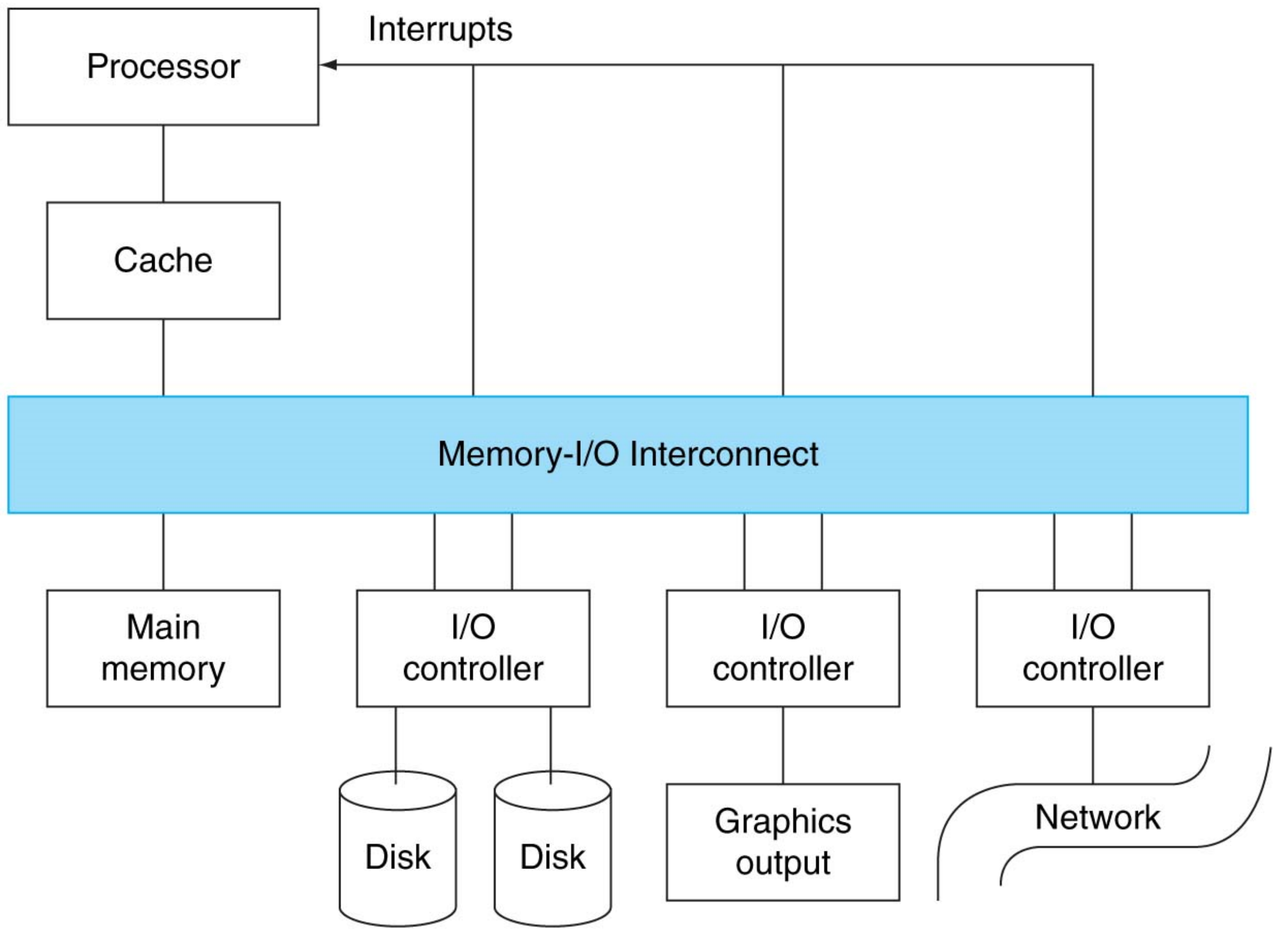
Lecture 26
Input and Output

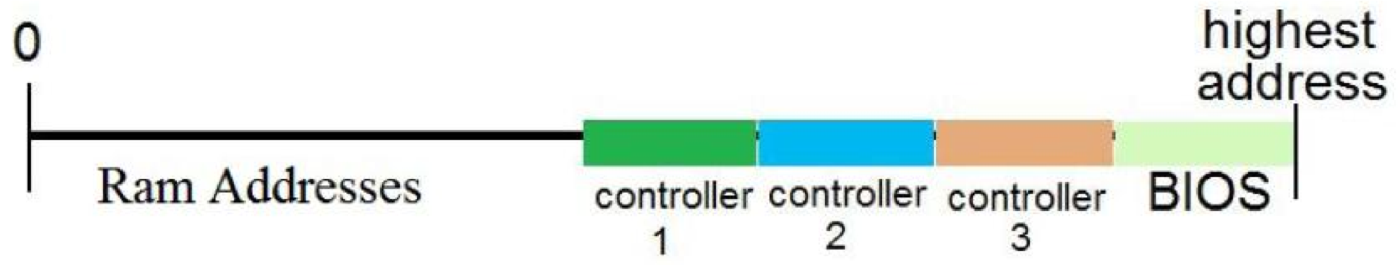
School of Computer Science and Engineering
Soongsil University

