Advanced x86:
BIOS and System Management Mode Internals
Input/Output

Xeno Kovah && Corey Kallenberg

LegbaCore, LLC

LEGBACORE
WE DO DIGITAL VOODOO

Input/Output (I/O)

I/O, I/O, it's off to work we go…

2 Types of I/O

1. Memory-Mapped I/O (MMIO)
2. Port I/O (PIO)
   - Also called Isolated I/O or port-mapped IO (PMIO)

- X86 systems employ both-types of I/O
- Both methods map peripheral devices
- Address space of each is accessed using instructions
  - typically requires Ring 0 privileges
  - Real-Addressing mode has no implementation of rings, so no privilege escalation needed
- I/O ports can be mapped so that they appear in the I/O address space or the physical-memory address space (memory mapped I/O) or both
  - Example: PCI configuration space in a PCIe system – both memory-mapped and accessible via port I/O. We'll learn about that in the next section
- The I/O Controller Hub contains the registers that are located in both the I/O Address Space and the Memory-Mapped address space

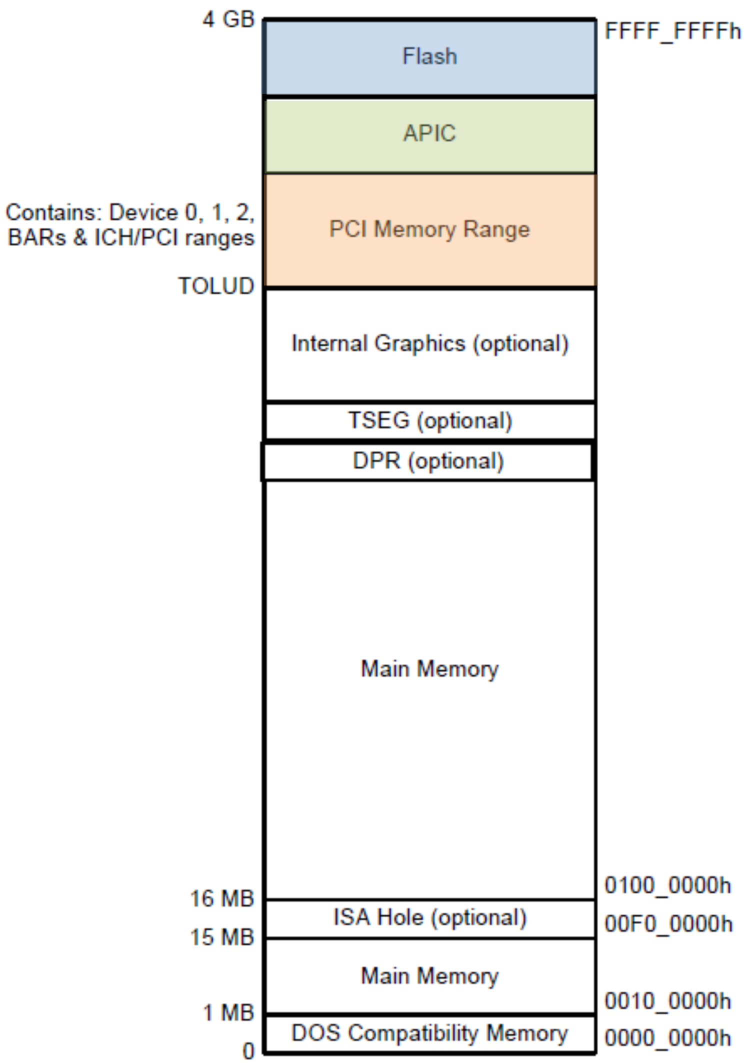
Memory-Mapped I/O

- Devices can also be mapped to the physical address space instead of (or in addition to) the I/O address space
- Even though it is a hardware device on the other end of that access request, you can operate on it like it's memory:
  – Any of the processor’s instructions that reference memory can be used to access an I/O port located at a physical-memory address (MOV, for example)
  – Operations like AND, OR, and TEST can be used on data at a memory-mapped address
- Access byte, word, dword
- The MOV instruction itself requires privileges only in protected mode based on the privilege level of the descriptor describing the segment

Memory-Mapped I/O

- For people not accustomed to working in low-level space, the term memory mapping can be a little confusing, mainly because of how the term is often used, for example:
- “Device X is mapped to memory.”
- People sometimes get confused by this phrasing:
  - Are it’s contents copied to RAM? Or are memory accesses destined for that memory range redirected (decoded) to the device?
- It’s the second one. Accesses destined to that memory range are decoded to the device

Memory Mapped IO



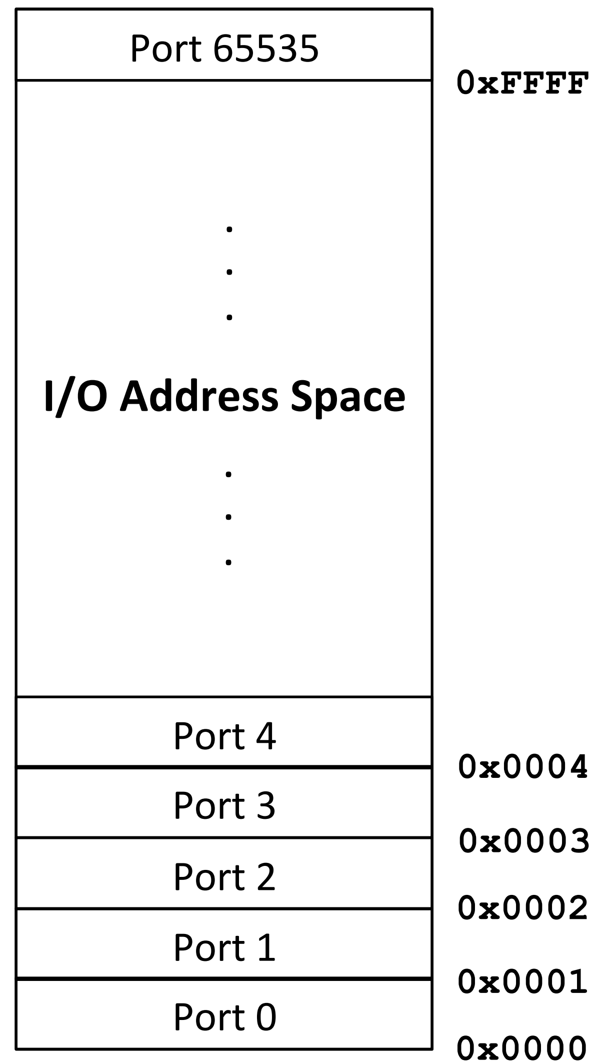
- The colored regions are memory mapped devices
- Accesses to these memory ranges are decoded to a device itself
- Flash refers to the BIOS flash
- APIC is the Advanced Programmable Interrupt Controller
- PCI Memory range is programmed by BIOS in the PCIEXBAR

