Exam 3 is ongoing.

Interrupts and Exceptions

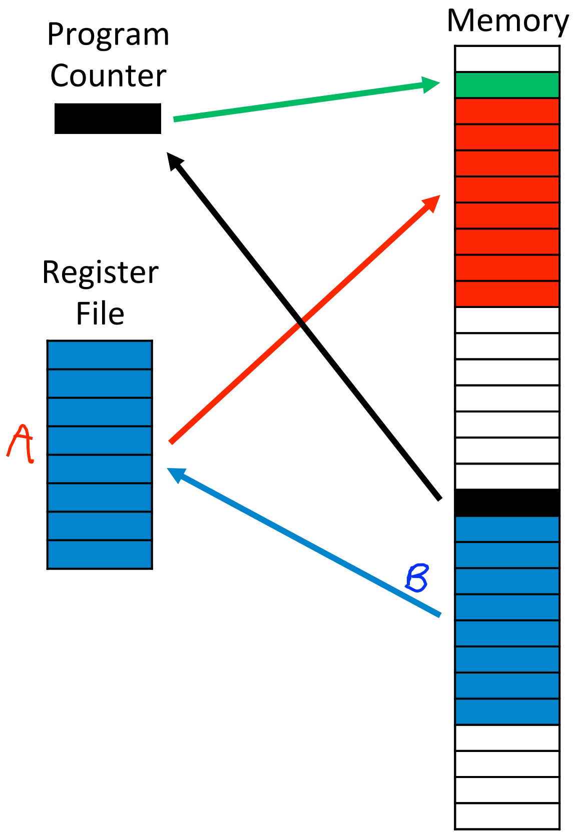
Today's lecture

- Use addressing to get data from the outside world
  - Data is moved from peripherals to memory
  - Addressing schemes
    - Memory-mapped vs. isolated I/O
  - Data movement schemes
    - Programmed I/O vs. Interrupt-driven I/O vs. Direct memory access

Most modern operating systems pre-emptively schedule programs

- If a computer is running two programs A and B, the O/S will periodically switch between them
  1. Stop A from running
  2. Copy A's register values to memory
  3. Copy B's register values from memory
  4. Start B running

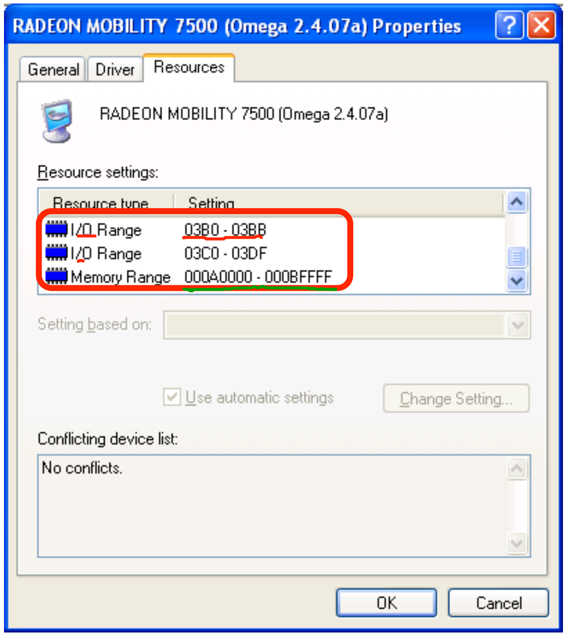
How does the O/S stop program A?



We can treat most devices “as if” they were memory with an “address” for reading/writing

- Many ISAs often make this analogy explicit — to transfer data to/from a particular device, the CPU can access special addresses

- *Example*: Video card can be accessed via addresses 3B0-3BB, 3C0-3DF and A0000-BFFFF



RADEON MOBILITY 7500 (Omega 2.4.07a) Properties

RADEON MOBILITY 7500 (Omega 2.4.07a)

Resource settings:

Resource type	Setting
I/O Range	03B0 - 03BB
I/O Range	03C0 - 03DF
Memory Range	000A0000 - 000BFFFF

Setting based on:

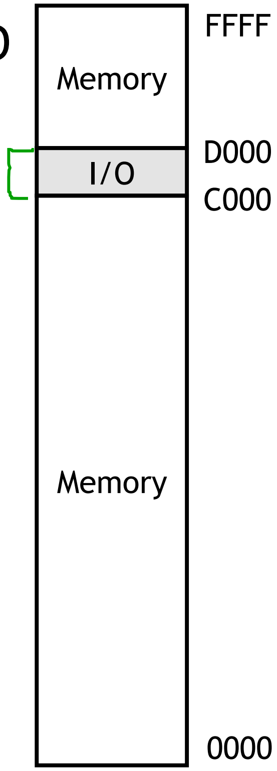
☑ Use automatic settings

Conflicting device list:

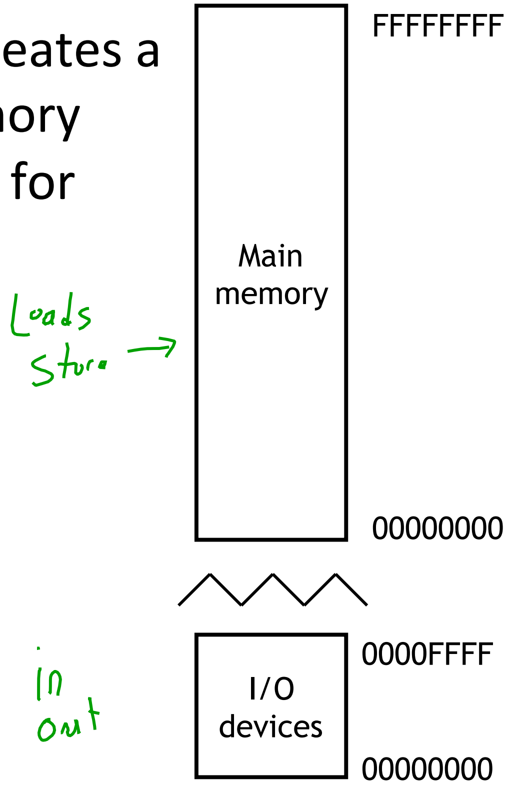
No conflicts.

Most ISAs one of two protocols for addressing devices: memory-mapped I/O or isolated I/O

Memory-mapped I/O reserves a portion of main memory addresses for I/O

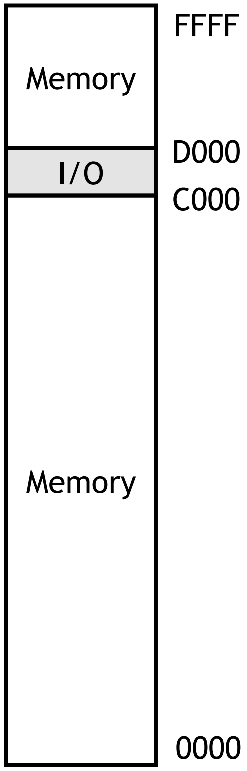


Isolated I/O creates a separate memory address space for devices

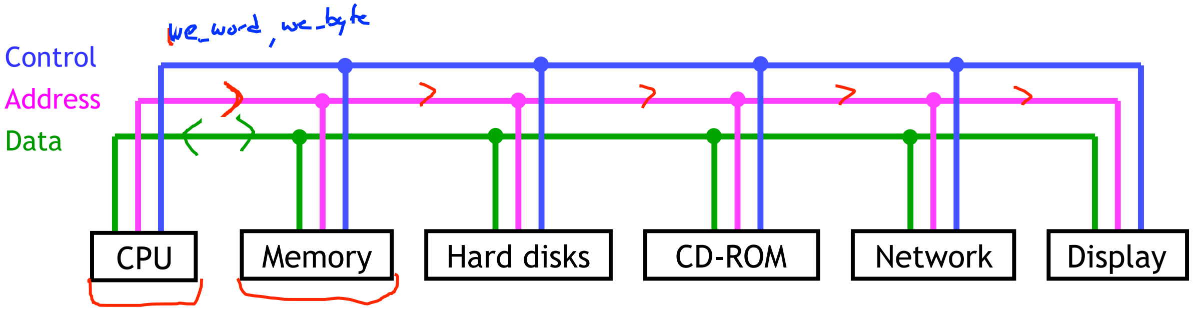


Memory-mapped I/O divides main memory addresses into actual memory and devices

- Apple IIe (right) had a 16-bit address bus
  - Addresses C000-CFFF accessed I/O devices.
  - No actual main memory at C000-CFFF
  - All other addresses reference main memory.
- I/O addresses are shared by many peripherals.
  - C010 → keyboard
  - C030 → speaker
- Some devices may need several I/O addresses.



We use control and addressing to determine when data goes to memory or devices



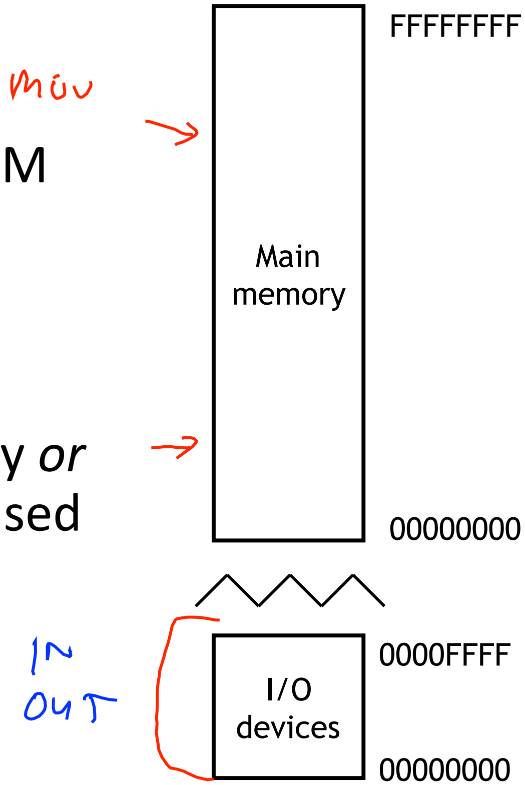
- Each device has to monitor the address bus to see if it is the target. (Apple IIe example)
  - Main memory ignores any transactions with addresses C000-CFFF.
  - The speaker only responds when C030 appears on the address bus.

