

CMPE 200  
Computer Architecture & Design

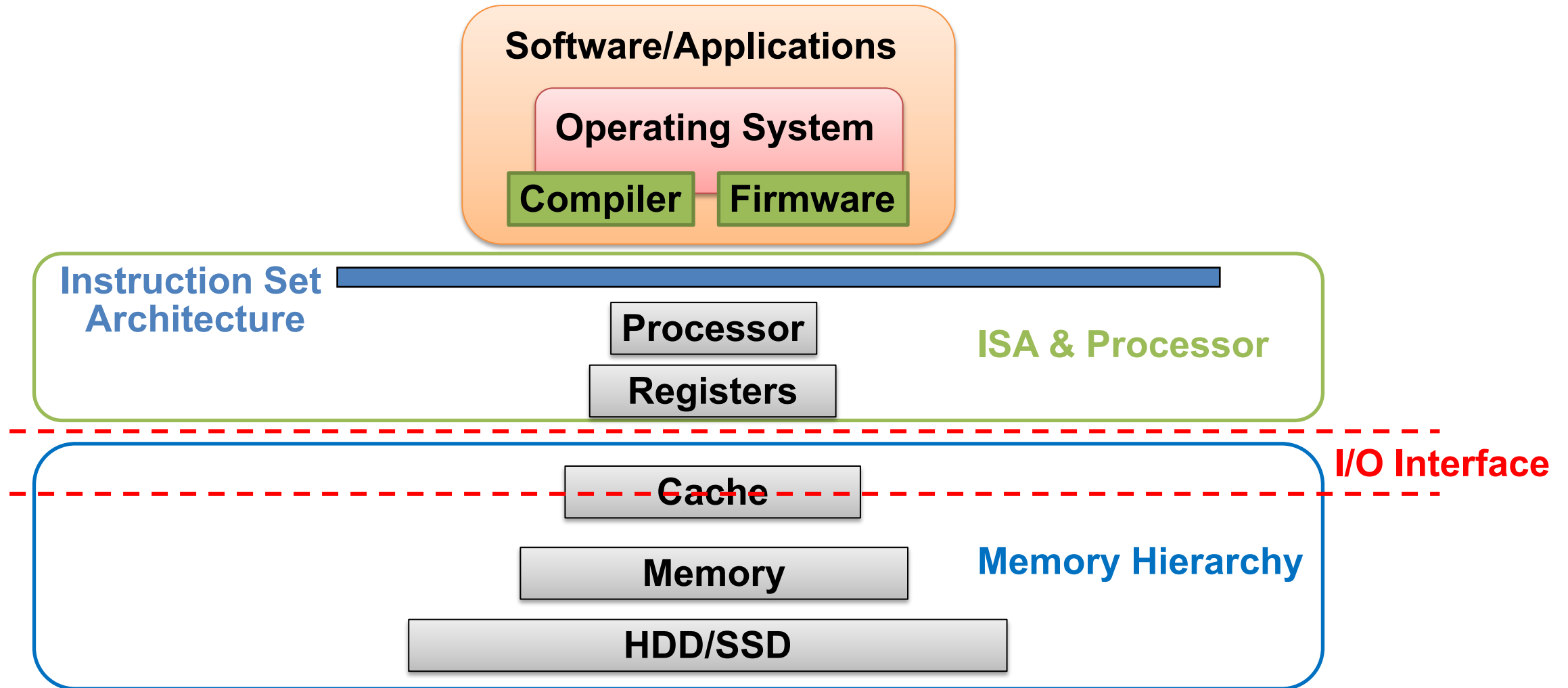
## **Lecture 3.5.** **I/O and Interface**

Haonan Wang



SAN JOSÉ STATE  
UNIVERSITY

# Computer Architecture Overview



# Peripheral Devices

Device	Behavior	Partner	Protocol	Data rate (Mb/sec)
Keyboard	input	human	USB, PS/2	0.0001
Mouse	input	human	USB, PS/2	0.0038
Laser printer	output	human	Parallel Port	3.20
Graphics display	output	human	DP, HDMI	8000-80000
Graphics card	accelerator	machine	PCIE	2000-8000
Network/LAN	input or output	machine	TCP/IP	10-10000
Magnetic disk	storage	machine	SATA	240-2560
Solid state drive	storage	machine	SATA, PCIE	4000-28000