Peripherals that Map to Both

- Devices can map to both memory and IO address space
- PCI Express is a good example of devices that map to both the IO address space and the physical memory address space
- Compatible PCI configuration space maps to IO Addresses CF8h and CFCh
- Both Compatible PCI configuration space plus the extended header are also mapped to a memory location/ size defined by the PCIEXBAR register located in the DRAM Controller
- We'll get into this again once we get to PCI

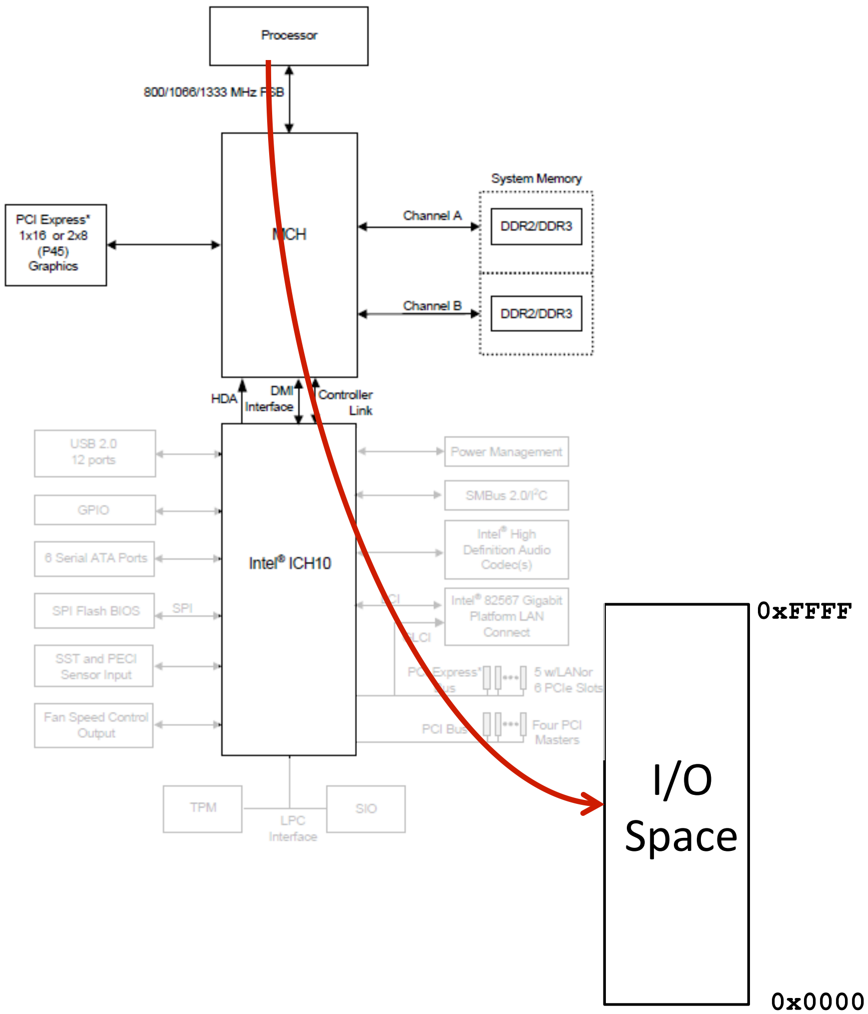
Port I/O Address Space

- Software and hardware architectures of x86 architecture support a separate address space called “I/O Address Space”
  - Separate from memory space
- Access to this separate I/O space is handled through a set of I/O instructions
  - IN,OUT, INS, OUTS
- Access requires Ring0 privileges
  - Access requirement does not apply to all operating modes (like Real-Mode)
- The processor allows 64 KB+3 bytes to be addressed within the I/O space
- Harkens back to a time when memory was not so plentiful
- You may never see port I/O when analyzing high-level applications, but in systems programming (and especially BIOS) you will see lots of port I/O
- One of the biggest impediments to understanding what's going on in a BIOS



