

2023年度(令和5年)版


Ver. 2023-11-21a

Course number: CSC.T363



# コンピュータアーキテクチャ Computer Architecture

## 12. ベクタ、SIMDにおけるデータレベル並列性 Data-Level Parallelism in Vector and SIMD



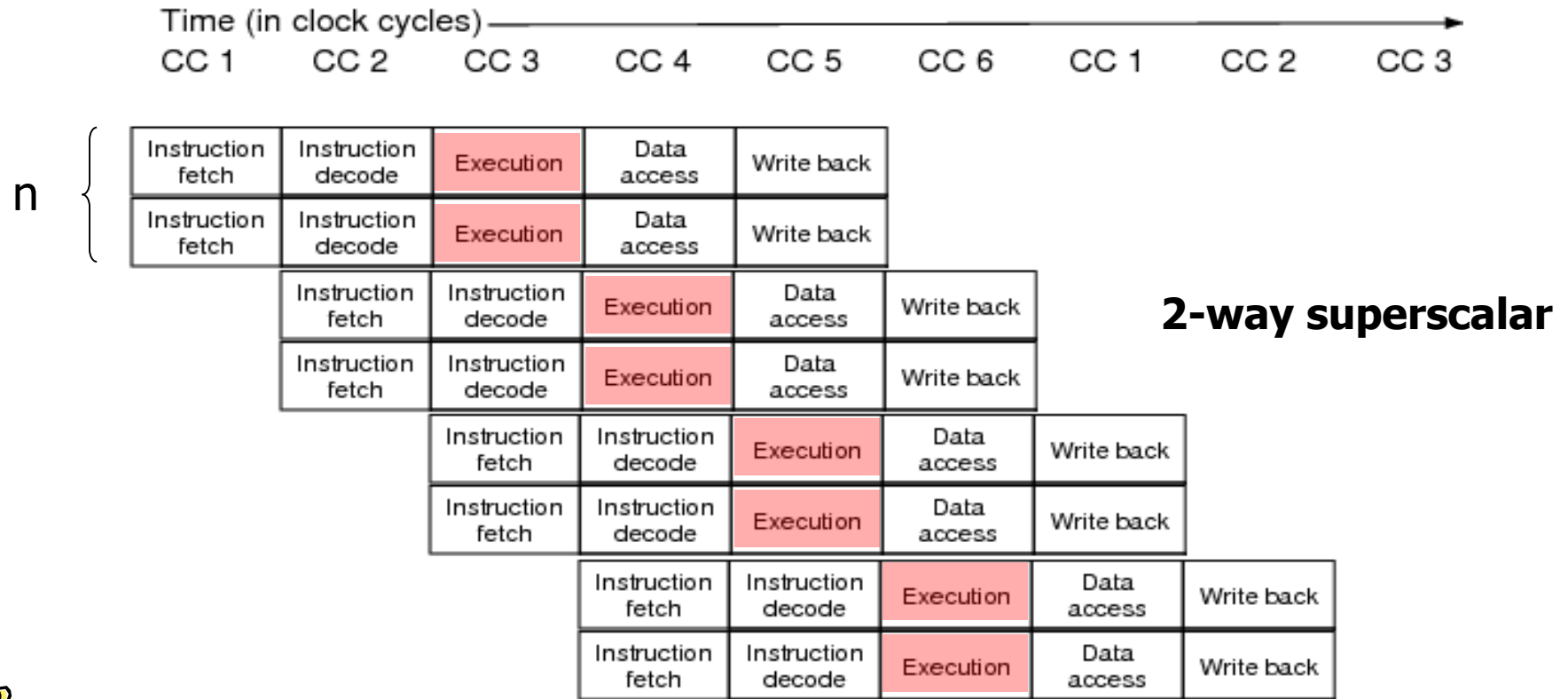
[www.arch.cs.titech.ac.jp/lecture/CA/](http://www.arch.cs.titech.ac.jp/lecture/CA/)  
Tue 13:30-15:10, 15:25-17:05  
Fri 13:30-15:10



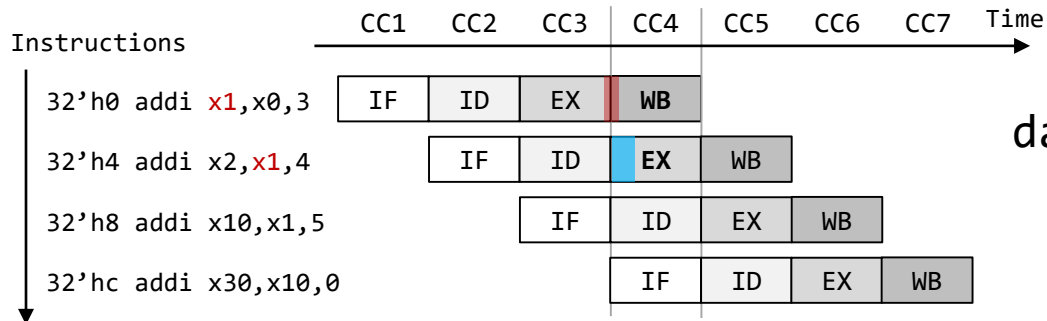
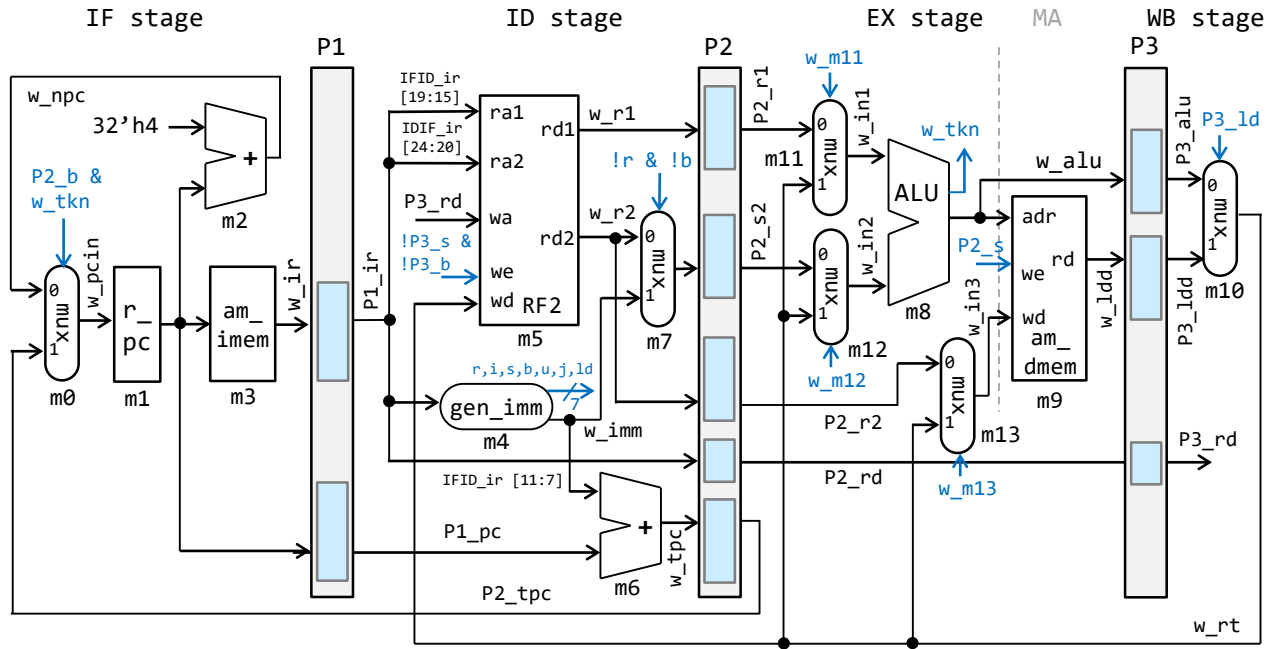
吉瀬 謙二 情報工学系  
Kenji Kise, Department of Computer Science  
kise\_at\_c.titech.ac.jp