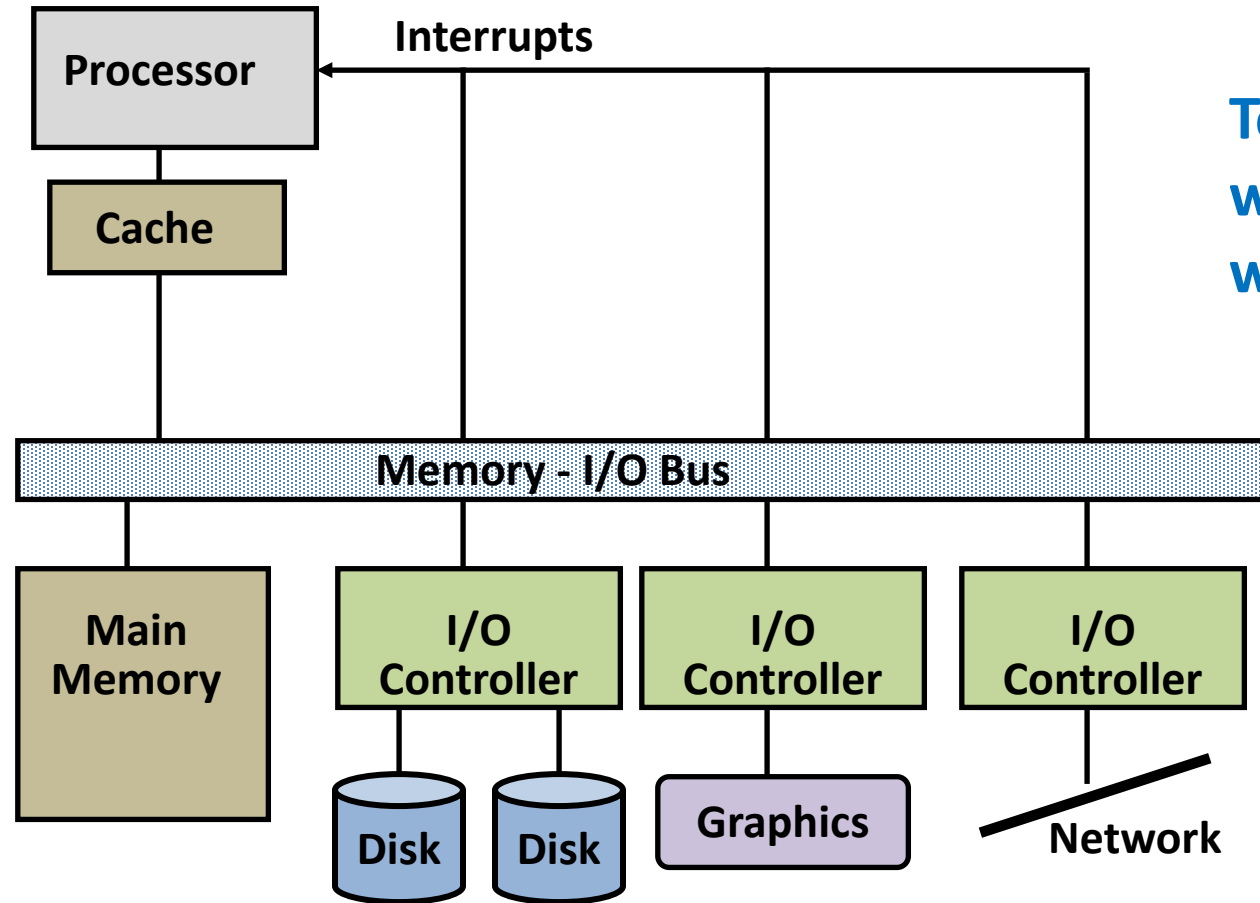
9 orders of magnitude  
range

# I/O Performance Measures

- **I/O bandwidth (throughput)**
  - How much data can we move through the system in a certain time?
  - How many I/O operations can we do per unit time?
- **I/O response time (latency)**
  - The total elapsed time to accomplish an input or output operation
  - Especially important in real-time systems
- **Many applications require both high throughput and short response times**

**Both performance and fairness matters!**

# A Typical I/O System



To interface a peripheral device with a processor, an interface wrapper (e.g., controller) is needed

Important factors:

- Performance
- Expandability
- Resilience
- ...