Intel Programmer’s guide, Vol 1, 16.1

Port I/O Accesses



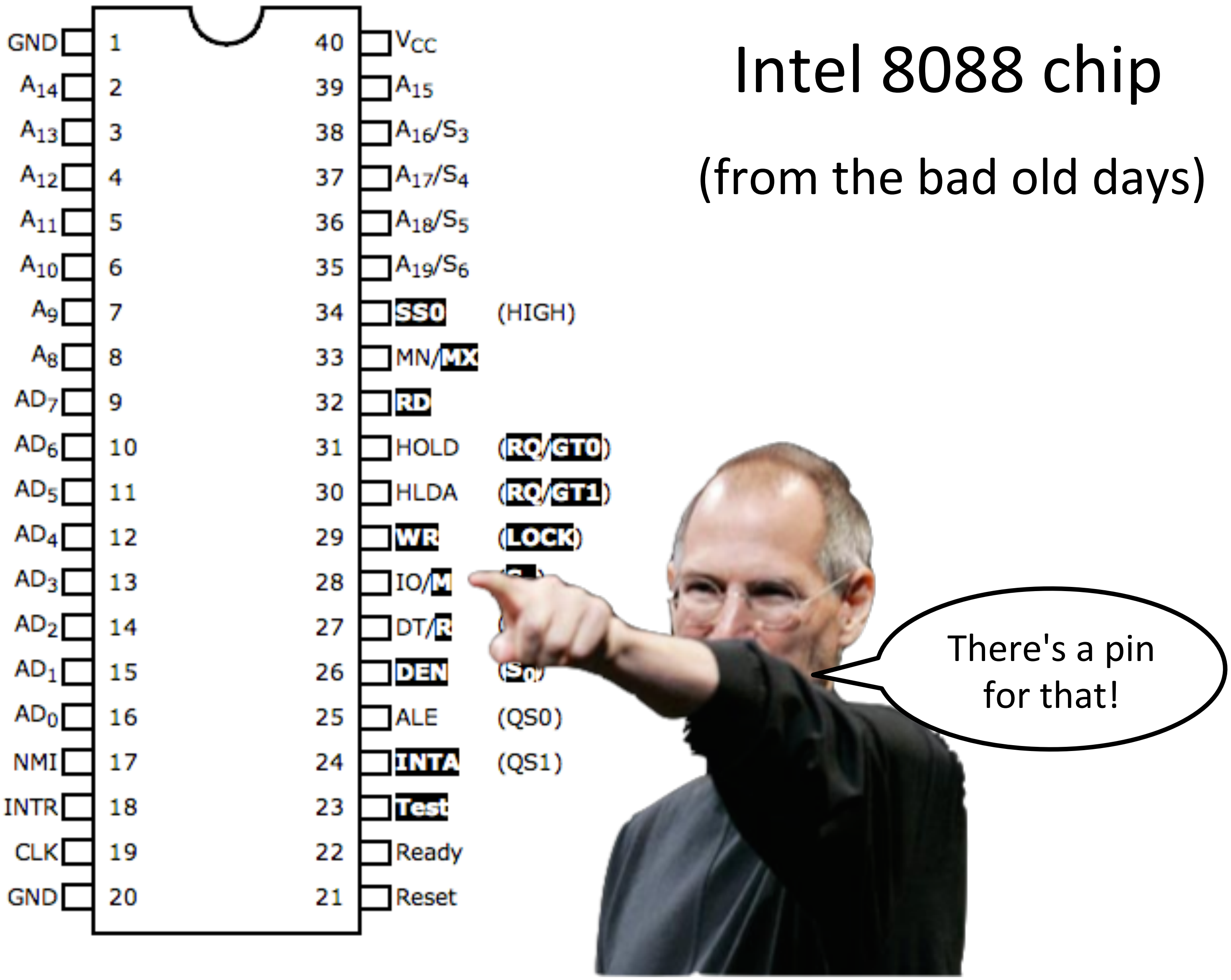
- Port I/O access are handled by the Controller Hub (ICH/ PCH)
  - So in a chipset that has a Memory Controller Hub (MCH), the MCH performs no translation of accesses to I/O space
  - The MCH just forwards them to DMI (and thus to the I/O Controller Hub)
- The Controller Hub contains the registers that are located in the I/O address space
- Again, separate and distinct from physical memory address space

Intel Programmer's guide, Vol 1, 16.1

How does the hardware distinguish between port IO and memory access?

Intel 8088 chip

(from the bad old days)



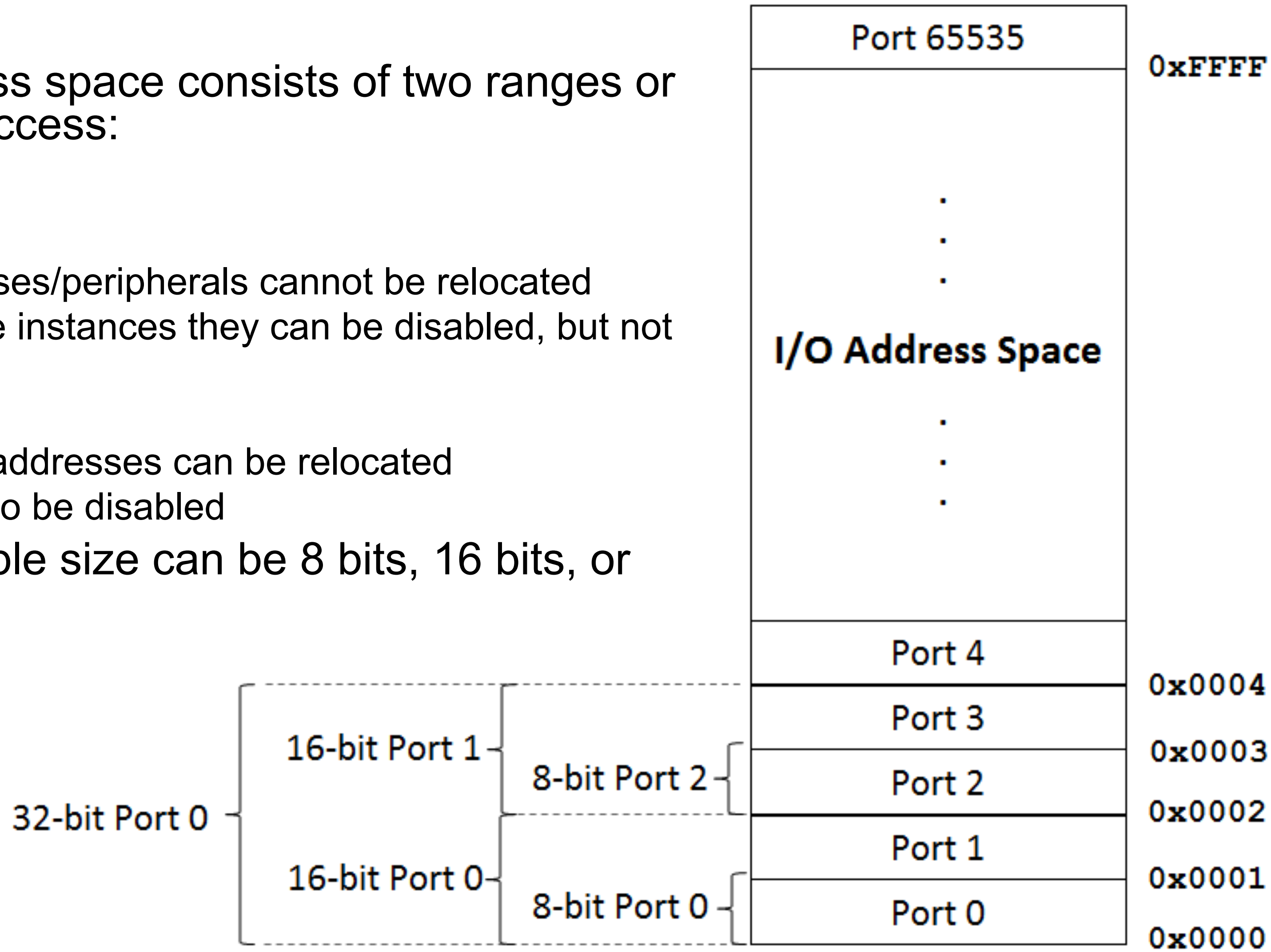
http://www.cpu-world.com/info/Pinouts/8088.html