Computer System

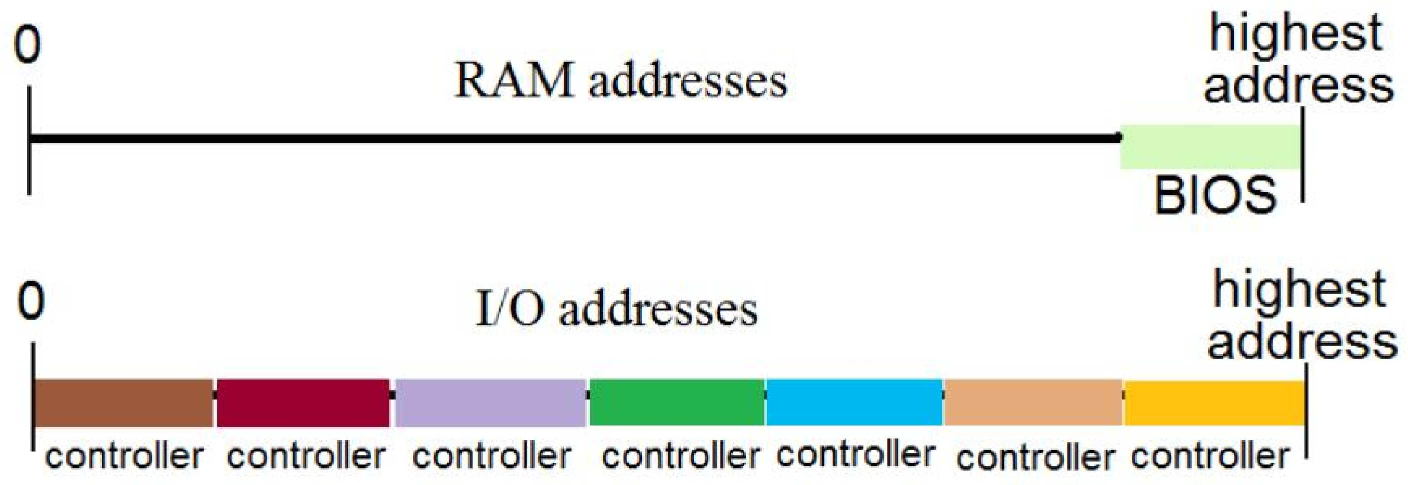


I/O Addressing

- **Memory-mapped I/O**



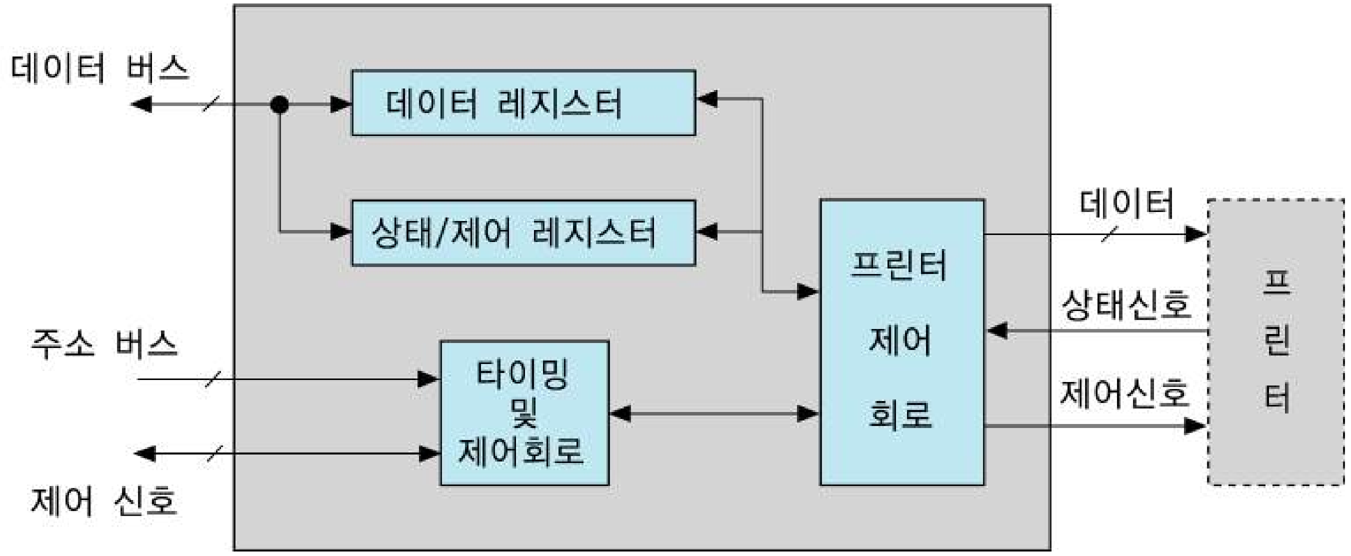
- **Port-mapped I/O**



Memory-mapped vs. Port-mapped

1. **Memory-mapped I/O**
    - Portions of the address space are assigned to I/O devices.
    - Reads and writes to those addresses are interpreted as commands to the I/O device
    - No I/O instructions
    - Motorola M680x0, ARM, MIPS, PowerPC ...
2. **Port-mapped I/O (= Isolated I/O = I/O-mapped I/O)**
    - Separate address spaces for memory and I/O
    - I/O instruction
        - Specifying both the device number and the command word
    - $M/\overline{IO}$ control signal
    - Protection
        - Making the I/O instructions illegal to execute when not in kernel mode
    - Intel IA-32, IBM 370 ...

