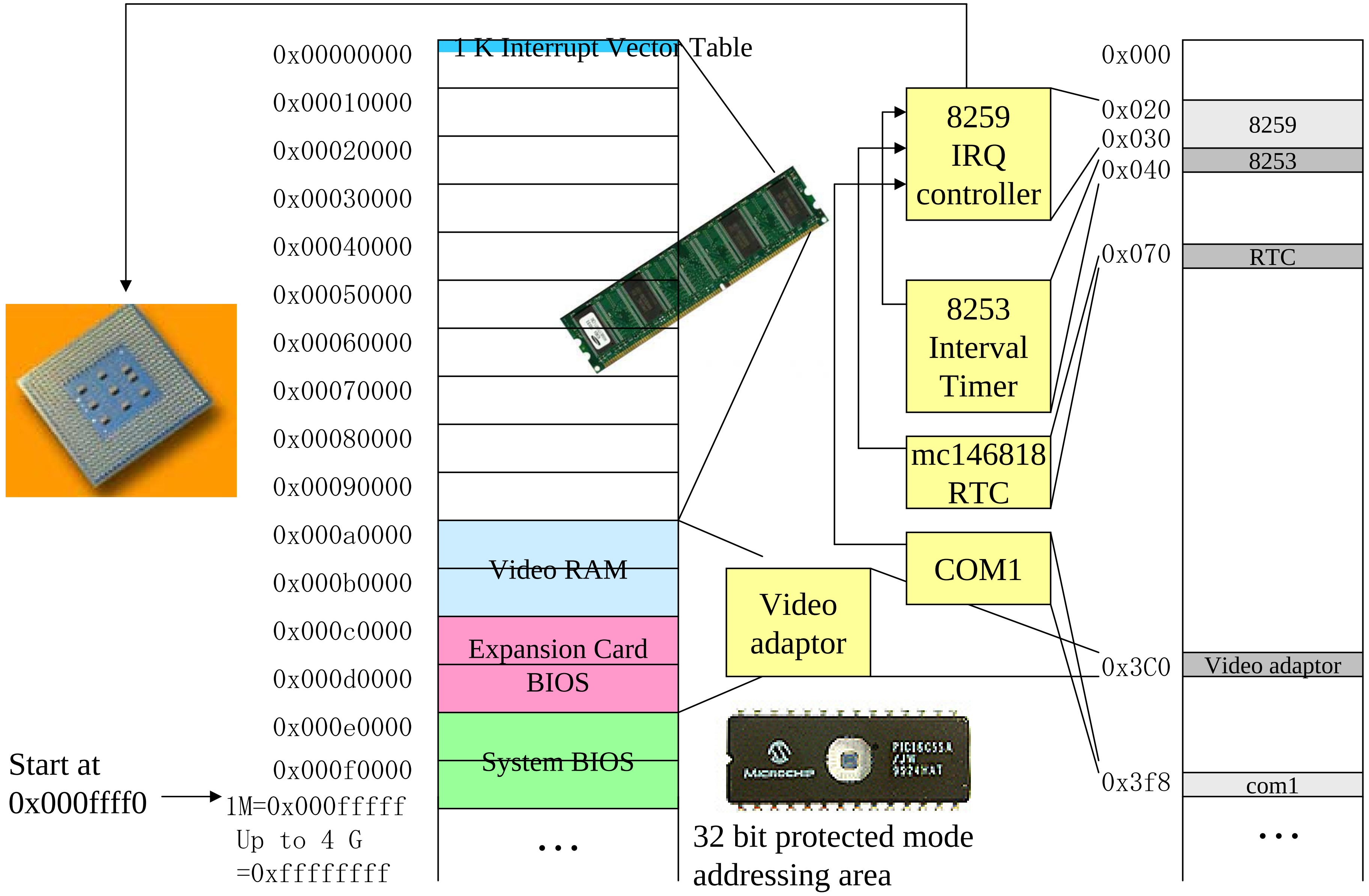
Traditional DIY approach

DIY approach

- Do everything by yourself
  - Design hardware
    - CPU, Peripherals
    - I/O and Memory address mapping
  - Entire software
    - ROM Bios or Monitor program
    - Interrupt processing
    - Application functions
    - In charge of entire CPU control flow

HW Config. (x86 PC)



References

- Detail mapping and control is each component specific
- For more details, refer
  - Memory map:
    http://www.osdata.com/system/physical/memmap.htm
    http://savage.net.au/Ron/html/hex-ram-tutorial.html
  - Port map:
    http://members.tripod.com/~oldboard/assembly/ports.html
  - CMOS RTC:
    http://members.tripod.com/~oldboard/assembly/cmos_ram.html
    http://sunsite.lanet.lv/ftp/mirror/x2ftp/msdos/programming/serial/rtc
  - VGA Adaptor:
    http://www.osdever.net/FreeVGA/vga/vga.htm

Boot sequence (PC)

- Power on: processor jumps to 0xffff0
- Jumps to ROM BIOS start point
- Check and initialize hardware
- Retrieve and/or store BIOS data (e.g, boot device order)
- Load the Master Boot Sector from the primary boot device
- Jump to the loaded boot code
- Now OS specific (Boot loader, DOS, Windows, Linux)

Boot sequence
(Embedded system)

- Power on: processor jumps to a specific address depending on microprocessor
- Jump to monitor ROM start point
- Check and initialize hardware
- Set the interrupt vector table with corresponding handlers
- Start executing a "super loop"
- Everything is managed inside of the super loop during the system life time

Our first embedded real-time system (stepping motor controller)