# Superscalarと命令レベル並列性

- 複数のパイプラインを利用して IPC (instructions per cycle) を 1以上に引き上げる, 複数の命令を並列に実行
  - n-way スーパースカラ



# proc8: 4-stage pipelining processor



data forwarding

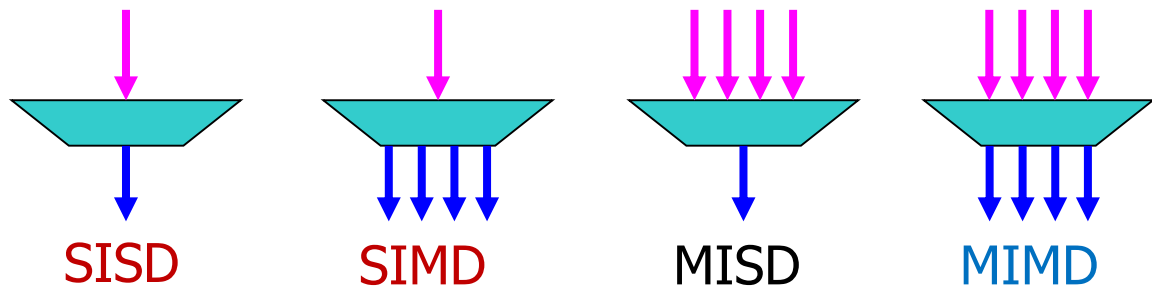


# アーキテクチャの異なる視点による分類

- Flynnによる命令とデータの流りに注目した並列計算機  
の分類(1966年)
  - **SISD** (Single Instruction stream, Single Data stream)
  - **SIMD** (Single Instruction stream, Multiple Data stream)
  - **MISD** (Multiple Instruction stream, Single Data stream)
  - **MIMD** (Multiple Instruction stream, Multiple Data stream)

Instruction stream

Data stream



# SIMD Variants and multicore



- Vector architectures
- SIMD extensions
- Graphics Processing Units (GPUs)
  
- SIMD variants exploit data-level parallelism
- **Instruction-level parallelism** in superscalar processors
  - window size
- **Thread-level parallelism** in multicore processors



# SIMD extensions

- Media applications operate on data types narrower than the native word size
  - Example: disconnect carry chains to “partition” adder
- Implementations:
  - Intel MMX (1996)
    - Eight 8-bit integer ops or four 16-bit integer ops
  - Streaming SIMD Extensions (SSE) (1999)
    - Eight 16-bit integer ops
    - Four 32-bit integer/fp ops or two 64-bit integer/fp ops
  - Advanced Vector Extensions (AVX 2010)
    - Four 64-bit integer/fp ops
    - 256 bit vectors -> 512 -> 1024
  - Operands must be consecutive and aligned memory locations



# Vector architecture

- Computers designed by Seymour Cray starting in the 1970s
- Basic idea:
  - Read sets of data elements into “vector registers”
  - Operate on those registers
  - Disperse the results back into memory



Cray Supercomputer



# DAXPY (double precision $a \times X + Y$ )

```
void daxpy(int n, double a, double x[], double y[])
{
    for (int i = 0; i < n; i++) {
        y[i] = a*x[i] + y[i];
    }
}
```





# DAXPY in MIPS Instructions



Example: DAXPY (double precision  $a \times X + Y$ )

```
      L.D      F0,a          ; load scalar a
      DADDIU   R4,Rx,#512    ; upper bound of what to load
Loop: L.D      F2,0(Rx      ) ; load X[i]
      MUL.D   F2,F2,F0      ; a x X[i]
      L.D     F4,0(Ry)      ; load Y[i]
      ADD.D   F4,F2,F2      ; a x X[i] + Y[i]
      S.D     F4,9(Ry)      ; store into Y[i]
      DADDIU   Rx,Rx,#8     ; increment index to X
      DADDIU   Ry,Ry,#8     ; increment index to Y
      SUBBU   R20,R4,Rx     ; compute bound
      BNEZ    R20,Loop      ; check if done
```

- Requires almost 600 MIPS operations



# DAXPY in VMIPS (MIPS with Vector) Instructions

- ADDV.D : add two vectors
- ADDVS.D : add vector to a scalar
- LV/SV : vector load and vector store from address

- Example: DAXPY (double precision  $a \cdot X + Y$ )

```
L.D      F0,a      ; load scalar a
LV       V1,Rx     ; load vector X
MULVS.D  V2,V1,F0  ; vector-scalar multiply
LV       V3,Ry     ; load vector Y
ADDV.D   V4,V2,V3  ; add
SV       Ry,V4     ; store the result
```