1. Programmed I/O (=Polling-based Transfer)

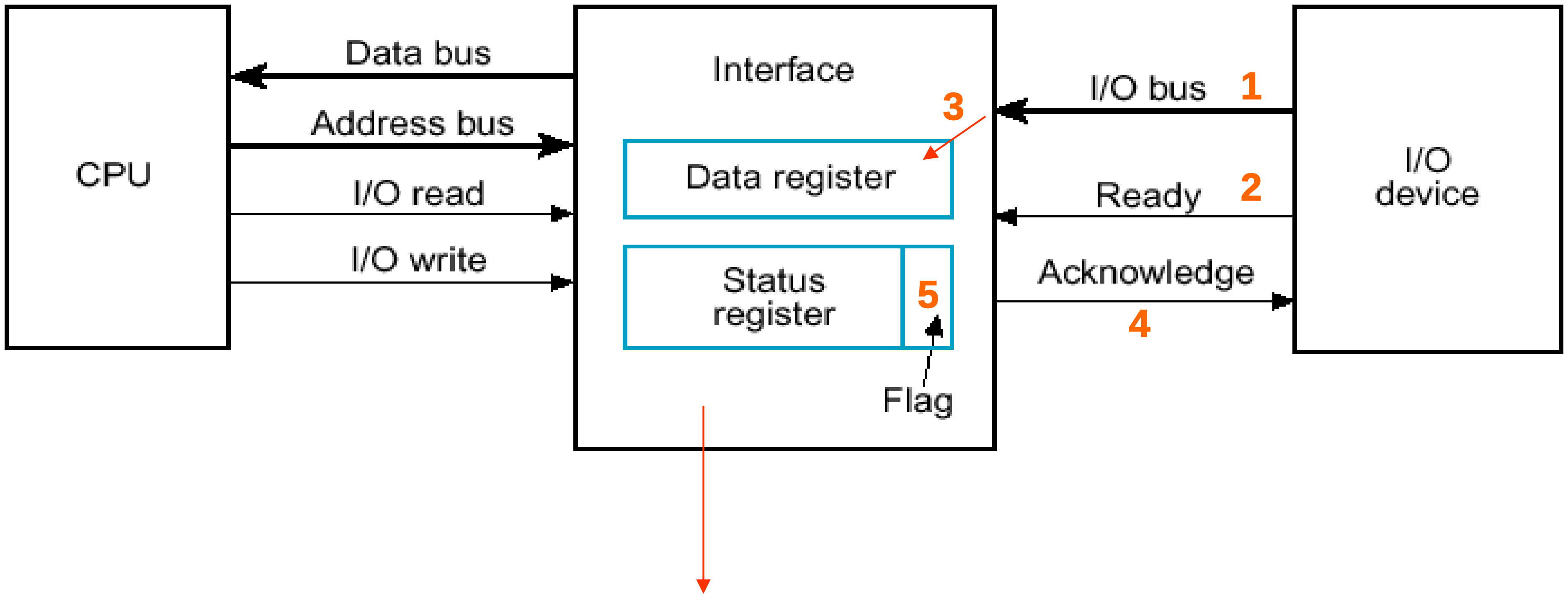


- **Status register**
  - Done bit and Error bit
- **Data register**
  - Data to be printed
- **Operations of processor**
  - Must wait until the done bit set by the printer

Programmed I/O

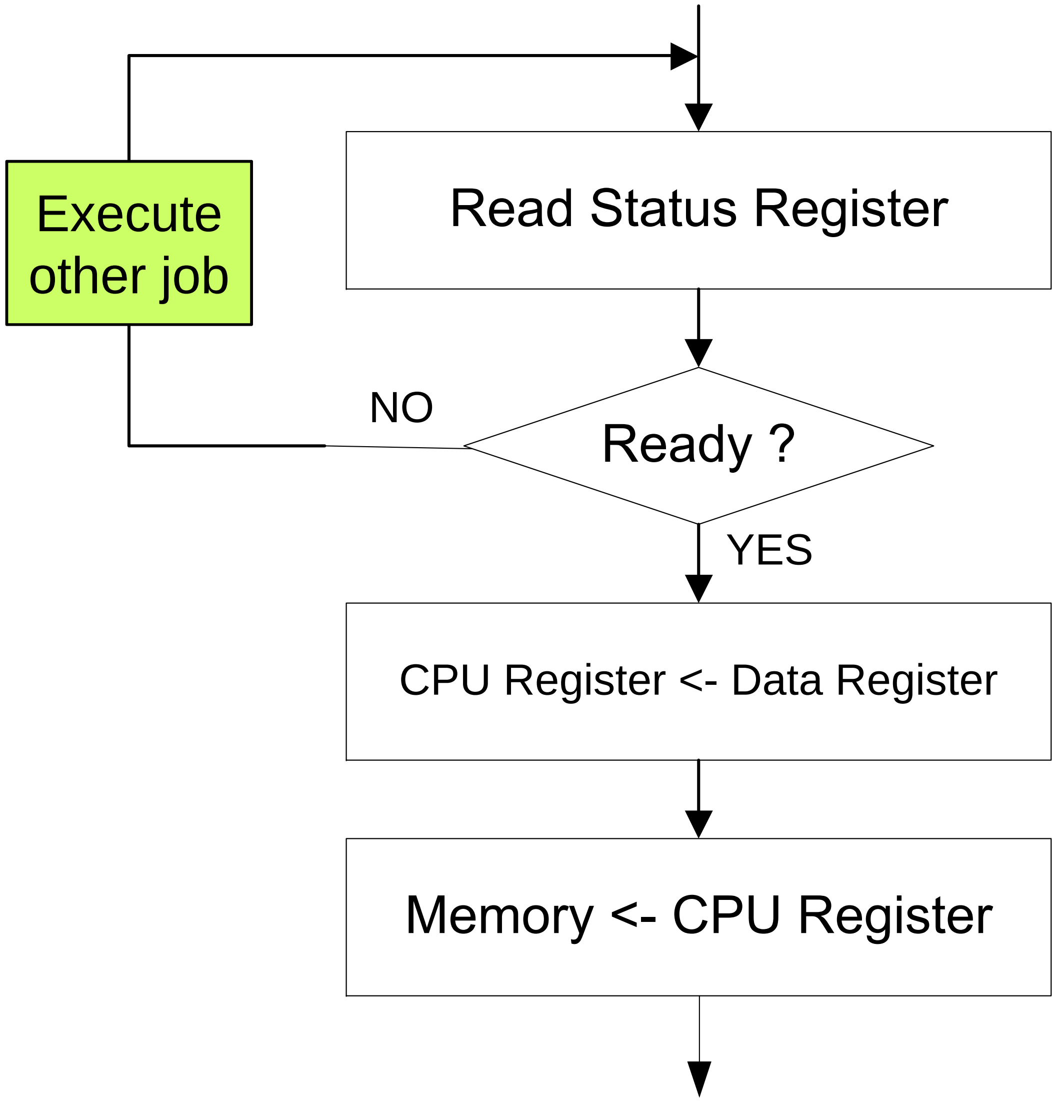
- **Polling**
  - The process of periodically checking status bits to see if it is time for the next I/O operation
- **Programmed I/O (= program-controlled transfer)**
  - I/O device simply puts the information in a Status register.
  - The simplest way for an I/O device to communicate with the processor
  - The processor is totally in control and does all the work.
  - Best with lower-bandwidth devices
- **Waste a lot of processor time**
  - Because the processors are much faster than I/O devices
  - Read the Status register many times, only to find that the device has not yet completed I/O operation.
  - idle loop, busy waiting, spinlock

