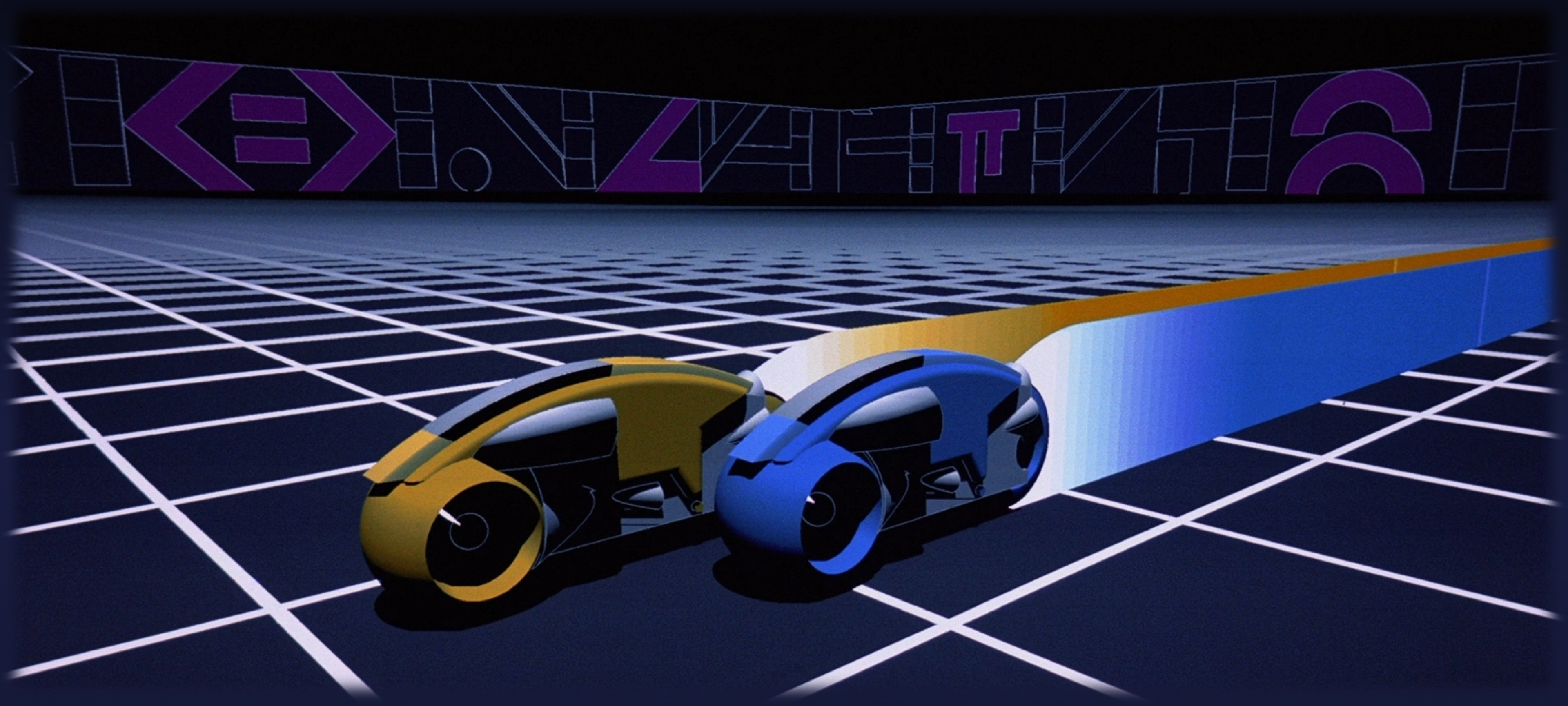
Lecture 11: Wrap-up and Farewell

We're Almost Done

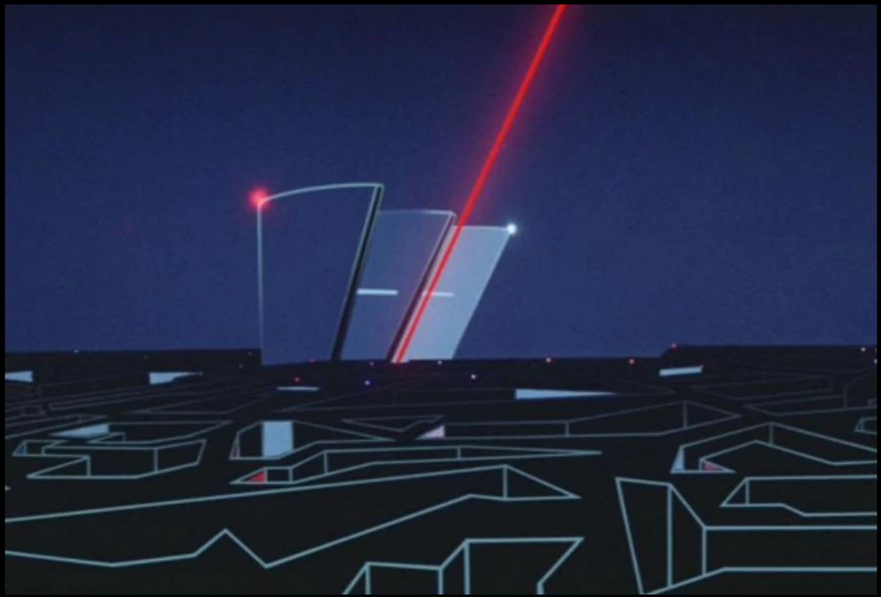
- We've covered
  - Arithmetic and logical operations
  - Branches for loops and conditions
  - Memory
  - Functions
  - Stack
  - Calling conventions

Talking To Hardware



Input and Output

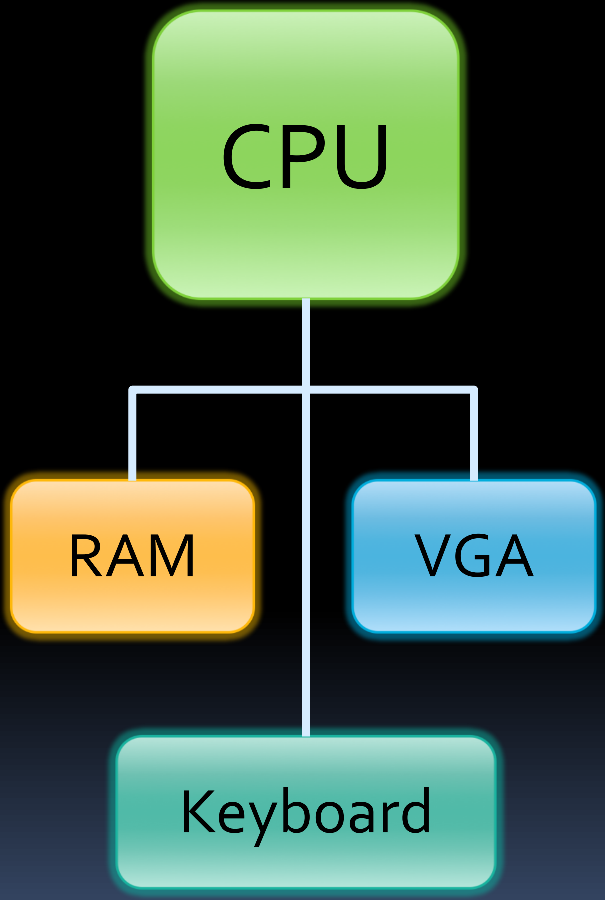
- There is a world outside the CPU
  - VGA
  - Hard drives
  - Keyboard, mouse
  - Network cards
  - etc.



- How do we communicate with this hardware?
- How do we do I/O (Input/Output)?

Memory Mapped I/O

- Certain memory addressed don't go to RAM.
- Instead, they go to device registers.
  - Write to control a device.
  - Read to get data or device status.
- Often works with polling:
  - Example: to know if an operation is finished, we read memory in a loop until status is "finished".



Interrupts

- Rather than polling, devices can interrupt the processor to signal important status
  - Operation completed, error, and so on.
- Interrupts are special signals that go from devices to the CPU.
- When an interrupt occurs, the CPU stops what it is doing and jumps to an interrupt handler routine
  - This routine handles the interrupt and returns to the original code.

Handling Interrupts

- Polled handling (not related to previous polling):
  - CPU branches to generic handler code for all exceptions.
  - Handler checks the cause of the exception and branches to specific code depending on the type of exception.
  - This is what MIPS uses.
- Vectored handling:
  - We first assign a unique id (number) for each device and interrupt/exception type (example from 0 to 255).
  - We set up a table containing the address of the specific interrupt handler for every possible id.
  - On interrupt with type X, the CPU gets the address from row X of the table and branches to the address.
  - This is what x86 uses.

Exceptions

- An exception is like an interrupt that comes from inside the CPU.
  - The mechanism is similar, the difference is semantic.