# Interface Wrappers

- **Contains registers to hold signals (control, data, and status) that communicate between the peripheral device and the processor**
  - memory mapped: each interface register can be accessed via an address
- **The interface wrapper usually contains additional logics (such as address decoder)**
  - If the interface protocol is more complicated, the interface wrapper will also contain control unit (FSM)

# Processor Interface Mechanisms

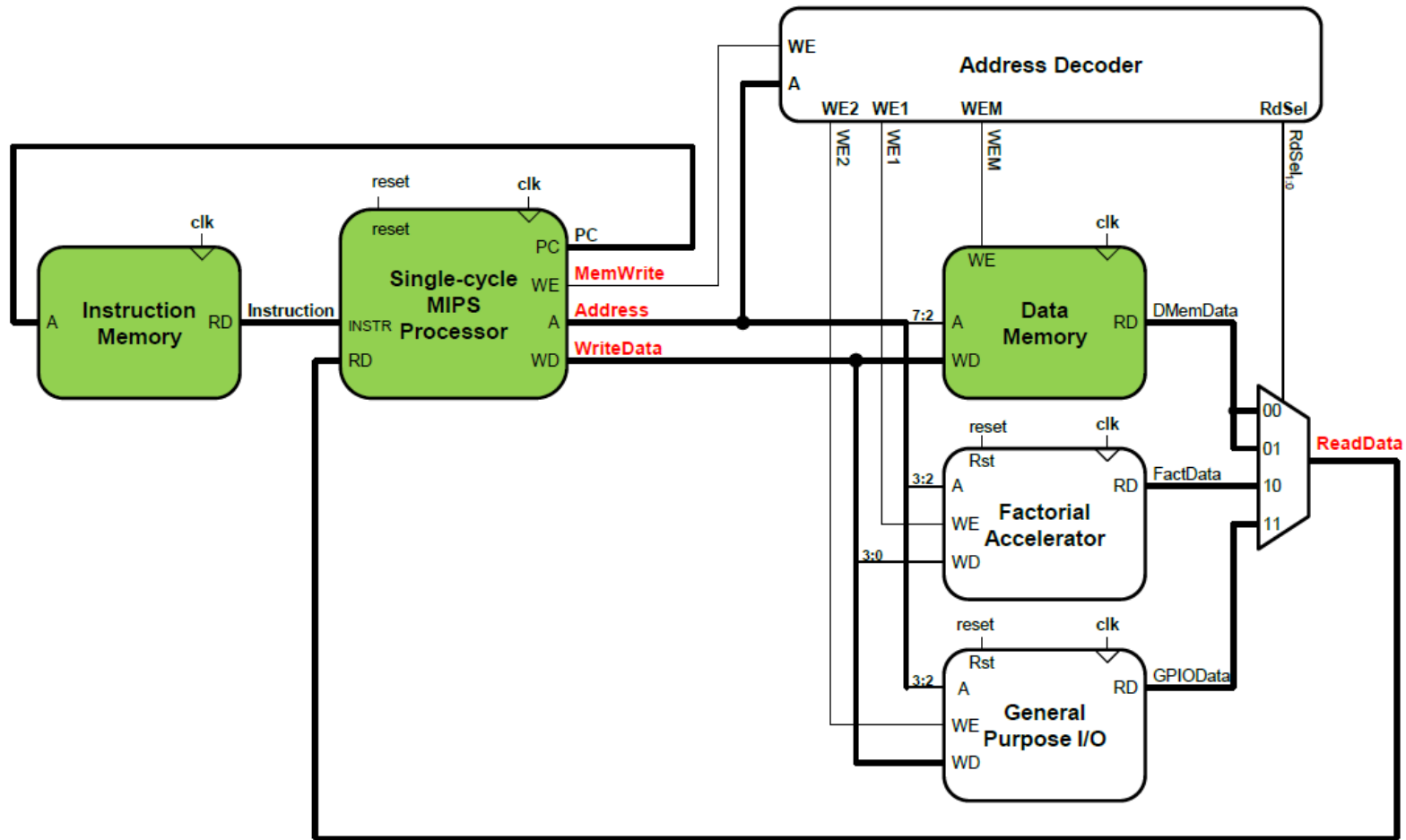
- **Two ways for processors to access peripheral devices:**
  - Port-mapped I/O (PMIO): isolated I/O, special I/O instructions, separate address spaces, time shared bus
  - Memory-mapped I/O (MMIO): one address space
- **MIPS processors use memory-mapped I/O**
  - Use load and store instructions to communicate with peripheral devices
  - For a MIPS processor that only supports lw and sw, all data transfer will be 32 bits

# Processor Interface Mechanisms

- **Two ways to communicate with CPU:**
  - Polling: The CPU polls the device periodically.
    - Simple but time consuming
  - Interrupt: The I/O device calls the CPU actively
    - Fast but complex: which interrupt has high priority?
- **Two ways for data transfer between the memory and I/O:**
  - CPU controlled
  - Direct Memory Access (DMA)