Example Programmed Input

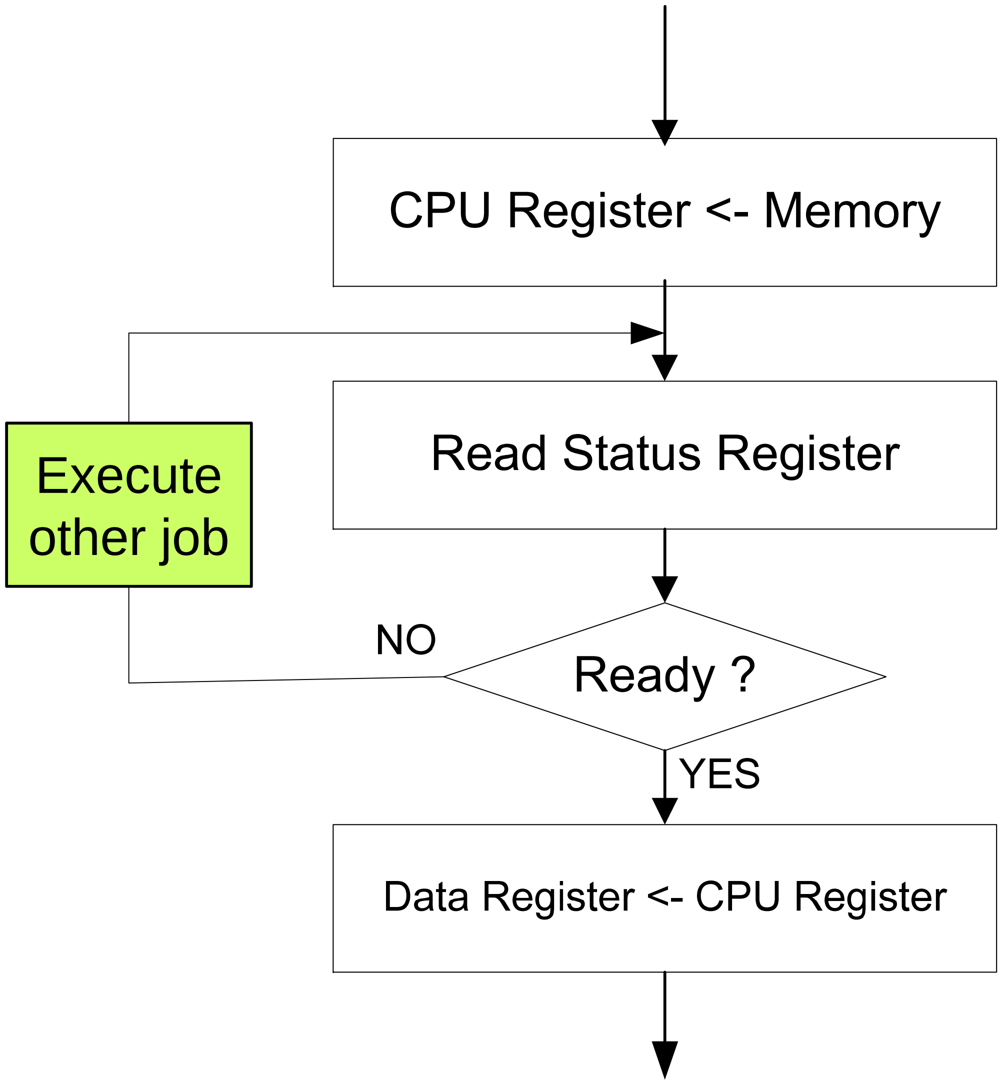


CPU must check the value of the flag - wasting time

Programmed Input



Programmed Output



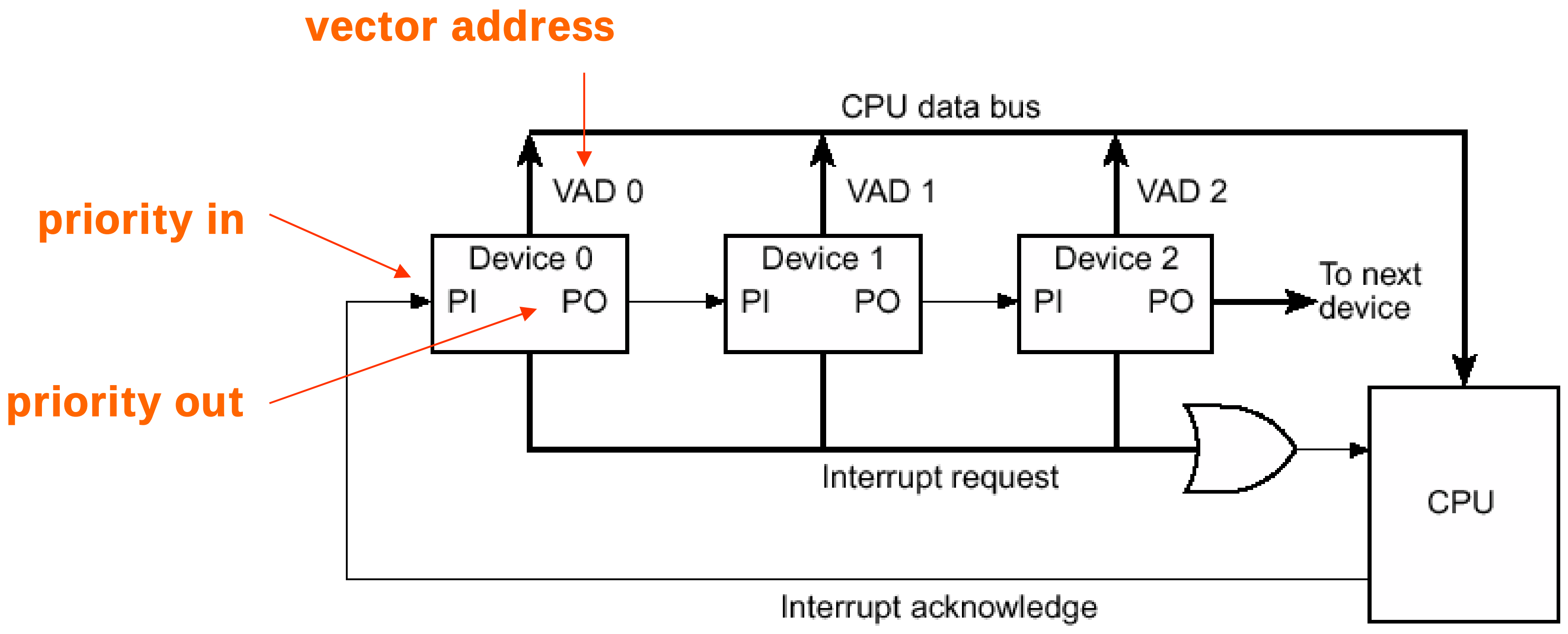
2. Interrupt Driven I/O

- **Common characteristics with programmed I/O**
  - OS still transfers data in small number of bytes.
  - Best with lower-bandwidth devices
  - More interested in reducing the cost
- **Difference**
  - I/O device informs the processor when ready
  - Relieving the processor from having to wait for every I/O
- **I/O operation**
  - OS simply works on other tasks while data is being read from or written to the device.