- Reasons for interrupts/exceptions:
  - Device I/O (interrupt)
  - Invalid instruction (can't decode!)
  - Arithmetic overflow (add with overflow).
  - Divide by zero.
  - System calls (also called traps)

(Invalid instruction, arithmetic overflow, divide by zero, system calls: exceptions)

MIPS Interrupt Handling

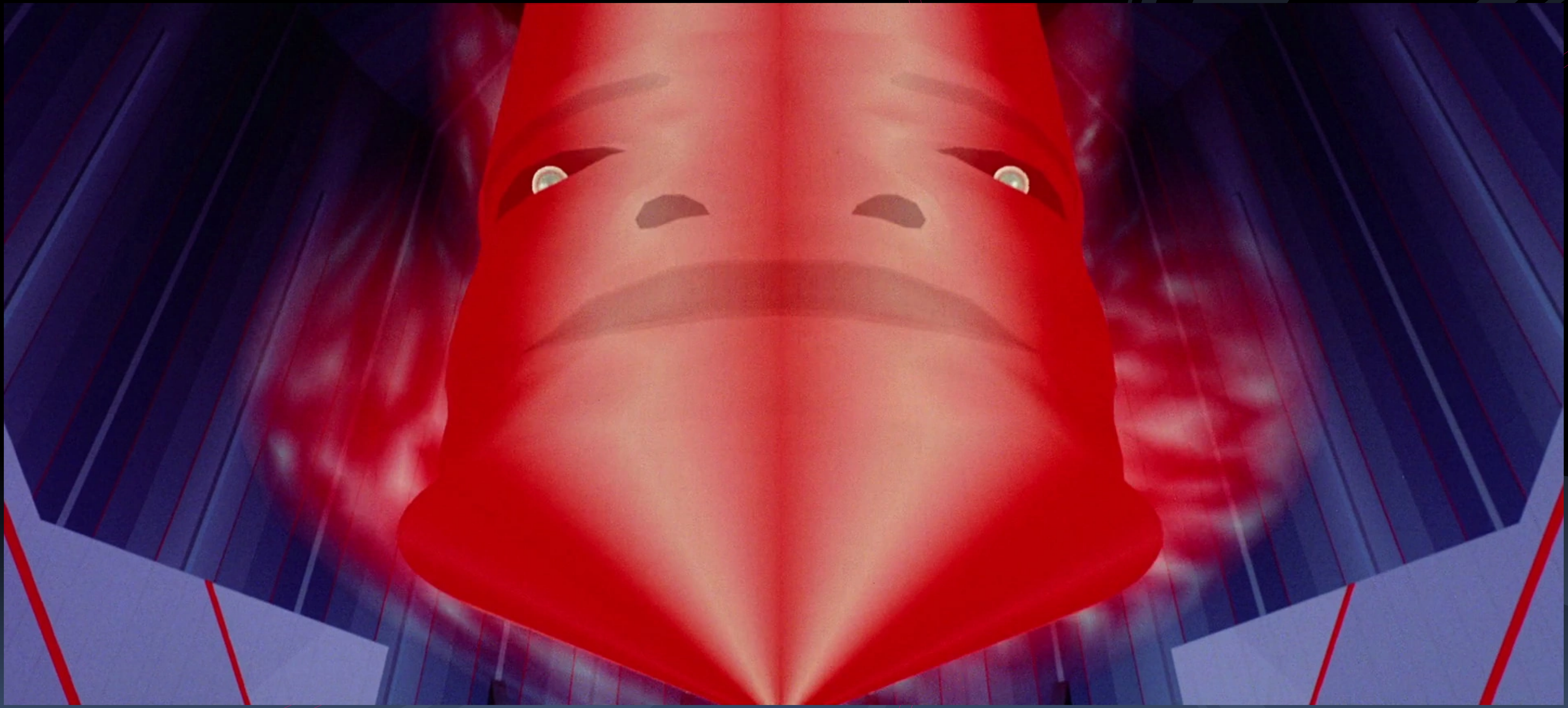
- MIPS has polled interrupt handling: the processor jumps to exception handler code, based on the value in the cause register (see table).
- If the original program can resume afterwards, this interrupt handler returns to program by calling `rfe` instruction.
- Alternatively, the OS terminates the program.
  - For example, dump the contents of the stack to disk or screen to help debugging.

0 (INT)	external interrupt.
4 (ADDRL)	address error exception (load or fetch)
5 (ADDRS)	address error exception (store).
6 (IBUS)	bus error on instruction fetch.
7 (DBUS)	bus error on data fetch
8 (Syscall)	Syscall exception
9 (BKPT)	Breakpoint exception
10 (RI)	Reserved Instruction exception
12 (OVF)	Arithmetic overflow exception

Coordination

- Talking with hardware is a lot of work.
  - What if you change your hardware, do you need to change every program?
  - Should we duplicate code (e.g., for handling keyboard) in every program that needs it?
- Who will manage all the different programs on the computer and offer them I/O services?
- We need some sort of master control program to coordinate all this…

The Operating System



The Operating System

- The operating system is the program that manages all the other programs.
  - Loading, running, and stopping programs.
  - Running multiple programs simultaneously.
  - It abstracts hardware and I/O, and offers services.
- Programs invoke the OS to do things like:
  - Read/write from files.
  - Write to screen.
  - Run other programs

Learn more in C69

- Invoking the OS is done via system calls or traps.