- Requires 6 instructions



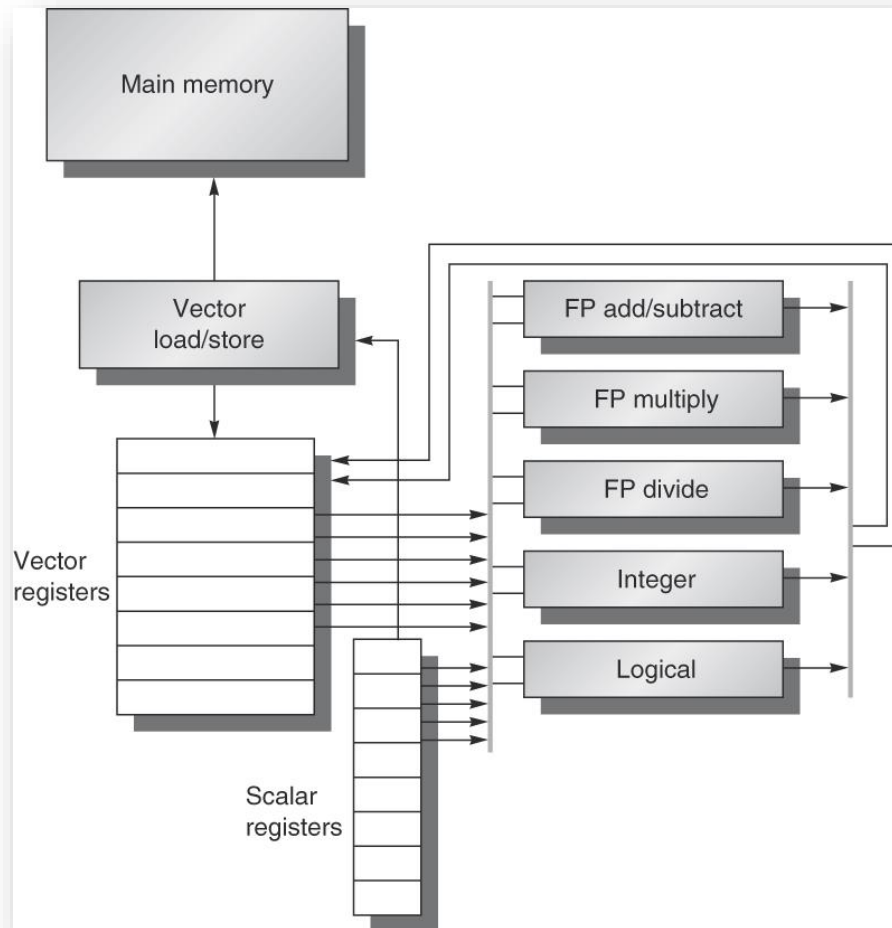
# Vector-vector add in RISC-V instructions

```
# vector-vector add routine of 32-bit integers
# void vvaddint32(size_t n, const int*x, const int*y, int*z)
# { for (size_t i=0; i<n; i++) { z[i]=x[i]+y[i]; } }
#
# a0 = n, a1 = x, a2 = y, a3 = z
# Non-vector instructions are indented
vvaddint32:
    vsetvli t0, a0, e32, ta, ma # Set vector length based on 32-bit vectors
    vle32.v v0, (a1)           # Get first vector
    sub a0, a0, t0             # Decrement number done
    slli t0, t0, 2             # Multiply number done by 4 bytes
    add a1, a1, t0             # Bump pointer
    vle32.v v1, (a2)           # Get second vector
    add a2, a2, t0             # Bump pointer
    vadd.vv v2, v0, v1         # Sum vectors
    vse32.v v2, (a3)           # Store result
    add a3, a3, t0             # Bump pointer
    bnez a0, vvaddint32       # Loop back
    ret                        # Finished
```



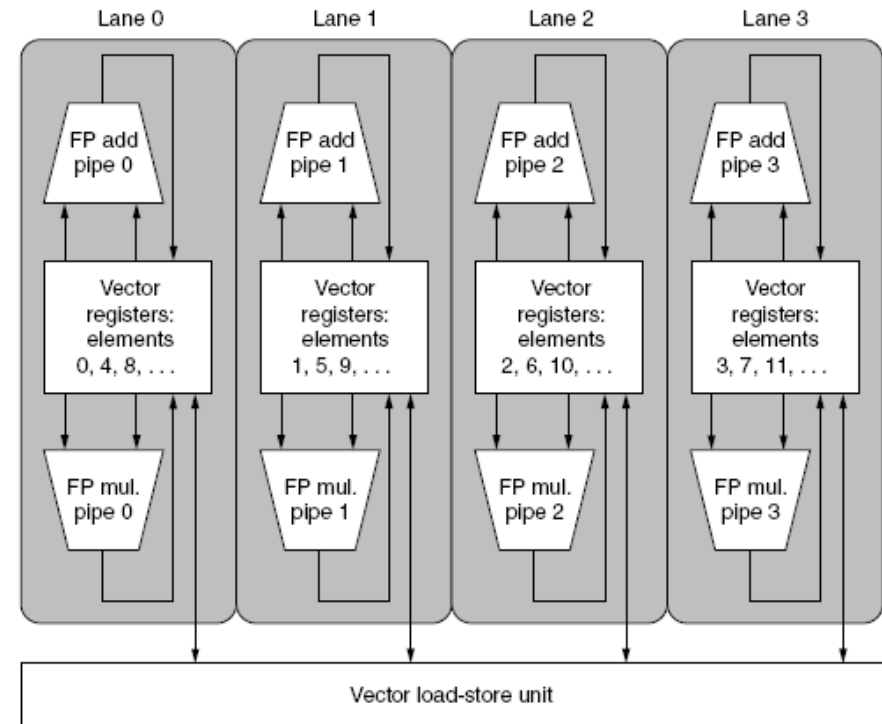
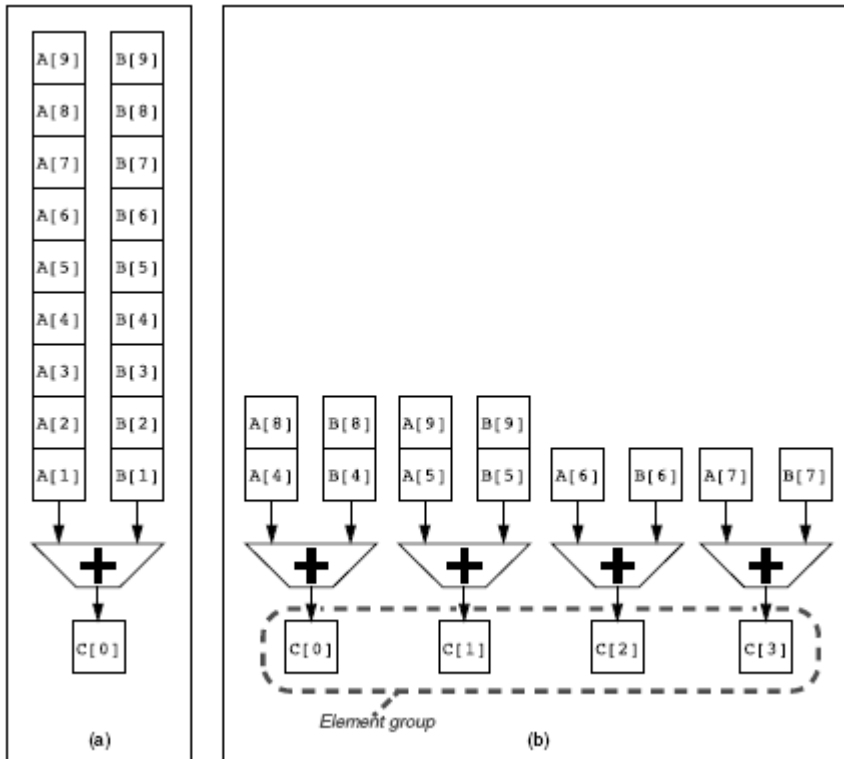
# The basic structure of a vector architecture, VMIPS