  - On interrupts, OS reads the status to check for errors.
  - If none, OS transfers data.
  - When I/O completed, OS can inform the program.

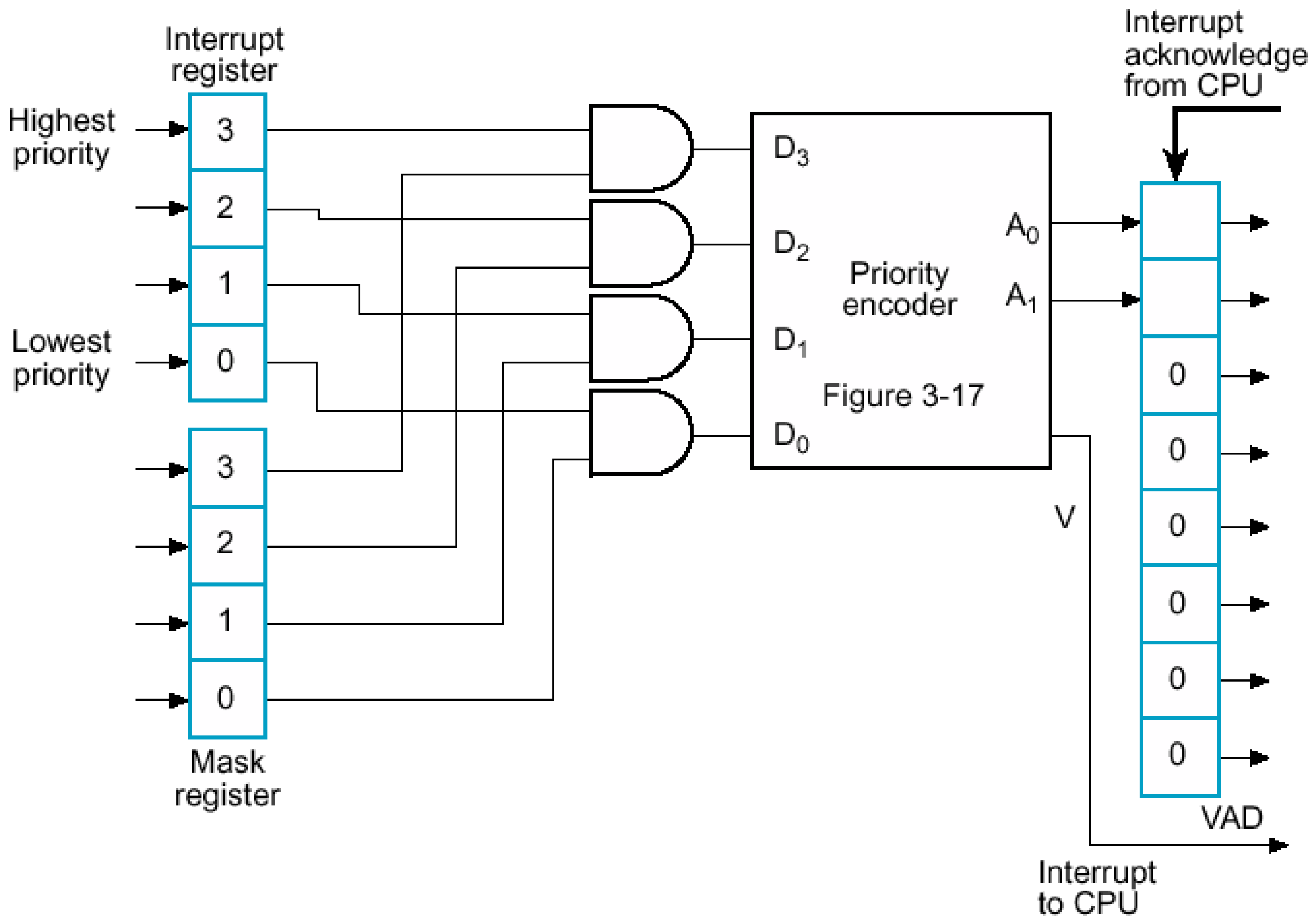
I/O Interrupts

- **I/O interrupts**
  - Alleviate overhead in polling interface
  - Using interrupts to notify the processor when an I/O device requires attention from the processor
  - Do not waste processor time
- **Interrupt identification**
  1. Vectored interrupt
     - The interrupting device sends interrupt vector address
     - Identified by hardware
  2. Non-vectored interrupt = Polled interrupt
     - Processor checks all potential interrupting devices one by one
     - Identified by software