# Memory-Mapped I/O Datapath



# Memory-Mapped I/O Example

System Integration Memory Map

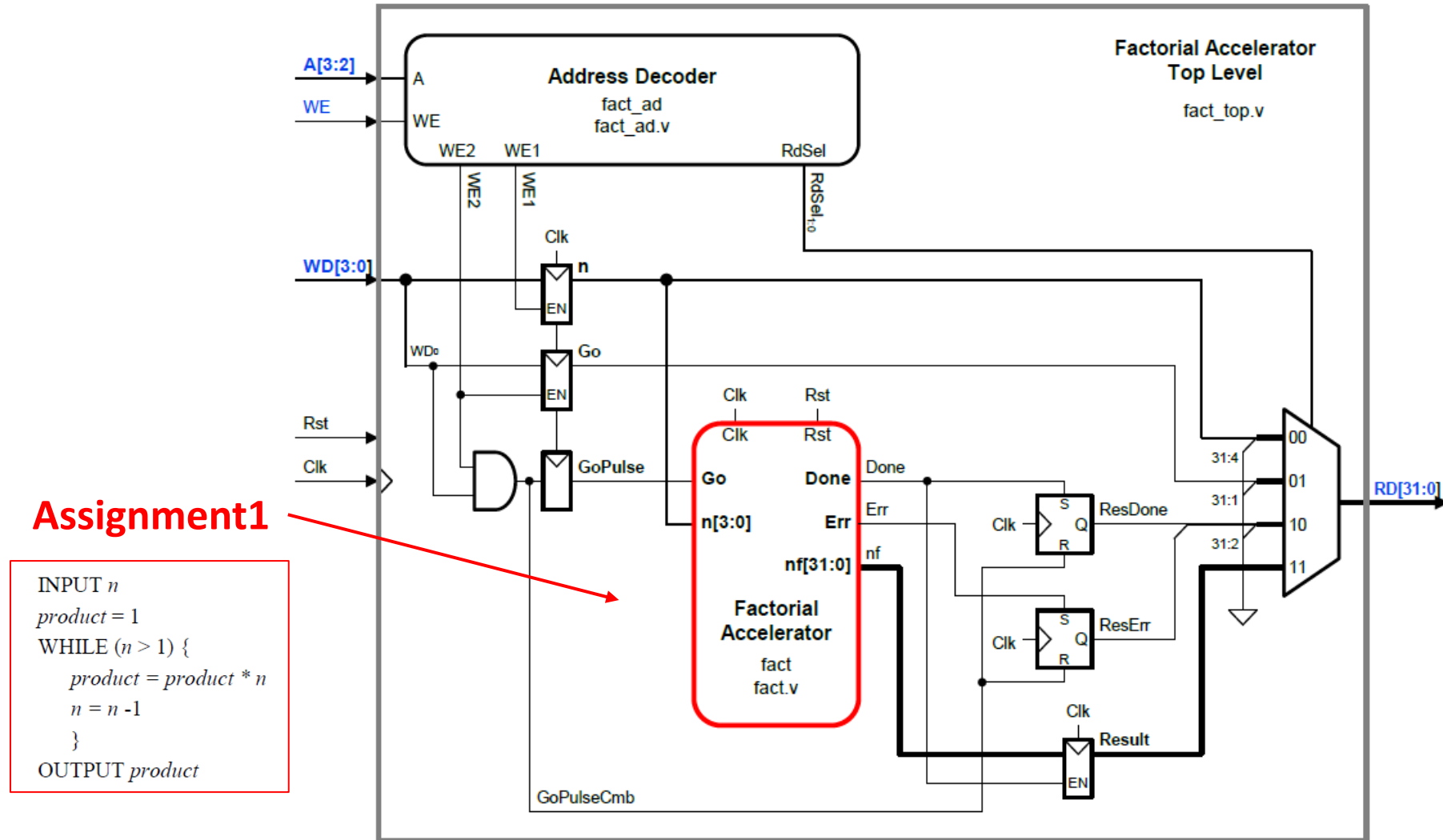
Base Address	Address Range	R/W	Description
0x00000000	0x00 – 0xFC	R/W	Data Memory
0x00000800	0x00 – 0x0C	R/W	Device 1
0x00000900	0x00 – 0x0C	R/W	Device 2

