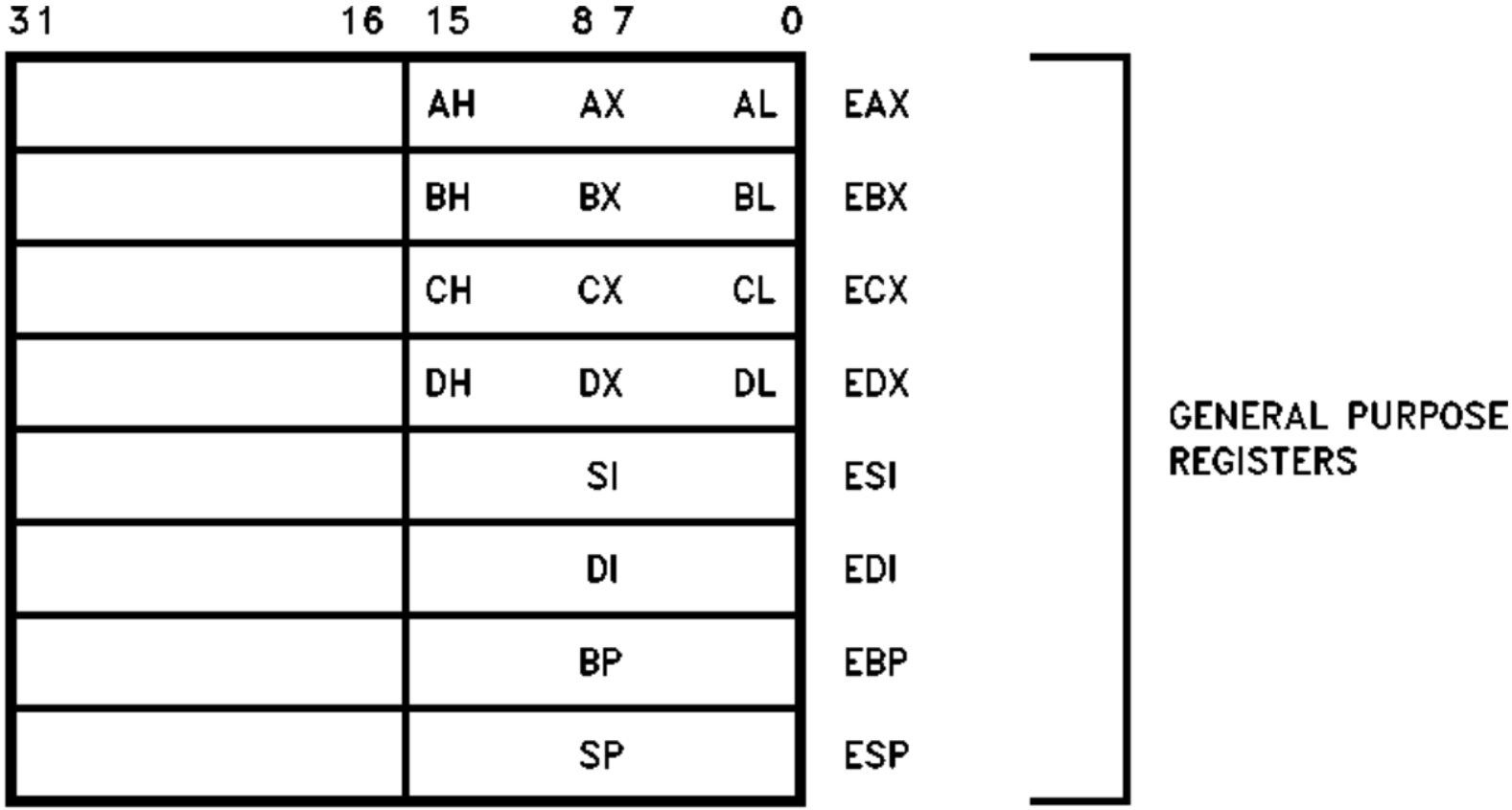

Figure: General purpose registers layout

IP and FLAGS registers

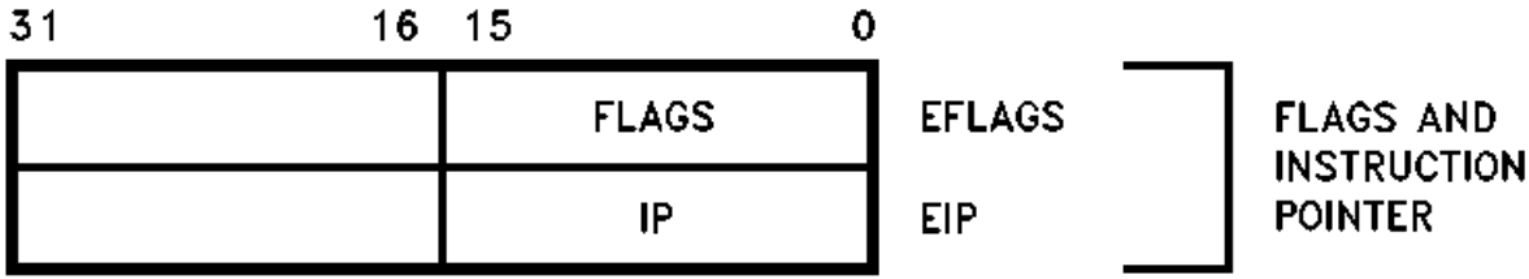

Figure: EIP/IP and EFLAGS/FLAGS

Flags register

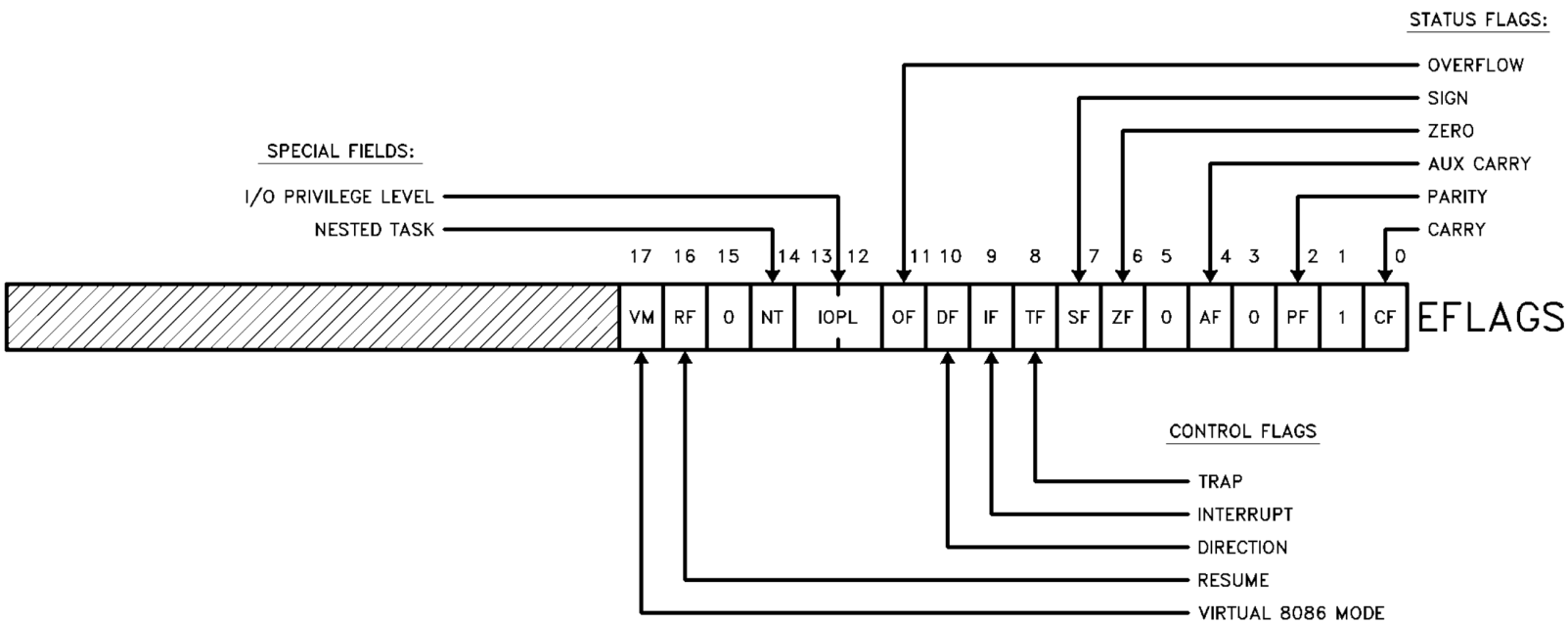

Figure: Flags layout

Rings

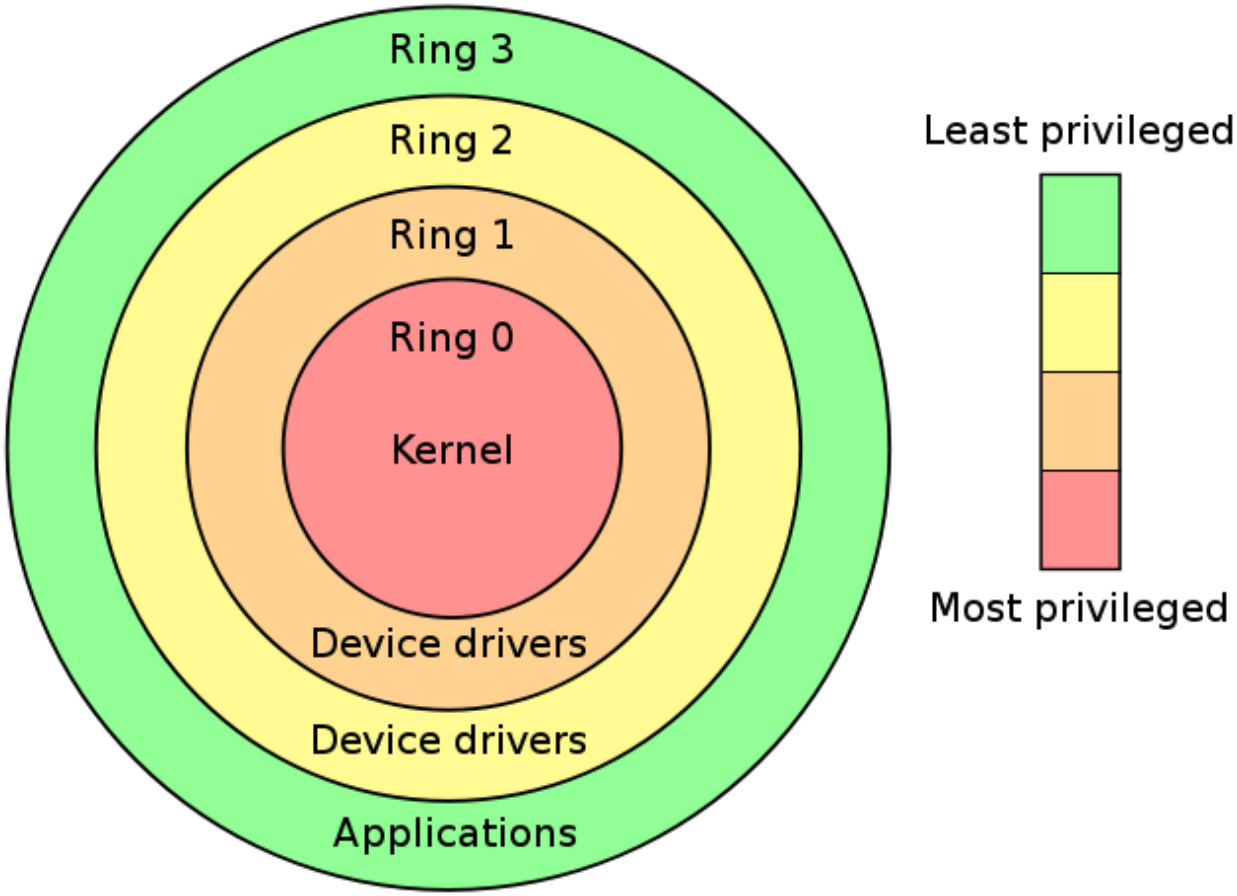

Figure: x86 privileges rings

Calling Conventions

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

C declaration:

```
pushl %eax ; arg3
pushl %ebx ; arg2
pushl %ecx ; arg1
call  foo  ; foo(arg1, arg2, arg3)
```

Think of call as:

```
pushl %eip
%eip = ADDRESS
```

Think of ret as:

```
popl %eip
```

Exemple

Asm exemple for sum(int, int)

```
sum:
    pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %eax ; put first arg in %eax
    addl 12(%ebp), %eax ; add second arg to %eax