I/O Mapped Address Space

- I/O Address space consists of two ranges or types of access:

1. Fixed
   - Addresses/peripherals cannot be relocated
   - In some instances they can be disabled, but not all
2. Variable
   - These addresses can be relocated
   - Can also be disabled

- Addressable size can be 8 bits, 16 bits, or 32 bits



1. Fixed I/O Ports

- The addresses depend on the implementation of the I/O Controller Hub present in your system
  - Check the I/O Controller Hub Datasheet to make sure you are interpreting these signals properly
- Address ranges that are not listed or marked “Reserved” are not decoded by the ICH
  - Unless one of the variable ranges has been relocated to that address
- Each fixed IO address is a 2-byte word
- Remember, on the “other side” of each port address/range there is a hardware device
  - Device interaction and behavior will differ between devices
  - This is why it can be difficult to decipher when analyzing
- Port I/O is a gateway to a black box

Intel, Vol 1, 16.1

Example: ICH 9 Fixed Range

I/O Address	Read Target	Write Target	Internal Unit
00h–08h	DMA Controller	DMA Controller	DMA
09h–0Eh	RESERVED	DMA Controller	DMA
0Fh	DMA Controller	DMA Controller	DMA
10h–18h	DMA Controller	DMA Controller	DMA
19h–1Eh	RESERVED	DMA Controller	DMA
1Fh	DMA Controller	DMA Controller	DMA
20h–21h	Interrupt Controller	Interrupt Controller	Interrupt
24h–25h	Interrupt Controller	Interrupt Controller	Interrupt
28h–29h	Interrupt Controller	Interrupt Controller	Interrupt
2Ch–2Dh	Interrupt Controller	Interrupt Controller	Interrupt
2E–2F	LPC SIO	LPC SIO	Forwarded to LPC
30h–31h	Interrupt Controller	Interrupt Controller	Interrupt
34h–35h	Interrupt Controller	Interrupt Controller	Interrupt
38h–39h	Interrupt Controller	Interrupt Controller	Interrupt
3Ch–3Dh	Interrupt Controller	Interrupt Controller	Interrupt
40h–42h	Timer/Counter	Timer/Counter	PIT (8254)
43h	RESERVED	Timer/Counter	PIT
4E–4F	LPC SIO	LPC SIO	Forwarded to LPC
50h–52h	Timer/Counter	Timer/Counter	PIT
53h	RESERVED	Timer/Counter	PIT
60h	Microcontroller	Microcontroller	Forwarded to LPC

Port 60 is the historic location of the 8042 keyboard controller status/command port. And port 64 is the data port. Notice how it doesn't tell you that, it just says they're being forwarded on to LCP. Annoying for trying to figure out what's being talked to

I/O Address	Read Target	Write Target	Internal Unit
60h	Microcontroller	Microcontroller	Forwarded to LPC
61h	NMI Controller	NMI Controller	Processor I/F
62h	Microcontroller	Microcontroller	Forwarded to LPC
64h	Microcontroller	Microcontroller	Forwarded to LPC
66h	Microcontroller	Microcontroller	Forwarded to LPC
70h	RESERVED	NMI and RTC Controller	RTC
71h	RTC Controller	RTC Controller	RTC
72h	RTC Controller	NMI and RTC Controller	RTC
73h	RTC Controller	RTC Controller	RTC
74h	RTC Controller	NMI and RTC Controller	RTC
75h	RTC Controller	RTC Controller	RTC
76h	RTC Controller	NMI and RTC Controller	RTC
77h	RTC Controller	RTC Controller	RTC
80h	DMA Controller, or LPC, or PCI	DMA Controller and LPC or PCI	DMA
81h–83h	DMA Controller	DMA Controller	DMA
84h–86h	DMA Controller	DMA Controller and LPC or PCI	DMA
87h	DMA Controller	DMA Controller	DMA
88h	DMA Controller	DMA Controller and LPC or PCI	DMA
89h–8Bh	DMA Controller	DMA Controller	DMA
8Ch–8Eh	DMA Controller	DMA Controller and LPC or PCI	DMA
08Fh	DMA Controller	DMA Controller	DMA
90h–91h	DMA Controller	DMA Controller	DMA
92h	Reset Generator	Reset Generator	Processor I/F
93h–9Fh	DMA Controller	DMA Controller	DMA