Daisy-chain Priority Interrupt



Parallel Priority Interrupt

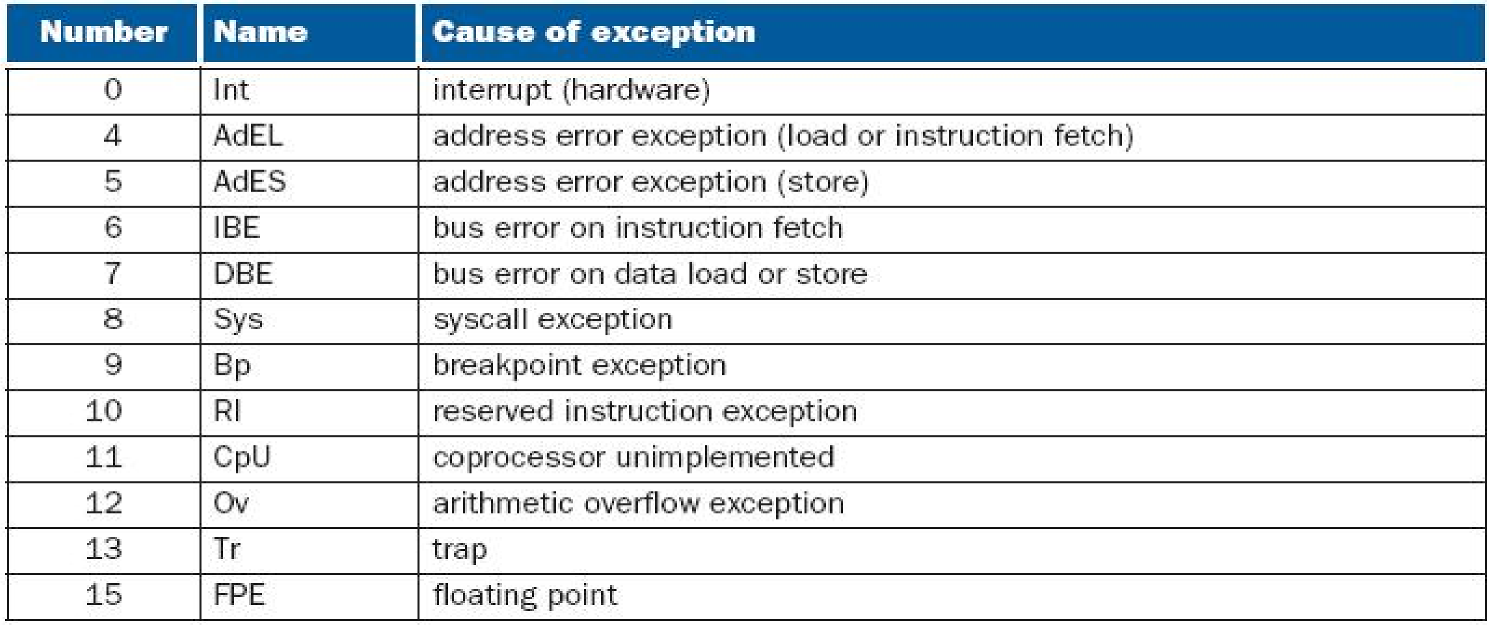


Interrupt Handling

1. Logically AND the pending interrupt field and the interrupt mask field.
2. Select the highest priority interrupt.
3. Save the interrupt mask.
4. Change the interrupt mask.
   - Disable all interrupts of equal or lower priority.
5. Save the processor state.
6. Set the interrupt enable bit.
   - Allow higher-priority interrupts.
7. Call the appropriate interrupt routine.
8. Reset the interrupt enable bit and restore mask.

Interrupt Handling in MIPS

- **Exception handler**
  - 8000 0180_{hex} in the kernel space
  - Examines the exception's cause and jumps to an appropriate point in the operating system
- **Cause of exception**



Number	Name	Cause of exception
0	Int	interrupt (hardware)
4	AdEL	address error exception (load or instruction fetch)
5	AdES	address error exception (store)
6	IBE	bus error on instruction fetch
7	DBE	bus error on data load or store
8	Sys	syscall exception
9	Bp	breakpoint exception
10	RI	reserved instruction exception
11	CpU	coprocessor unimplemented
12	Ov	arithmetic overflow exception
13	Tr	trap
15	FPE	floating point