- Super loop approach with Real-Time Clock support
  - Initialize mc146818rtc so that it generates periodic interrupts
  - Register RTC (IRQ8) interrupt handler
  - Super loop keeps doing
    - if an event happens, turn the motor one step

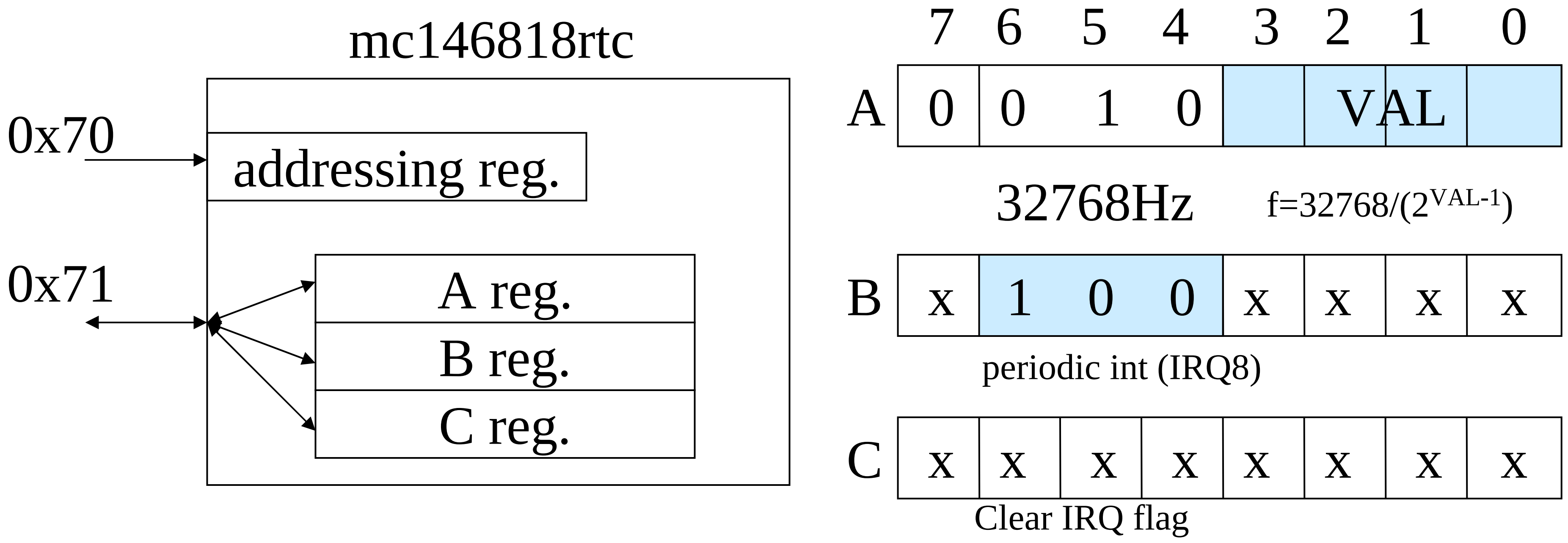
External Interrupts (IRQs)

INT no.	IRQ no.	Function
0-7h	CPU	Divide error, etc.
8h	0	System Timer 8253
9h	1	Keyboard
Ah	2	IRQ Cascading
Bh	3	COM2
Ch	4	COM1
Dh	5	Hard disk
Eh	6	Floppy disk
Fh	7	Printer
10-67h	BIOS	BIOS or OS
70h	8	CMOS RTC
71h	9	General use
72h	10	General use
73h	11	General use
74h	12	Mouse port
75h	13	Math co-proc.
76h	14	Primary Hard disk
77h	15	General use

Ref. The undocumented PC: A Programmer’s guide to I/O, CPUs, and Fixed Memory Areas

Peripheral device control (ex, mc146818rtc)

- IO mapped IO vs Memory mapped IO
- outp_b(byteValue, IOport), byteVariable = inp_b(IOport)
  - macro using inline assembly “out” and “in”



Register Interrupt Handler
(ex. CMOS RTC interrupt handler)

- Interrupt vector table access (DOS: getvect, setvect)
  - Direct access to the table entry address
- In our linux environment (32 bit protected mode, virtual address), direct access is difficult
- Let’s get help from RtLinux functions
    - #include <rtl_core.h>
    - int rtl_request_irq(unsigned int irq, unsigned int (*handler)());
    - int rtl_free_irq(unsigned int irq);
    - int rtl_hard_enable_irq(unsigned int irq);
    - int rtl_hard_disable_irq(unsigned int irq);
- Interrupt handler
    - unsigned int handler(unsigned int irq, struct pt_regs *regs);
    - Increase a counter and set EVENT if one step time has been passed

Super Loop

- Infinite loop – while(1){}
- Check if the EVENT (one step time) happened
- If so, call “motor(id, x, y)” and clear EVENT

How to take the full control in Linux environment?

- Make our super loop as a kernel module
- insmod superloop.o
- Entry and Exit function of a kernel module
  - int init_module(void)
  - int cleanup_module(void)
- Inside of init_module, implement the super loop
- The super loop will take the full control
- Anything else (such as linux service) cannot be done
- To kill your module Reboot the system (Sorry ^--^)
  - So… please “sync” first before inserting superloop.o to avoid loss of your data

Project assignment

- Using the super loop approach, implement a system that controls two motors: 1) the first motor should take one turn (4 steps) at every 4 sec, and 2) the second motor should take four turns at every 1sec. Run it in text-mode. The program should print the number of turns of each motor at every 10 sec. Wait for 500 sec (measure 500 sec with your watch) and see the number of turns matches with your expectation. Discuss about it!