I/O Address	Read Target	Write Target	Internal Unit
A0h–A1h	Interrupt Controller	Interrupt Controller	Interrupt
A4h–A5h	Interrupt Controller	Interrupt Controller	Interrupt
A8h–A9h	Interrupt Controller	Interrupt Controller	Interrupt
ACh–ADh	Interrupt Controller	Interrupt Controller	Interrupt
B0h–B1h	Interrupt Controller	Interrupt Controller	Interrupt
B2h–B3h	Power Management	Power Management	Power Management
B4h–B5h	Interrupt Controller	Interrupt Controller	Interrupt
B8h–B9h	Interrupt Controller	Interrupt Controller	Interrupt
BCh–BDh	Interrupt Controller	Interrupt Controller	Interrupt
C0h–D1h	DMA Controller	DMA Controller	DMA
D2h–DDh	RESERVED	DMA Controller	DMA
DEh–DFh	DMA Controller	DMA Controller	DMA
F0h	PCI and Master Abort[1]	FERR#/IGNNE# / Interrupt Controller	Processor I/F
170h–177h	SATA Controller or PCI	SATA Controller or PCI	Forwarded to SATA
1F0h–1F7h	SATA Controller or PCI	SATA Controller or PCI	Forwarded to SATA
376h	SATA Controller or PCI	SATA Controller or PCI	Forwarded to SATA
3F6h	SATA Controller or PCI	SATA Controller or PCI	Forwarded to SATA
4D0h–4D1h	Interrupt Controller	Interrupt Controller	Interrupt
CF9h	Reset Generator	Reset Generator	Processor I/F

This one we'll talk about explicitly later in the context of SMM

Takeaway: there's a lot of fixed IO address space, and it's fragmented too. This is why it's recommended that devices map their interfaces to memory rather than IO address space

2. Variable I/O Ports

- Can be relocated to another address
- Can be set/disabled using Base Address Registers (BARs) or configuration bits in the various PCI configuration spaces
  - Which we shall discuss very soon!
- The BIOS (and/or other PCI devices or ACPI) can adjust these values
  - Actually pretty much any privileged app can…
- The same as the fixed range, on the “other side” of each port address/range there is a peripheral device
  - Device interaction and behavior will differ between devices
- ICH does not check for overlap
  - Results “unpredictable” if overlapping
    - Has been used for virtualization attacks

Intel, Vol 1, 16.1

Example: ICH 9 Variable IO Range

Table 9-3. Variable I/O Decode Ranges

Range Name	Mappable	Size (Bytes)	Target
ACPI	Anywhere in 64 KB I/O Space	64	Power Management
IDE Bus Master	Anywhere in 64 KB I/O Space	16	IDE Unit
Native IDE Command	Anywhere in 64 KB I/O Space	8	IDE Unit
Native IDE Control	Anywhere in 64 KB I/O Space	4	IDE Unit
USB UHCI Controller #1	Anywhere in 64 KB I/O Space	32	USB Unit 1
USB UHCI Controller #2	Anywhere in 64 KB I/O Space	32	USB Unit 2
USB UHCI Controller #3	Anywhere in 64 KB I/O Space	32	USB Unit 3
USB UHCI Controller #4	Anywhere in 64 KB I/O Space	32	USB Unit 4
USB UHCI Controller #5	Anywhere in 64 KB I/O Space	32	USB Unit 5
USB UHCI Controller #6	Anywhere in 64 KB I/O Space	32	USB Unit 6
SMBus	Anywhere in 64 KB I/O Space	32	SMB Unit
TCO	96 Bytes above ACPI Base	32	TCO Unit
GPIO	Anywhere in 64 KB I/O Space	64	GPIO Unit
Parallel Port	3 Ranges in 64 KB I/O Space	8	LPC Peripheral
Serial Port 1	8 Ranges in 64 KB I/O Space	8	LPC Peripheral
Serial Port 2	8 Ranges in 64 KB I/O Space	8	LPC Peripheral
Floppy Disk Controller	2 Ranges in 64 KB I/O Space	8	LPC Peripheral
LAN	Anywhere in 64 KB I/O Space	32	LAN Unit
LPC Generic 1	Anywhere in 64 KB I/O Space	4 to 256	LPC Peripheral
LPC Generic 2	Anywhere in 64 KB I/O Space	4 to 256	LPC Peripheral
LPC Generic 3	Anywhere in 64 KB I/O Space	4 to 256	LPC Peripheral
LPC Generic 4	Anywhere in 64 KB I/O Space	4 to 256	LPC Peripheral
I/O Trapping Ranges	Anywhere in 64 KB I/O Space	1 to 256	Trap on Backbone

IN - Input from Port

IN—Input from Port

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
E4 *ib*	IN AL, *imm8*	Valid	Valid	Input byte from *imm8* I/O port address into AL.
E5 *ib*	IN AX, *imm8*	Valid	Valid	Input word from *imm8* I/O port address into AX.
E5 *ib*	IN EAX, *imm8*	Valid	Valid	Input dword from *imm8* I/O port address into EAX.
EC	IN AL,DX	Valid	Valid	Input byte from I/O port in DX into AL.
ED	IN AX,DX	Valid	Valid	Input word from I/O port in DX into AX.
ED	IN EAX,DX	Valid	Valid	Input doubleword from I/O port in DX into EAX.