0xFC = 11111100 → 64 (0x00 – 0x3F) word addresses

0x0C = 00001100 → 4 (0x00 – 0x03) word addresses

Last two bits are byte offsets

# Factorial Accelerator Wrapper



# Factorial Accelerator Interface Registers

Address	R/W	Register Name	Bits	Bit Definition
00	R/W	Data Input (n)	31:4	Unused
			3:0	Input Data, n[3:0]
01	R/W	Control Input (Go)	31:1	Unused
			0	Go bit
10	R	Control Output (Done, Err)	31:2	Unused
			1	Err bit
			0	Done bit
11	R	Data Output (Result)	31:0	Result Data, nf[31:0]

# Factorial Accelerator Interface (1)

- The factorial accelerator's local address decoder module:

```
module fact_ad (  
    input wire [1:0] A,  
    input wire      WE,  
    output reg      WE1,  
    output reg      WE2,  
    output wire [1:0] RdSel );  
always @ (*) begin  
    case (A)  
        2'b00: begin  
            WE1 = WE;  
            WE2 = 1'b0;  
        end  
        2'b01: begin  
            WE1 = 1'b0;  
            WE2 = WE;  
        end  
    end
```

```
        2'b10: begin  
            WE1 = 1'b0;  
            WE2 = 1'b0;  
        end  
        2'b11: begin  
            WE1 = 1'b0;  
            WE2 = 1'b0;  
        end  
        default: begin  
            WE1 = 1'bx;  
            WE2 = 1'bx;  
        end  
    endcase  
end  
    assign RdSel = A;  
endmodule
```

# Factorial Accelerator Interface (2)

- Registers for inputs *n*, *Go*, and output *Result*:

```
module fact_reg #(parameter w = 32) (  
    input wire      Clk, Rst,  
    input wire      [w-1:0] D,  
    input wire      Load_Reg,  
    output reg [w-1:0] Q );  
  
    always @ (posedge Clk, posedge Rst)  
        if (Rst)  
            Q <= 0;  
        else if (Load_Reg)  
            Q <= D;  
        else  
            Q <= Q;  
  
endmodule
```

# Factorial Accelerator Interface (3)

- Registers for status signals **Done**, and **Err**:

```
// Factorial Result Done Register
```

```
always @ (posedge Clk, posedge Rst)
```

```
begin
```

```
if (Rst)
```

```
    ResDone <= 1'b0;
```

```
else
```

```
    ResDone <= (~GoPulseCmb) & (Done | ResDone);
```

```
end
```

```
// Factorial Result Err Register
```

```
always @ (posedge Clk, posedge Rst)
```

```
begin
```

```
if (Rst)
```

```
    ResErr <= 1'b0;
```

```
else
```

```
    ResErr <= (~GoPulseCmb) & (Err | ResErr);
```

```
end
```

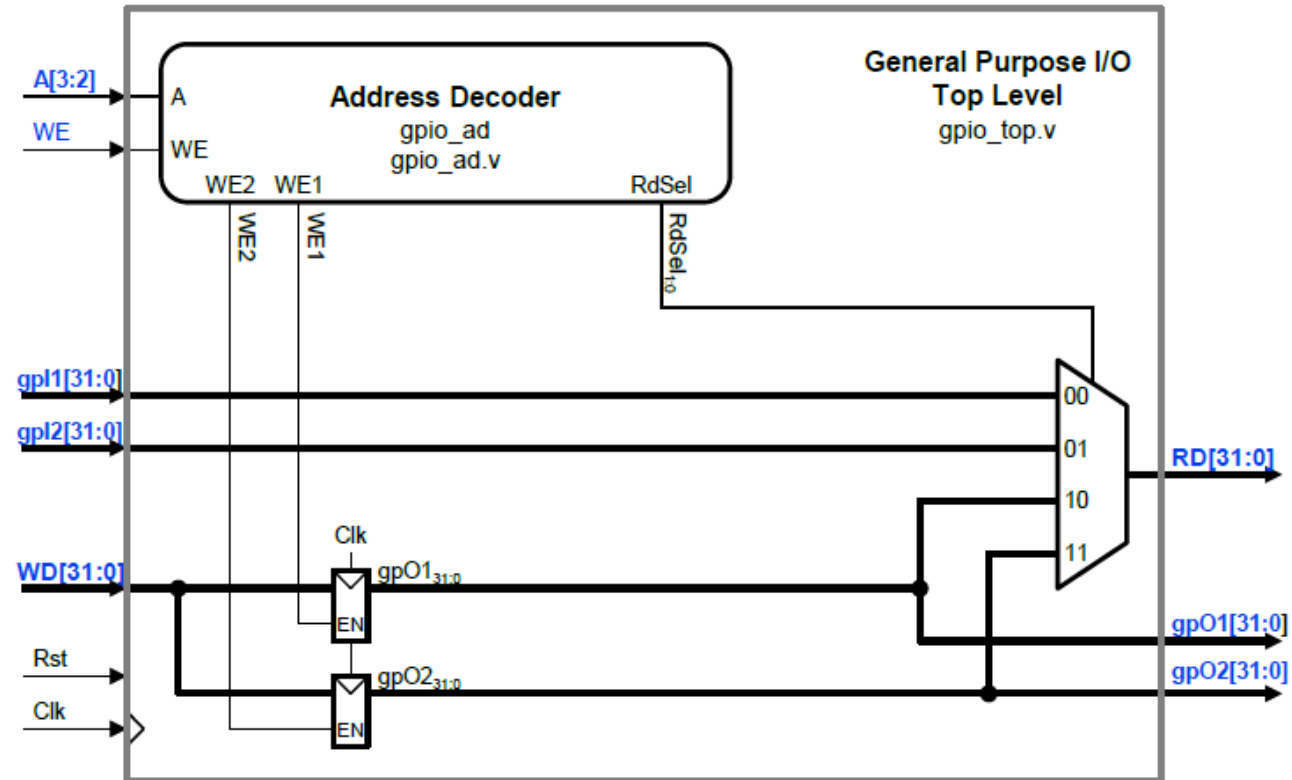
# Factorial Accelerator Interface (4)