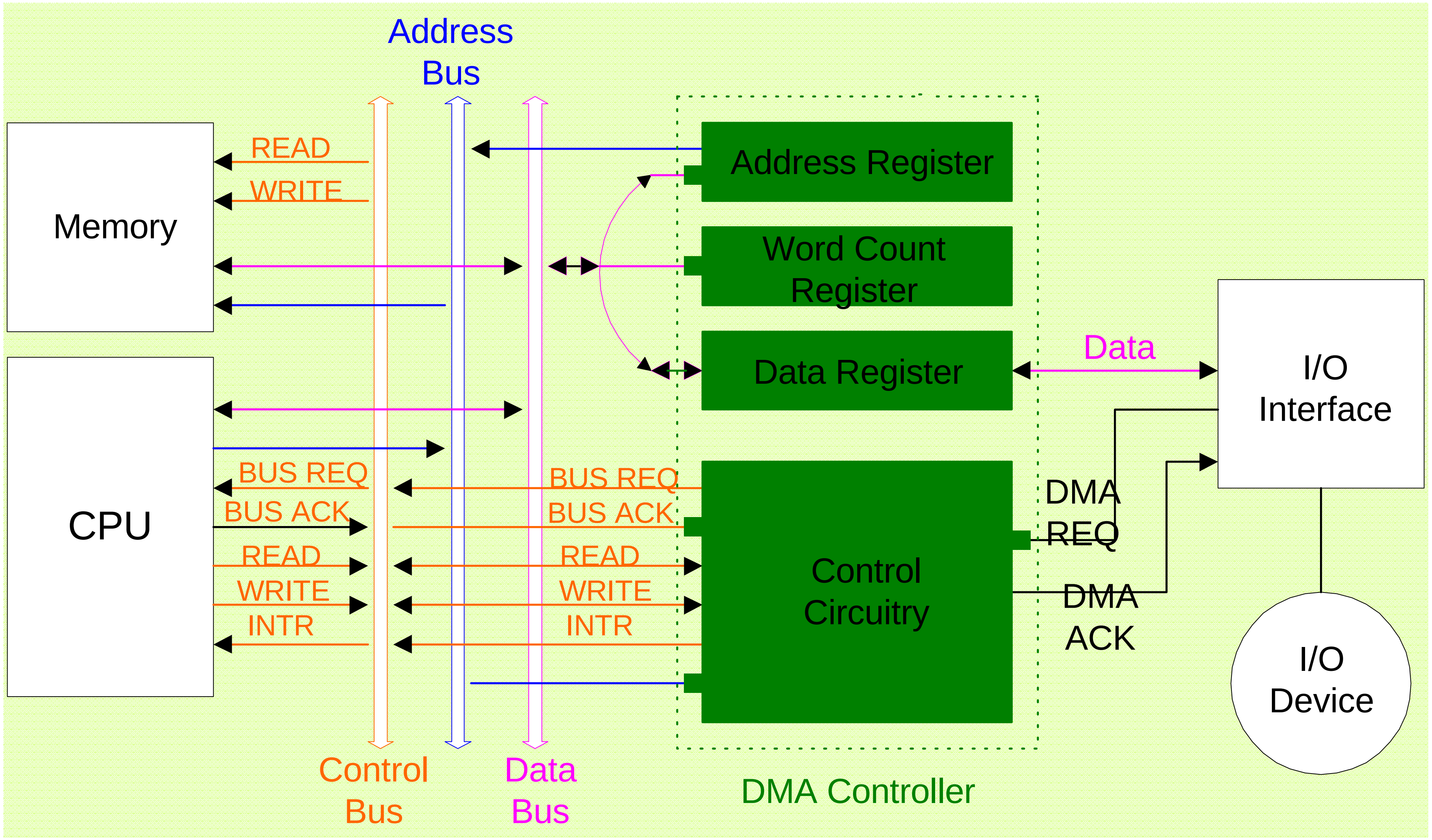
3. DMA (Direct Memory Access)

1. **Initial setup of the DMA**
   - Device ID, operation, memory address, number of bytes
2. **DMA operation (cf) cycle stealing**

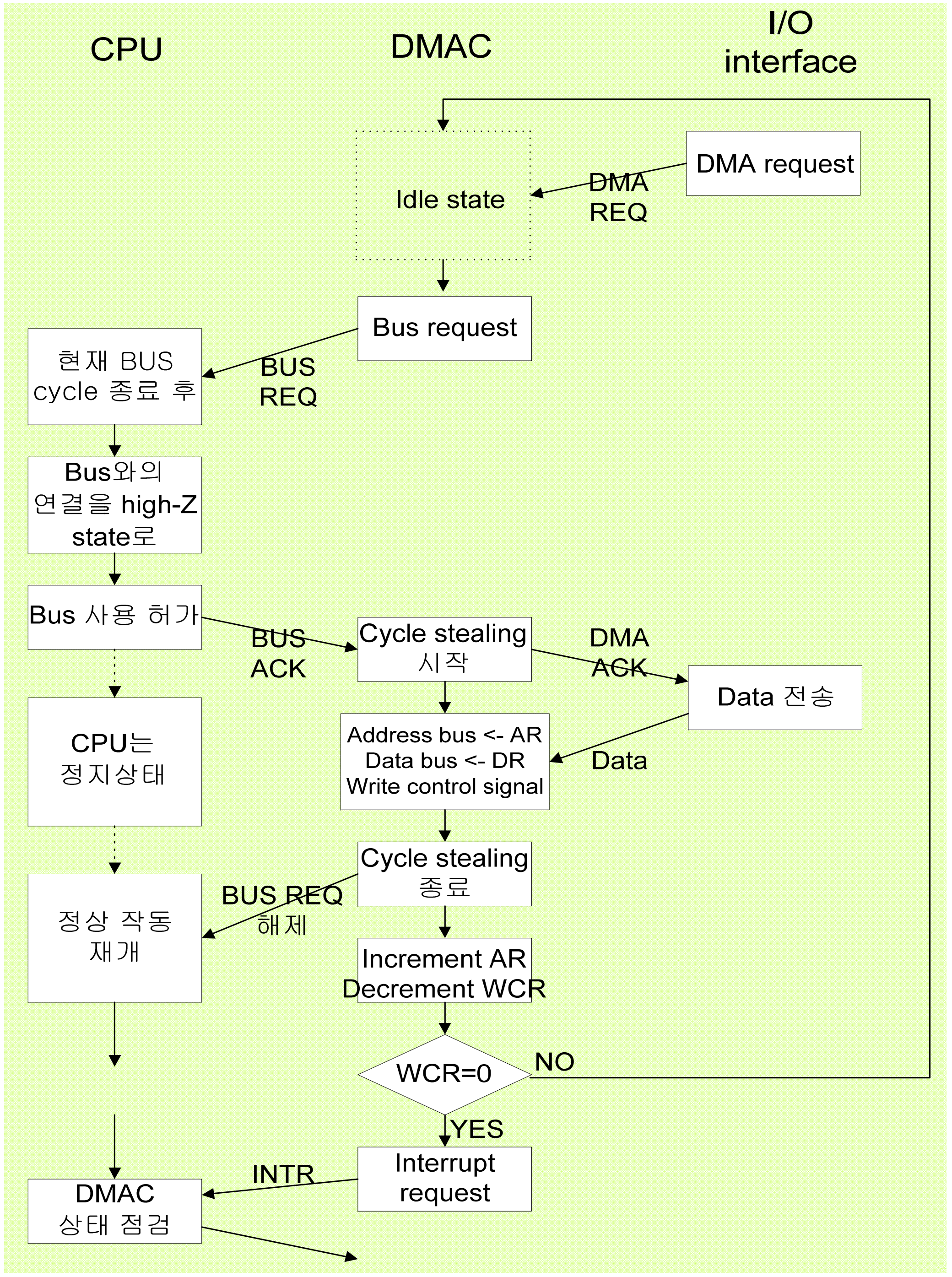
```
while (WCR>0) {acquire bus through arbitration;
               transfer data;
               WCR--; AR++;}
```

3. **Interrupt request**
   - When DMA transfer is complete
   - Processor checks whether the entire operation completed successfully.