- Note it's DX, not DL. That means the DX form can specify all 2^16 ports, but the IMM8 form can only specify 2^8 ports.
- "When accessing a 16- and 32-bit I/O port, the operand-size attribute determines the port size." (Because as usual there's an overloaded opcode for 16/32 bit form)
  - Remember if you're in a 16 bit segment it's 16 bit, if you're in a 32 bit segment it's 32 bit. But you can override it with an operand size instruction prefix which is talked about later.

OUT - Output to Port

OUT—Output to Port

Opcode*	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
E6 *ib*	OUT *imm8*, AL	Valid	Valid	Output byte in AL to I/O port address *imm8*.
E7 *ib*	OUT *imm8*, AX	Valid	Valid	Output word in AX to I/O port address *imm8*.
E7 *ib*	OUT *imm8*, EAX	Valid	Valid	Output doubleword in EAX to I/O port address *imm8*.
EE	OUT DX, AL	Valid	Valid	Output byte in AL to I/O port address in DX.
EF	OUT DX, AX	Valid	Valid	Output word in AX to I/O port address in DX.
EF	OUT DX, EAX	Valid	Valid	Output doubleword in EAX to I/O port address in DX.

- Basically the same caveat as IN

From Xeno Kovah's Intermediate X86 class: http://OpenSecurityTraining.info/IntermediateX86.html

IO Privilege Level

- There are two bits in the *FLAGS register which the OS will typically set to 0, which indicate that only ring 0 is allowed to issue the IN/OUT instructions

= Intro x86-64

= Intermediate x86-64

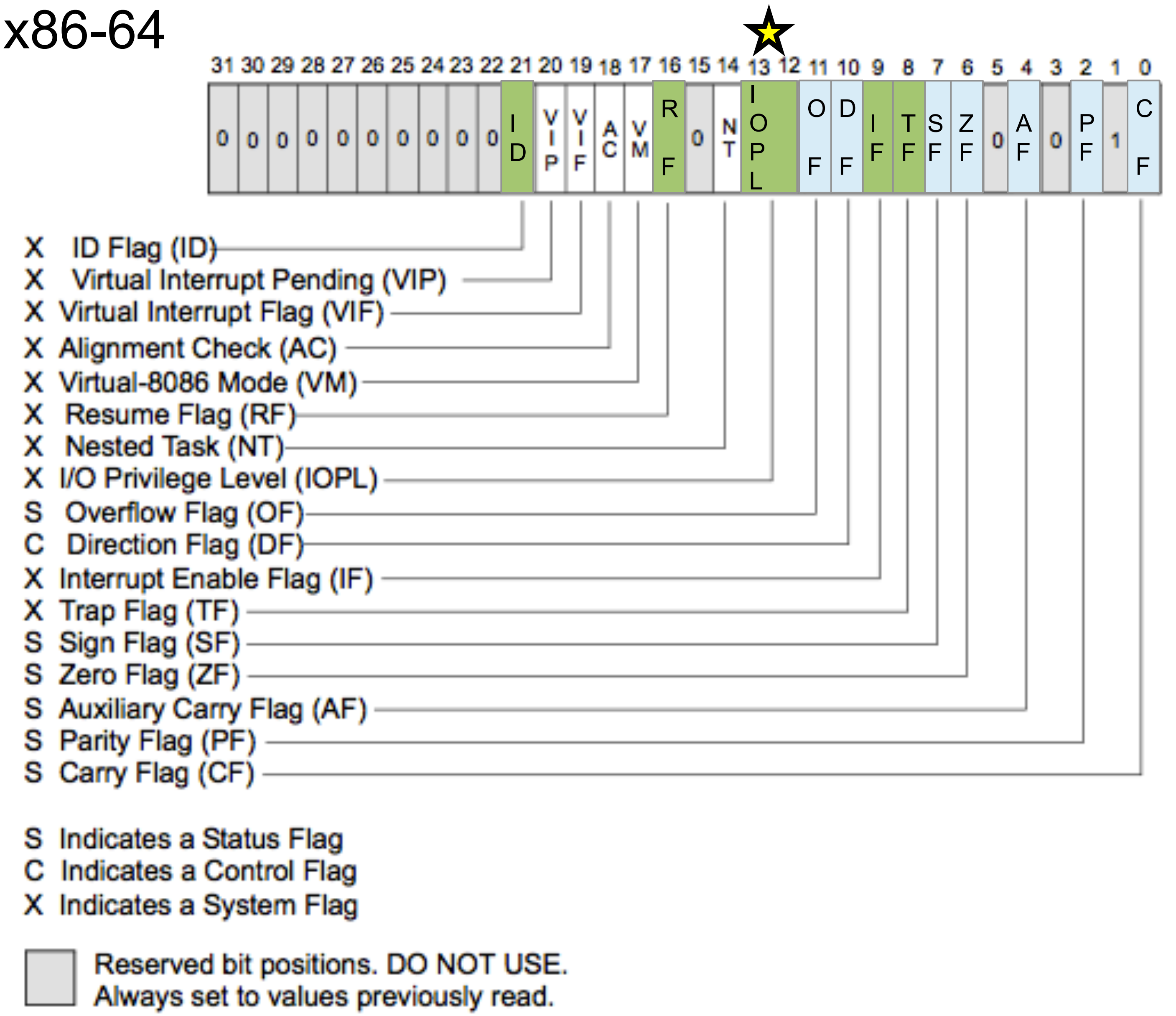


Figure 3-8. EFLAGS Register

Intel Vol 1 Sec 3.4.3.1 - page 3-21 - May 2012 manuals

Port IO Assembly Examples

(showing that you can either use an 8 bit immediate or a 16 bit register (dx) to specify port and 8 bit immediate or 8/16/32 bit registers (but only AL/AX/EAX) to specify the data being read/written)