- Eight 64-element vector registers
- All the functional units are vector functional units.



# Multiple functional units to improve the performance

- (a) can complete one addition per cycle
- (b) can complete four addition per cycle
- The vector register storage is divided across the lanes



2023年度(令和5年)版


Ver. 2023-11-21a

Course number: CSC.T363



コンピュータアーキテクチャ  
Computer Architecture

入出力、バス  
Input/Output and Bus



[www.arch.cs.titech.ac.jp/lecture/CA/](http://www.arch.cs.titech.ac.jp/lecture/CA/)  
Tue 13:30-15:10, 15:25-17:05  
Fri 13:30-15:10



吉瀬 謙二 情報工学系  
Kenji Kise, Department of Computer Science  
kise\_at\_c.titech.ac.jp

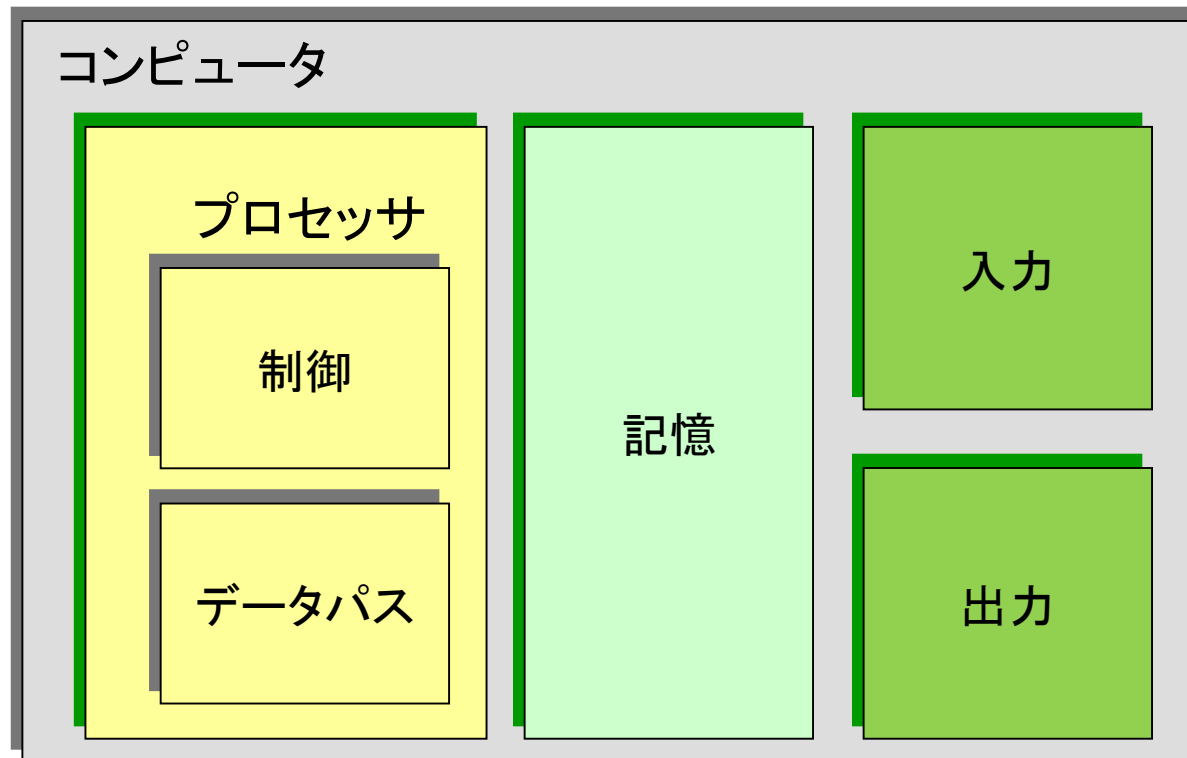
# コンピュータの古典的な要素

コンパイラ

Instruction Set Architecture (ISA), 命令セットアーキテクチャ

インタフェース

性能の評価



# Input and Output Devices





# Input and Output Devices

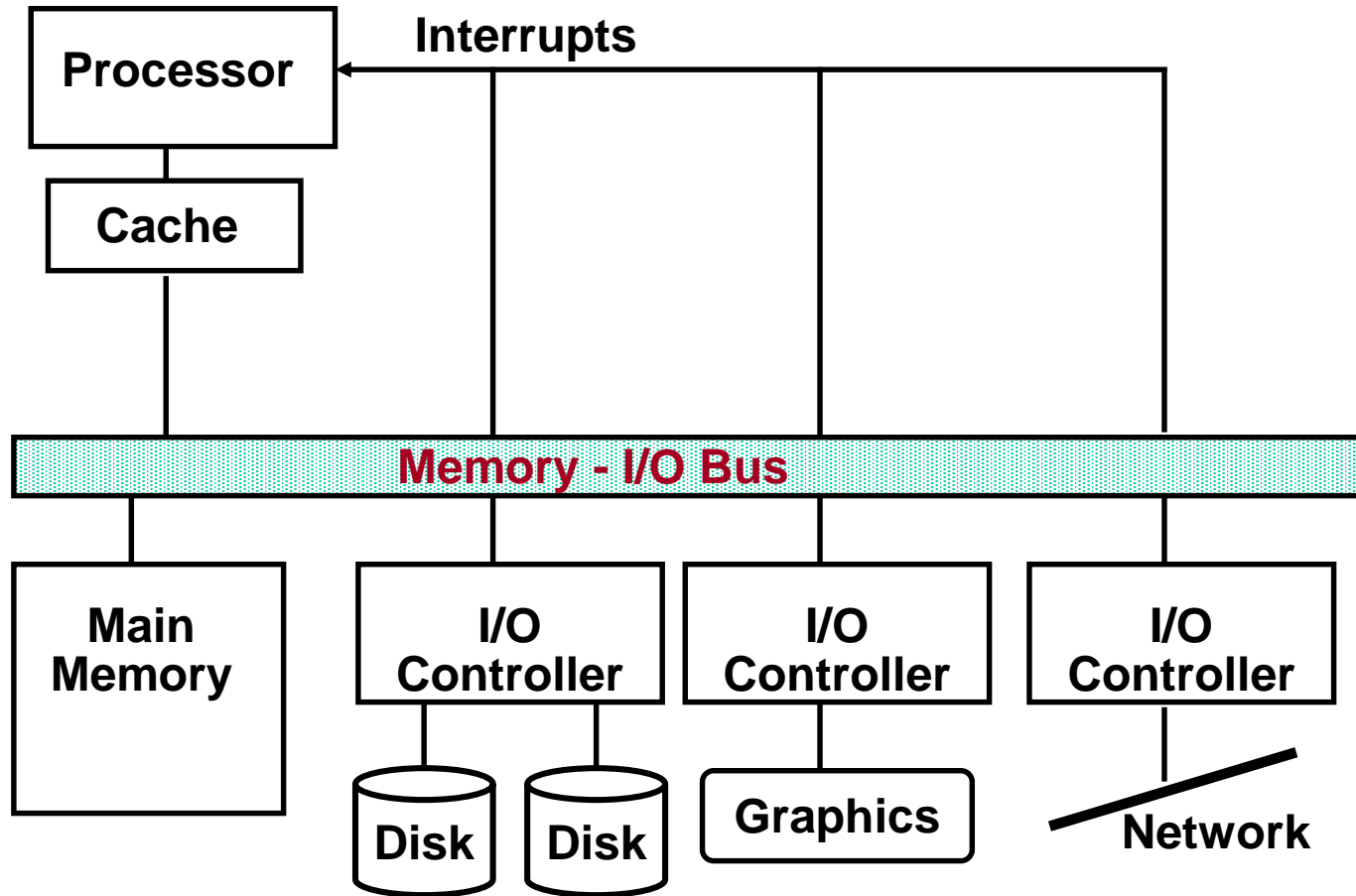
- I/O devices are **diverse** with respect to
  - Behavior – input, output or storage
  - Partner – human or machine
  - Data rate – the peak rate at which data can be transferred between the I/O device and the main memory or CPU

Device	Behavior	Partner	Data rate (Mb/s)
Keyboard	input	human	0.0001