    movl %ebp, %esp
    popl %ebp
    ret
```

ASM inline (GAS syntax)

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

Basic syntax:

```
__asm__("movb %ah, (%ebx)");
```

Tell GCC/GAS not to optimize your code:

```
asm volatile ("movl $0, %eax");
```

Note

You can either write GNU keywords and specifiers with or without double underscore around them to avoid name conflicts (`asm` or `__asm__`, `volatile` or `__volatile__`).

ASM inline - next level

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

ASM inline template

```
__asm__("[your assembly code]"
: output operands /* optional */
: input operands /* optional */
: list of clobbered /* optional */
);
```

ASM inline - next level

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

Different output/input constraints:

http://gcc.gnu.org/onlinedocs/gcc/Constraints.html

"m" : memory operand
"r" : register operand

Constraint modifiers

https://gcc.gnu.org/onlinedocs/gcc/Modifiers.html#Modifiers

- "=" : Write Only
- "+" : Read/Write

Different clobbers

memory
[register names]

ASM Inline - example

```
asm volatile("outb %0, %1\n\t"
             : /* No output */
             : "a" (val), "d" (port));

asm volatile("inb %1, %0\n\t"
             : "=&a" (res)
             : "d" (port));
```

Intel vs AT&T

Intel Code	AT&T Code
`mov eax,1`	`mov $1,%eax`
`int 80h`	`int $0x80`
`mov ebx,eax`	`mov %eax,%ebx`
`mov eax,[ebx+3]`	`mov 3(%ebx),%eax`
`mov eax,[ebx+20h]`	`mov 0x20(%ebx),%eax`
`lea eax,[ebx+ecx]`	`lea (%ebx, %ecx),%eax`

Bitfields

```
struct {
    unsigned char field_a : 1; // max value is 0b1
    unsigned char field_b : 2; // max value is 0b11
    unsigned char field_c : 5; // max value is 0x1F
} bitfields;
```

Note

`sizeof(bitfields)` is equal to `sizeof(unsigned char)`.

Packed Structs

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

Not Packed

```
struct {
    unsigned char   a; // aligns with 3 bytes
    unsigned int    b; // aligned
    unsigned char   c; // aligns with 3 bytes
} foo;
```

Note

`sizeof(foo)` gives 12 (1 + 3 padding + 4 + 1 + 3 padding).

Packed Structs

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

Packed

```
struct {
    unsigned char   a;
    unsigned int    b;
    unsigned char   c;
}__attribute__((packed)) bar;
```

Note

`sizeof(bar)` gives 6, struct is memory packed and no padding is inserted.

Your first kernel function !