Read from port 0xB3:

```
IN AL, 0xB3
```

Write 0x1234 to port 0xB2:

```
MOV AX, 0x1234
OUT 0xB2, AX
```

```
MOV AX, 0x1234
MOV DX, 0xB2
OUT DX, AX
```

Index/Data pair read offset 0x05:

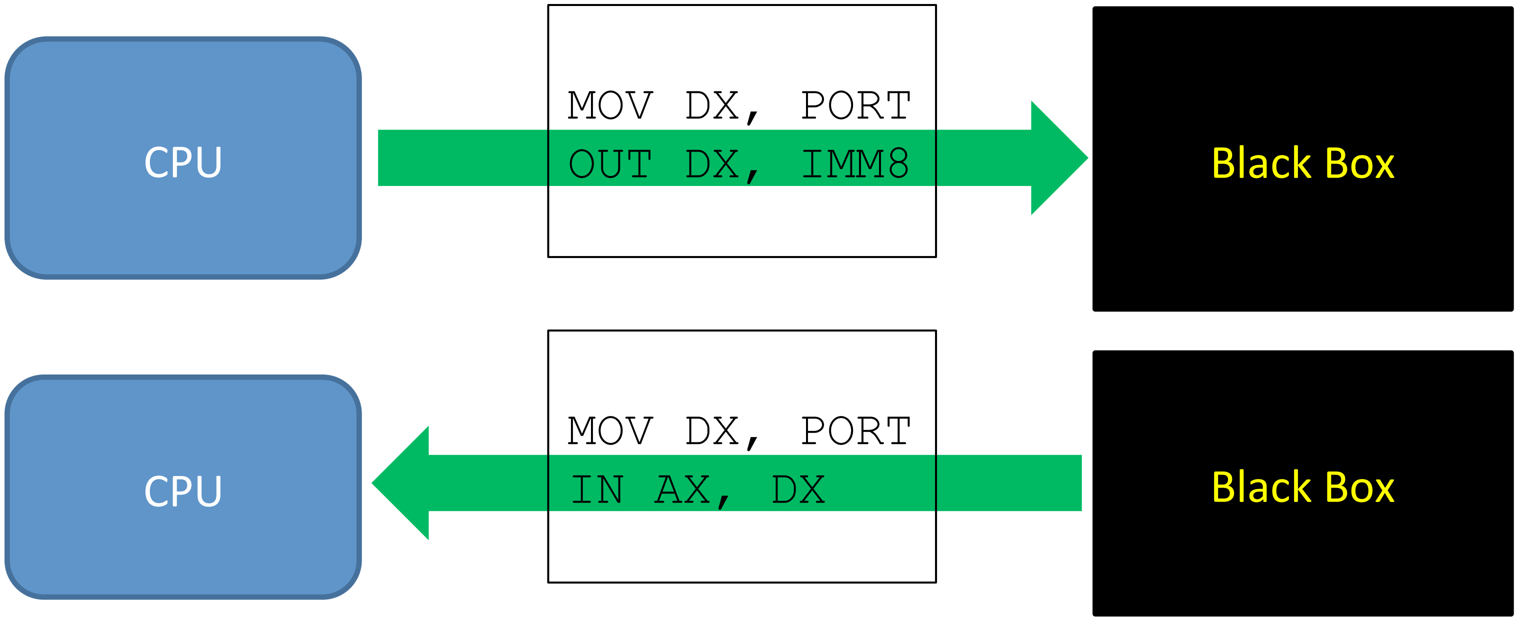
```
MOV DX, 0x70
OUT DX, 0x05
MOV DX, 0x71
IN EAX, DX
```

```
MOV AL, 0x05
MOV DX, 0x70
OUT DX, AL
MOV DX, 0x71
IN EAX, DX
```

Index/Data pair write to offset 0x05:

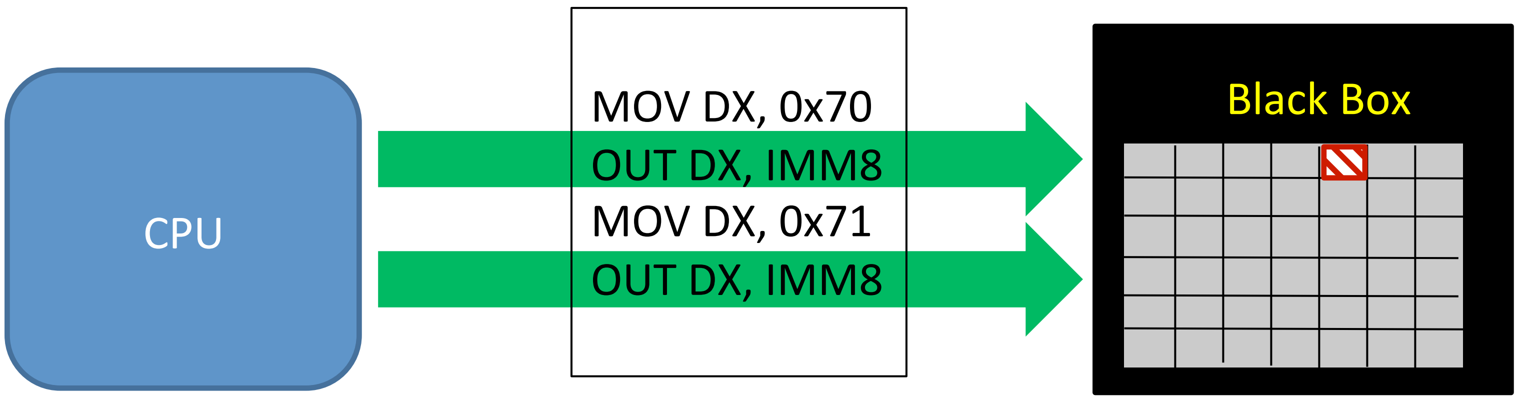
```
MOV AL, 0x05
MOV DX, 0x70
OUT DX, AL
MOV DX, 0x71
MOV EAX, 0x10
OUT DX, EAX
```