Mouse	input	human	0.0038
Laser printer	output	human	3.2000
Graphics display	output	human	800.0000-8000.0000
Network/LAN	input or output	machine	100.0000-1000.0000
Magnetic disk	storage	machine	240.0000-2560.0000

8 orders of magnitude range



# A Typical I/O System



# Bus, I/O System Interconnect

- A **bus** is a shared communication link

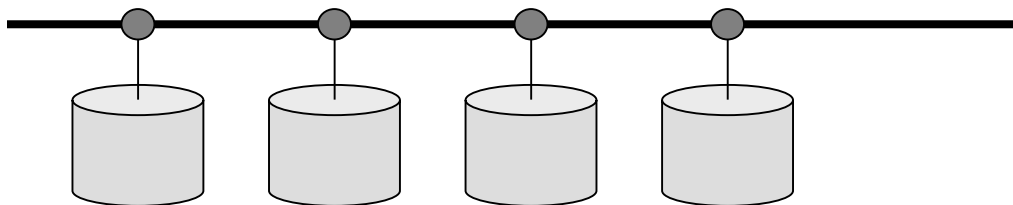
1bit data wire



1bit control wire



Bus

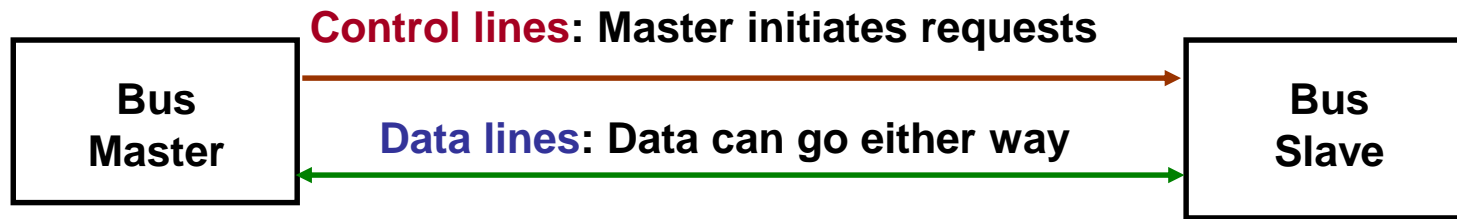


# Bus, I/O System Interconnect

- A **bus** is a **shared communication link** (a single set of wires used to connect multiple subsystems)
  - Advantages
    - Low cost – a single set of wires is shared in multiple ways
    - Versatile (多目的) – new devices can be added easily and can be moved between computer systems that use the same **bus standard**
  - Disadvantages
    - Creates a communication bottleneck – bus bandwidth limits the maximum I/O throughput
- The maximum bus speed is largely limited by
  - The length of the bus
  - The number of devices on the bus



# Bus Characteristics



- **Control lines**

- Signal requests and acknowledgments
- Indicate what type of information is on the data lines