Store

Isolated I/O creates two separate address spaces and needs two sets of instructions

- *Example* (x86): Ld /st
  - regular instructions like MOV reference RAM
  - special instructions IN and OUT access a separate I/O address space

- An address could refer to *either* main memory *or* an I/O device, depending on the instruction used



iclicker

MIPS provides the following instructions for managing memory: `load word`, `load halfword`, `load byte`, `store word`, `store halfword` and `store byte`.

Which I/O addressing method does MIPS use?

a) Memory-mapped I/O

b) Isolated I/O

MIPS/SPIMbot uses memory-mapped I/O

- Examples

```
lw     $reg, 0xffff0020($0)   # gets SPIMbot x-coord

sw     $reg, 0xffff0010($0)   # sets bot speed = $reg
                                             -10 .. +10
```

- Some control commands require a sequence of instructions

```
                angle in degrees
sw     $reg, 0xffff0014($0)
li     $t0, 1
sw     $t0, 0xffff0018        # sets bot angle = $reg
```

Example SPIMbot commands

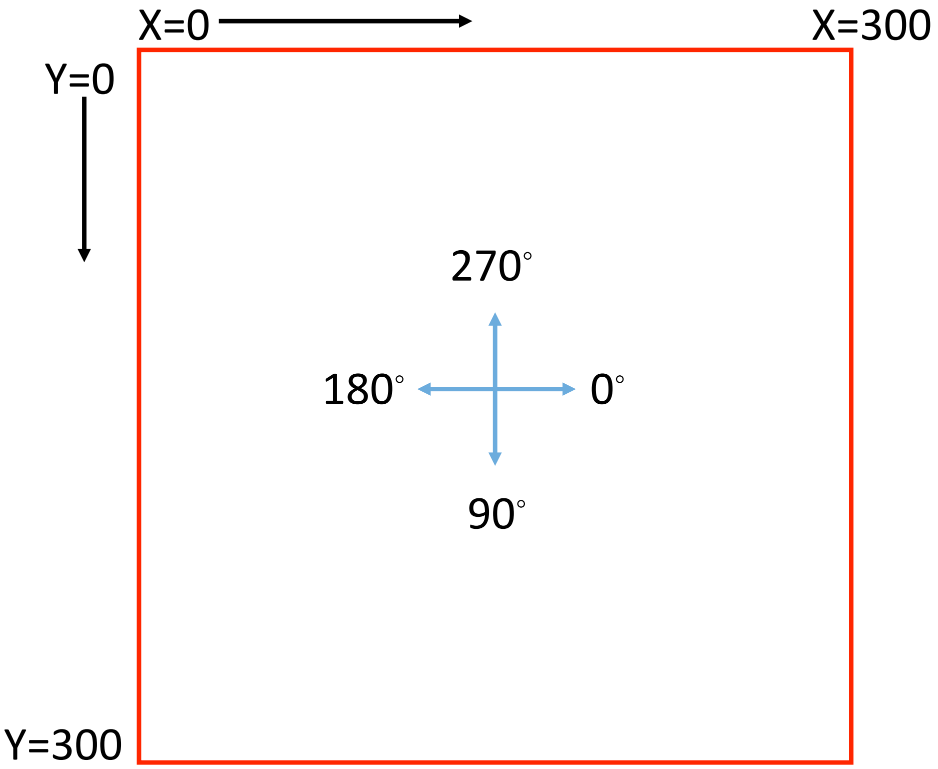
What	How

get SPIMbot's current x/y-coordinate	`lw` from `0xffff0020 (x)`
	`lw` from `0xffff0024 (y)`

set SPIMbot's angle (absolute)	`sw` the angle to `0xffff0014`
	`sw 1` to `0xffff0018`

set SPIMbot's angle (relative)	`sw` the angle to `0xffff0014` delta `sw 0` to `0xffff0018`
set SPIMbot's velocity	`sw` a number between `-10` and `10` to `0xffff0010`
read the current time	`lw` from `0xffff001c`
request a timer interrupt	`sw` the desired (future) time to `0xffff001c`
acknowledge a bonk interrupt	`sw` any value to `0xffff0060`
acknowledge a timer interrupt	`sw` any value to `0xffff006c`

SPIMbot coordinate system


X=0
X=300
Y=0
270°
180°
0°
90°
Y=300