Port IO



- IMM8 (one byte constant) could be a command or data – that's up to the interpretation by the device
- It is not necessarily known what the black box on the end of a port does
- Check your Controller Hub datasheet and/or the LPC decode registers (might offer clues)

Port IO Index/Data Pair



- Some devices use an index/data pair for IO
- An offset is written to the index port
- Next a value is read from or written to that offset from the Data port
- Devices such as this are CMOS, PCI, and the Keyboard Embedded Controller on the E6400 (per the below research)
  - http://esec-lab.sogeti.com/dotclear/public/publications/11-recon-stickyfingers_slides.pdf

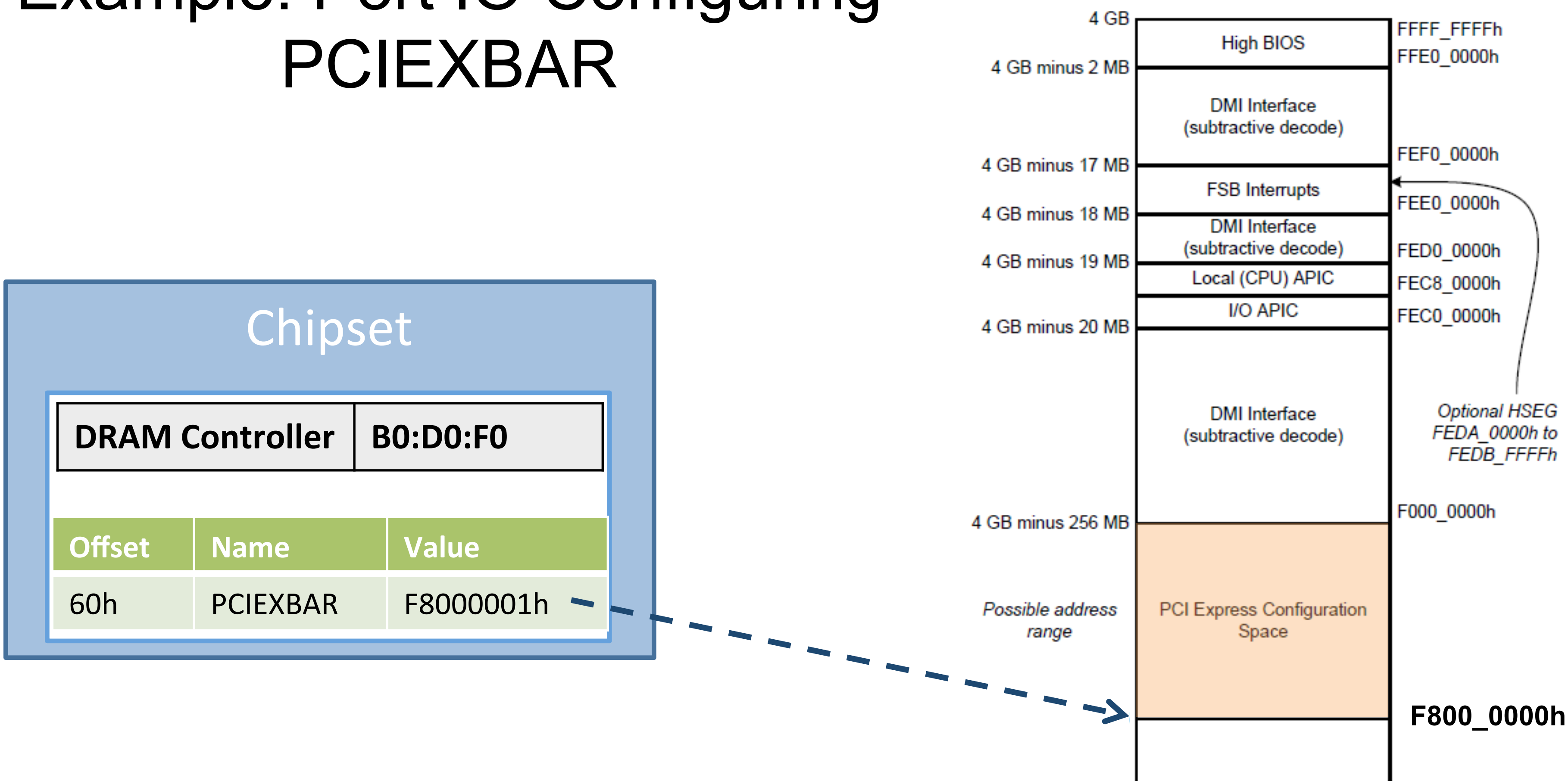
Identifying Port I/O

- First try deciphering port IO devices by using the datasheets (Controller Hub either ICH or PCH)
- OS Dev
  - http://wiki.osdev.org/I/O_Ports (which links you to...)
- Boch's or Ralf's
  - http://bochs.sourceforge.net/techspec/PORTS.LST
  - Last change was in 11/6/94 and that's just how it is with most BIOS information
- Vendors can extend a device interface to any unoccupied IO address

UEFI indirection

- When we eventually get to UEFI you will see that there's a lot of indirection.
- So this is just to say that if you were REing some code, while you might eventually find the actual IN/OUT instructions, it would suffice to find "**EFI_CPU_IO2_PROTOCOL.Io.Read() and Io.Write()**" which are functionally equivalent
  - You can read more about them in the UEFI specs' Volume 5

Example: Port IO Configuring PCIEXBAR



- On the Mobile 4-Series Chipset, the BIOS (executed by the CPU), configures the PCIEXBAR in the DRAM Controller
- F800_0000h (on an E6400 with 4GB RAM for example)
- PCI Memory range is now mapped
- So how does this configuration actually occur? PCI…