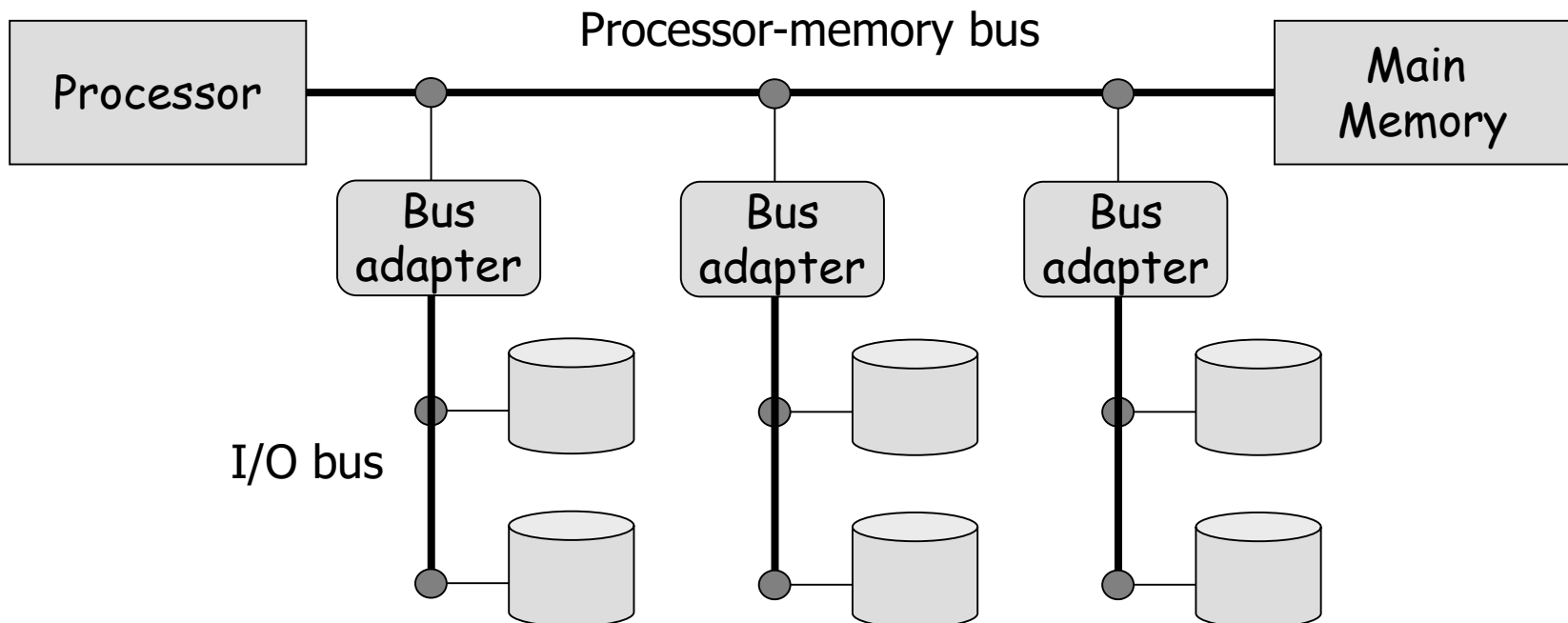
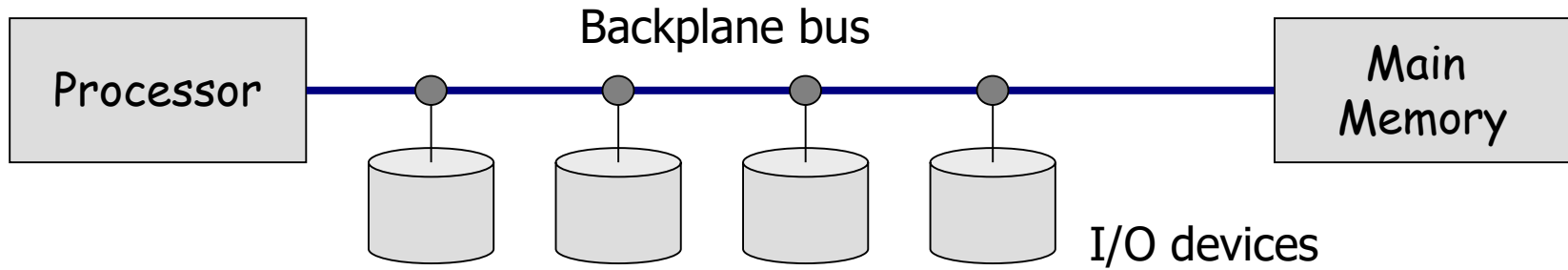
- **Data lines**