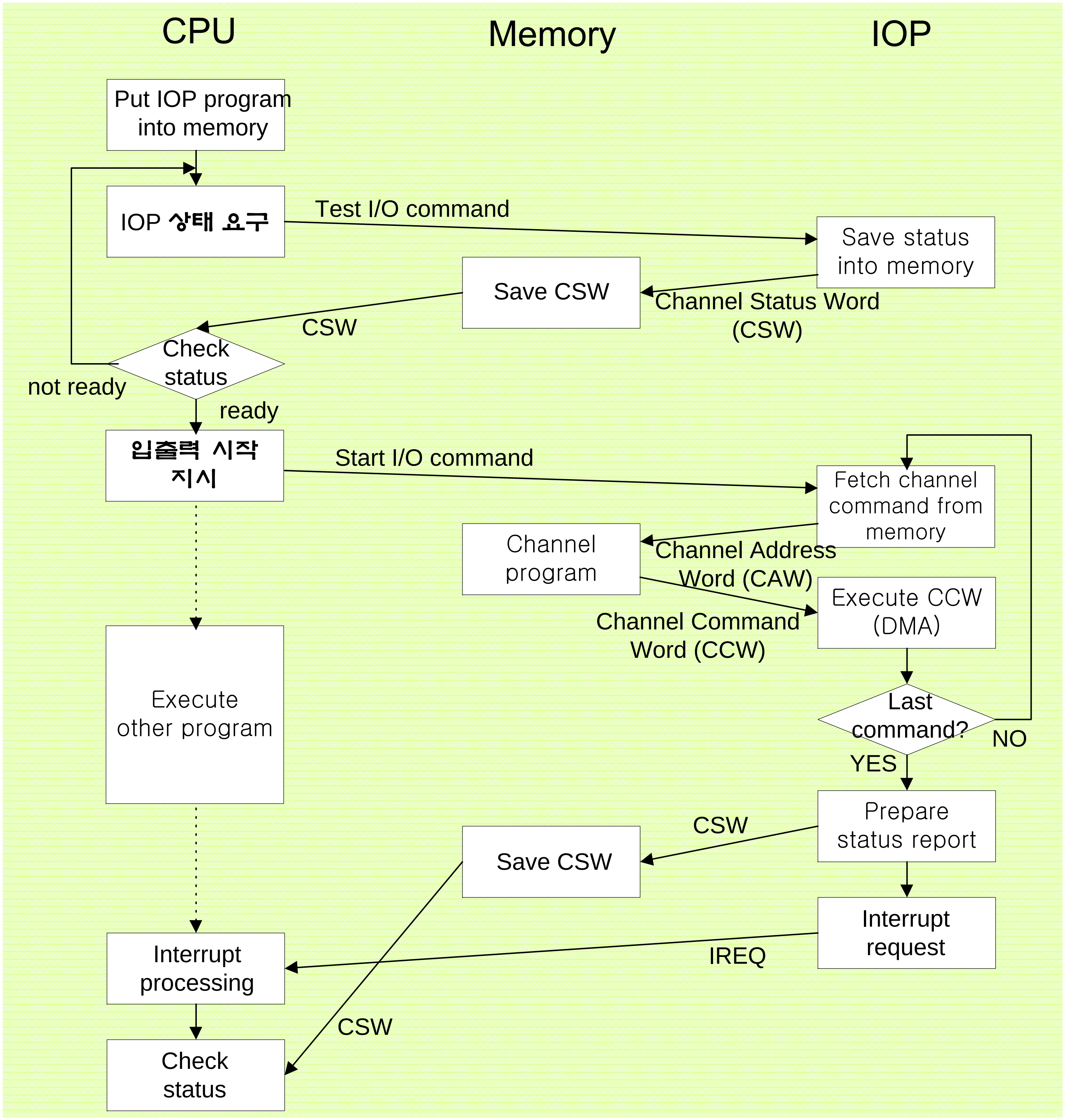
DMA Controller



DMA (Input)


CPU
DMAC
I/O interface
DMA request
DMA REQ
Idle state
Bus request
BUS REQ
현재 BUS cycle 종료 후
Bus와의 연결을 high-Z state로
Bus 사용 허가
BUS ACK
Cycle stealing 시작
DMA ACK
Data 전송
Data
Address bus <- AR
Data bus <- DR
Write control signal
CPU는 정지상태
Cycle stealing 종료
BUS REQ 해제
정상 작동 재개
Increment AR
Decrement WCR
WCR=0
NO
YES
Interrupt request
INTR
DMAC 상태 점검

IOP


CPU
Memory
IOP
Put IOP program into memory
IOP 상태 요구
Test I/O command
Save status into memory
Save CSW
Channel Status Word (CSW)
CSW
Check status
not ready
ready
입출력 시작 지시
Start I/O command
Fetch channel command from memory
Channel program
Channel Address Word (CAW)
Execute CCW (DMA)
Channel Command Word (CCW)
Last command?
NO
YES
Execute other program
Prepare status report
CSW
Save CSW
Interrupt request
IREQ
Interrupt processing
CSW
Check status

Comparison of I/O Transfer Methods

	Polling	Interrupt-driven	DMA
I/O-to-memory transfer	through CPU	through CPU	direct
Interrupt	no	every byte or word	every block
Overhead	busy waiting	context switches	bus cycles
Data transfer	by instruction execution	by instruction execution	cycle stealing
Unit of transfer	byte or word	byte or word	block