Instruction	Function	Syntax
`syscall`	001100 (R-type)	I

- Trap instructions send system calls to the operating system
  - e.g. interacting with the user, and exiting the program.
  - Trap code goes in `$v0`
- These are services offered by SPIM.

SPIM Service	Trap Code	Input/Output
print_int	1	$ao is int to print
print_string	4	$ao is address of ASCIIZ string to print
read_int	5	$vo is int read
read_string	8	$ao is address of buffer $a1 is buffer size in bytes
exit	10	
open_file	13	$ao is address of ASCIIZ string containing file name $a1 is flag $a2 is mode $vo is file descriptor
read_from_file	14	$ao is file descriptor $a1 is address of input buffer $a2 is number of characters to read
write_to_file	15	$ao is file descriptor $a1 is address of output buffer $a2 is number of characters to write
close_file	16	$ao is file descriptor

Example

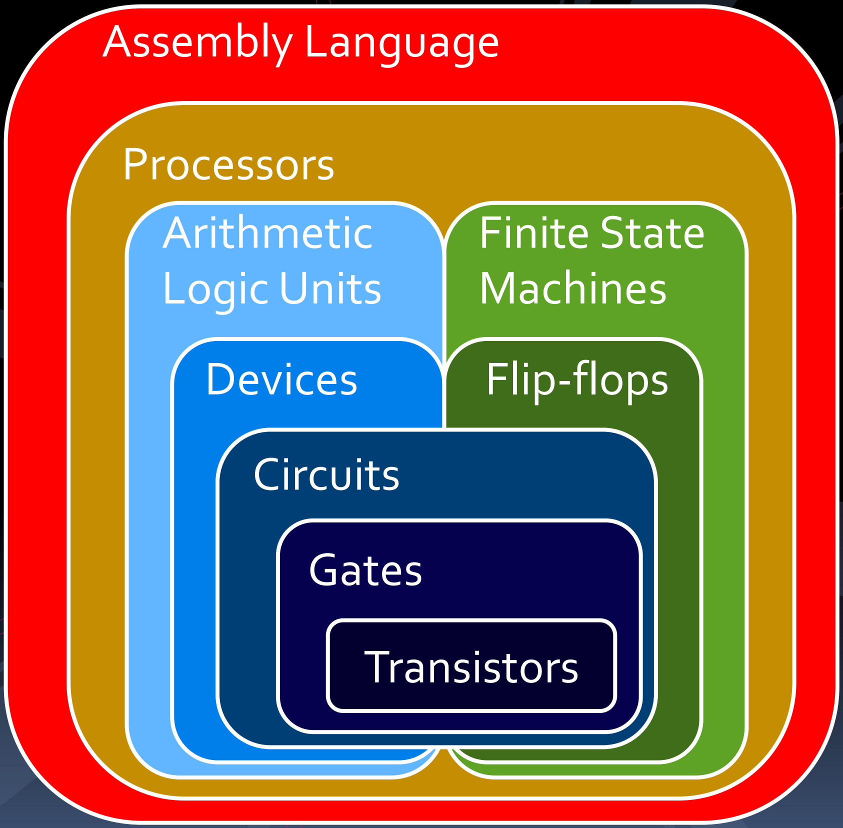
```
.data
var1:  .word  10
str1:  .ascii "Hello World"

.text
main:  li $v0, 1              # print the number stored in var1
       la $t0, var1
       lw $a0, 0($t0)
       syscall

       li $v0, 4              # print the string stored in str1
       la $a0, str1
       syscall

end:   li $v0, 10             # exit the running program
       syscall
```

One More Thing

- Calling future TAs!
  - Want to TA B58?
  - Looking for Verilog experience (small candidate pool)
  - You can help improve/shape the future of the course.
  - Ask your current TAs what they think!
- Course Evaluations
  - They are anonymous.
  - I do actually read them.
  - I do actually care.
  - They do actually make an impact.
  - No, I won’t bribe you

Cake Courtesy of Sophie Harrington

WE ARE DONE!



Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Gates
Transistors

Given enough silicon, phosphorus and boron, you are now able to build a computer!

Good luck!

Thank you all!