- Read Data Logic (the output MUX):

```
always @ (*) begin
    case (RdSel)
        2'b00: RD = {{(32-nw){1'b0}}, n};
        2'b01: RD = {{31{1'b0}}, Go};
        2'b10: RD = {{30{1'b0}}, ResErr, ResDone};
        2'b11: RD = Result;
        default: RD = {31{1'bx}};
    endcase
end
```



# General Purpose I/O (GPIO) Wrapper



# GPIO Memory-Mapped Registers

Address	R/W	Register Name	Bits	Bit Definition
00	R	Input 1	31:0	General Input
01	R	Input 2	31:0	General Input
10	R/W	Output 1	31:0	General Output
11	R/W	Output 2	31:0	General Output

# MIPS Driver Code

```
main:    addi    $t0, $0,    0x0F    # $t0 = 0x0F
         addi    $t1, $0,    1       # $t1 = 1
         sll     $t4, $t1, 4       # $t4 = $t1 << 4

fact:    lw      $t2, 0x0900($0)    # read switches
         and     $t3, $t2, $t0     # get input data n
         sw      $t3, 0x0800($0)    # write input data n
         sw      $t1, 0x0804($0)    # write control Go bit

poll:    lw      $t5, 0x0808($0)    # read status Done bit
         beq     $t5, $0,    poll   # wait until Done == 1

         srl     $t5, $t5, 1       # $t5 = $t5 >> 1
         and     $t5, $t5, $t1     # get status Error bit
         and     $t3, $t2, $t4     # get display Select
         or      $t3, $t3, $t5     # combine Sel and Err
         lw      $t5, 0x080C($0)    # read result data nf
         sw      $t3, 0x0908($0)    # display Sel and Err
         sw      $t5, 0x090C($0)    # display result nf

done:    j       fact              # repeat fact loop
```

# Midterm Exam Composition

- **Multiple-choice Single-answer: 4 points \* 8 = 32**
- **Multiple-choice Multiple-answer: 6 points (wrong or missing option -3) \* 4 = 24**
- **True or False: 3 points \* 8 = 24**
- **Questions & Answers: 10 points (steps 8 + answer 2) \* 2 = 20**
- **Bonus question: 10 points \* 1 = 10**

**Total = 32 + 24 + 24 + 20 + 10 = 100 + 10**

- If you get more than 100 points, the excessive points will be used to improved your overall grade.

## **Exam settings:**

- Time: Thursday, Oct. 12, 15:00 - 16:15 @ ENG 331 (in-person only) with LockDown Browser
- Closed book (but you can print out the MIPS data card and use it if needed)
- Cheat sheet (handwritten): A4 paper \* 1 allowed, pictures of both sides must be submitted before the exam
- Use of calculator and scratch paper allowed
- Double-check your laptop and turn off your cellphone before the exam

SAN JOSÉ STATE UNIVERSITY *powering* SILICON VALLEY