- Data, addresses, and complex commands

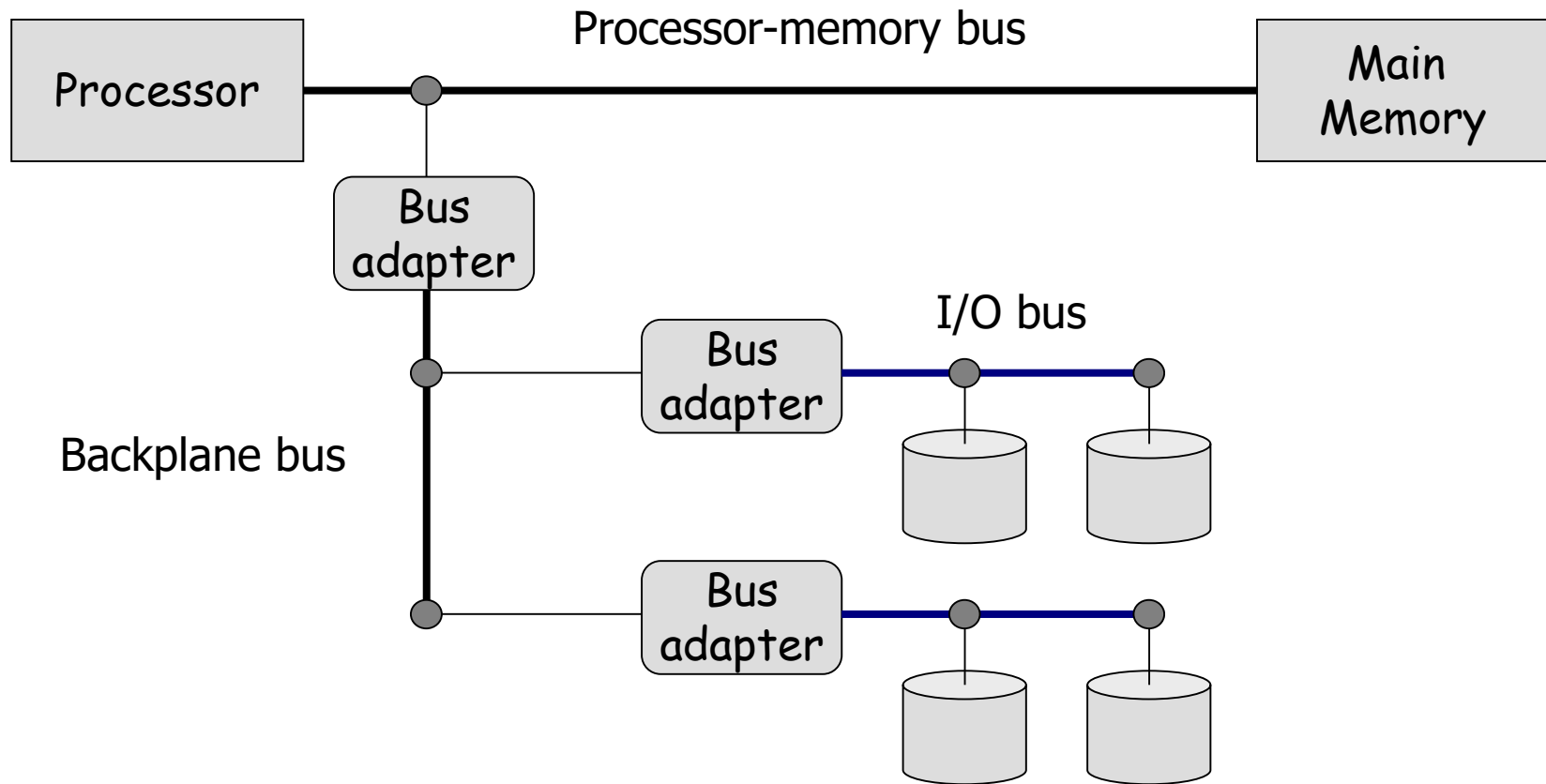
- Bus transaction consists of

- Master issuing the command (and address) – request
- Slave receiving (or sending) the data – action
- Defined by what the transaction does to memory
  - **Input** – inputs data from the I/O device to the memory
  - **Output** – outputs data from the memory to the I/O device

# Types of Buses (1)



# Types of Buses (2)



# Types of Buses (3)

- **Processor-memory bus**
  - Short and high speed
  - Matched to the memory system to maximize the memory-processor bandwidth
  - Optimized for cache block transfers
- **I/O bus** (industry standard, e.g., SCSI, USB, Firewire)
  - Usually is lengthy and slower
  - Needs to accommodate a wide range of I/O devices
  - Connects to the processor-memory bus or backplane bus
- **Backplane bus** (industry standard, e.g., ATA, PCI Express)
  - The backplane is an interconnection structure within the chassis
  - Used as an intermediary bus connecting I/O busses to the processor-memory bus



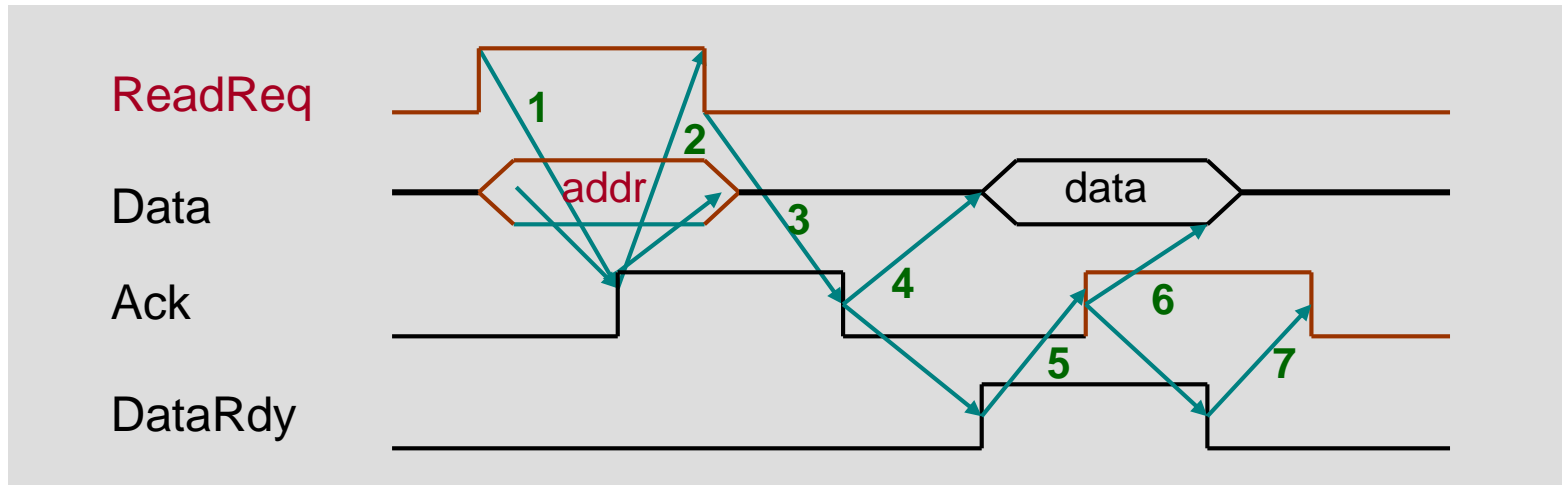


# Synchronous (同期式), Asynchronous (非同期式) Buses

- Synchronous bus (e.g., processor-memory buses)
  - Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
  - Advantage: involves very little logic and can run very fast
  - Disadvantages:
    - Every device communicating on the bus must use same clock rate
    - To avoid **clock skew**, they cannot be long if they are fast
- Asynchronous bus (e.g., I/O buses)
  - It is not clocked, so requires a **handshaking protocol** and additional control lines (ReadReq, Ack, DataRdy)
  - Advantages:
    - Can accommodate a wide range of devices and device speeds
    - Can be lengthened without worrying about clock skew or synchronization problems
  - Disadvantage: slow

# Asynchronous Bus Handshaking Protocol

An I/O device reads data from memory.