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

write()

```
int write(const char *buf, size_t count);
```

Note

write() sends to COM1

Note

printf() is available in your kernel and uses write()

PortIO

- Separated adress space
- 2^16 adresses
- in/out x86 instructions family
- Serial/PIC/PIT/Keyboard

Serial initialization

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

- COM1 + 3: (8 bits length) | (No parity)
- COM1 + 2: (FIFO) | (Interrupt trigger level 14 bytes) | (Clear transmit FIFO) | (Clear receive FIFO)
- COM1 + 1: Enable Transmitter Holding Register Empty Interrupt

Serial

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

- A line ends with `\r\n`
- You can redirect the serial output with:

```
qemu-system-x86_64 [...] -serial stdio
```

Serial

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

Serial Registers:
Eight I/O bytes are used for each UART to access its registers.
The following table shows, where each register can be found.
The base address used in the table is the lowest I/O port number assigned.
The switch bit DLAB can be found in the line control register LCR as bit 7 at I/O address base + 3.

UART

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

UART register to port conversion table

	DLAB = 0		DLAB = 1	
I/O port	Read	Write	Read	Write
base	**RBR** receiver buffer	**THR** transmitter holding	**DLL** divisor latch LSB	
base + 1	**IER** interrupt enable	**IER** interrupt enable	**DLM** divisor latch MSB	
base + 2	**IIR** interrupt identification	**FCR** FIFO control	**IIR** interrupt identification	**FCR** FIFO control
base + 3	**LCR** line control			
base + 4	**MCR** modem control			
base + 5	**LSR** line status	– factory test	**LSR** line status	– factory test
base + 6	**MSR** modem status	– not used	**MSR** modem status	– not used
base + 7	**SCR** scratch			

Figure: UART

Divisor

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

DLL and DLM : Divisor latch registers

Speed (bps)	Divisor	DLL	DLM
50	2,304	0x00	0x09
300	384	0x80	0x01
1,200	96	0x60	0x00
2,400	48	0x30	0x00
4,800	24	0x18	0x00
9,600	12	0x0C	0x00
19,200	6	0x06	0x00
38,400	3	0x03	0x00
57,600	2	0x02	0x00
115,200	1	0x01	0x00

Figure: Divisor

FCR

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

FCR : FIFO control register

Bit	Value		Description
0	0		Disable FIFO's
	1		Enable FIFO's
1	0		–
	1		Clear receive FIFO
2	0		–
	1		Clear transmit FIFO
3	0		Select DMA mode 0
	1		Select DMA mode 1
4	0		Reserved
5	0		Reserved (8250, 16450, 16550)
	1		Enable 64 byte FIFO (16750)
6,7	**Bit 7**	**Bit 6**	Receive FIFO interrupt trigger level
	0	0	1 byte
	0	1	4 bytes
	1	0	8 bytes
	1	1	14 bytes

Figure: FCR

IER

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

IER : Interrupt enable register

Bit	Description
0	Received data available
1	Transmitter holding register empty
2	Receiver line status register change
3	Modem status register change
4	Sleep mode (16750 only)
5	Low power mode (16750 only)
6	reserved
7	reserved

Figure: IER

LCR

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

LCR : line control register

Bit	Value			Description
	Bit 1		**Bit 0**	Data word length
0,1	0		0	5 bits
	0		1	6 bits
	1		0	7 bits
	1		1	8 bits
2		0		1 stop bit
		1		1.5 stop bits (5 bits word) 2 stop bits (6, 7 or 8 bits word)
	Bit 5	**Bit 4**	**Bit 3**	
3,4,5	x	x	0	No parity
	0	0	1	Odd parity
	0	1	1	Even parity
	1	0	1	High parity (stick)
	1	1	1	Low parity (stick)
6		0		Break signal disabled
		1		Break signal enabled
7		0		DLAB : RBR, THR and IER accessible
		1		DLAB : DLL and DLM accessible

Figure: LCR

Contact

The K Project
LSE Team
Introduction
Debugging
x86 Architecture
Serial
Conclusion

- lionel[at]lse.epita.fr with [K] tag