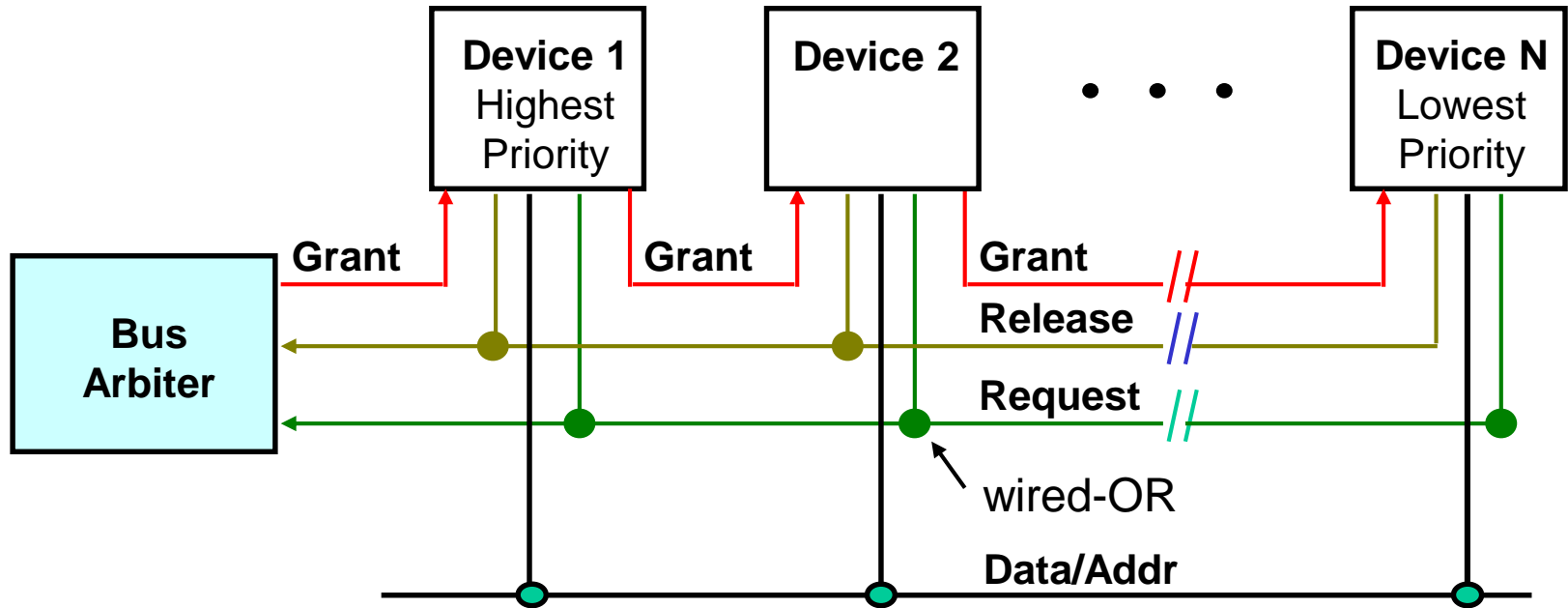
1. Memory sees **ReadReq**, reads **addr** from data lines, and raises **Ack**
2. I/O device sees **Ack** and releases the **ReadReq** and data lines
3. Memory sees **ReadReq** go low and drops **Ack**
4. When memory has data ready, it places it on data lines and raises **DataRdy**
5. I/O device sees **DataRdy**, reads the data from data lines, and raises **Ack**
6. Memory sees **Ack**, releases the data lines, and drops **DataRdy**
7. I/O device sees **DataRdy** go low and drops **Ack**

# The Need for Bus Arbitration (調停)

- Multiple devices may need to use the bus at the same time
- Bus arbitration schemes usually try to balance:
  - Bus priority - the highest priority device should be serviced first
  - Fairness - even the lowest priority device should never be completely locked out from the bus
- Bus arbitration schemes can be divided into four classes
  - Daisy chain arbitration
  - Centralized, parallel arbitration
  - Distributed arbitration by collision detection
    - device uses the bus when its not busy and if a collision happens (because some other device also decides to use the bus) then the device tries again later (Ethernet)
  - Distributed arbitration by self-selection



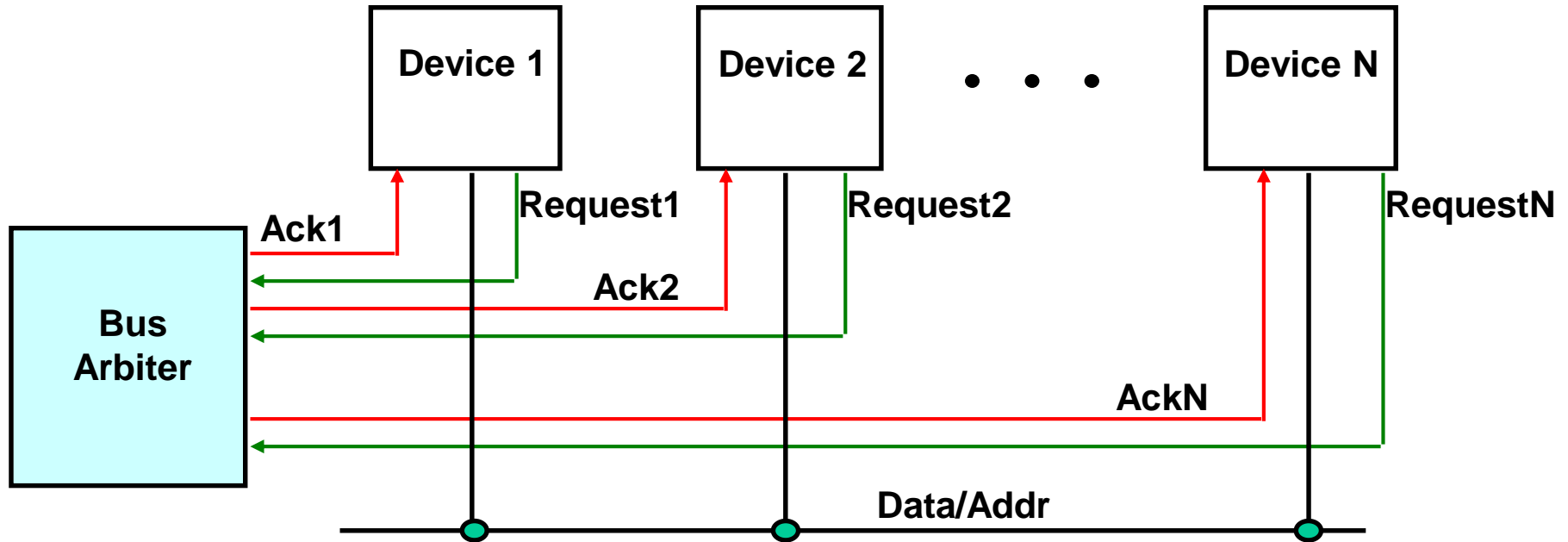
# Daisy Chain Bus Arbitration (デイジーチェーン)



- Advantage: simple
- Disadvantages:
  - Cannot assure fairness – a low-priority device may be locked out
  - Slower – the daisy chain grant signal limits the bus speed

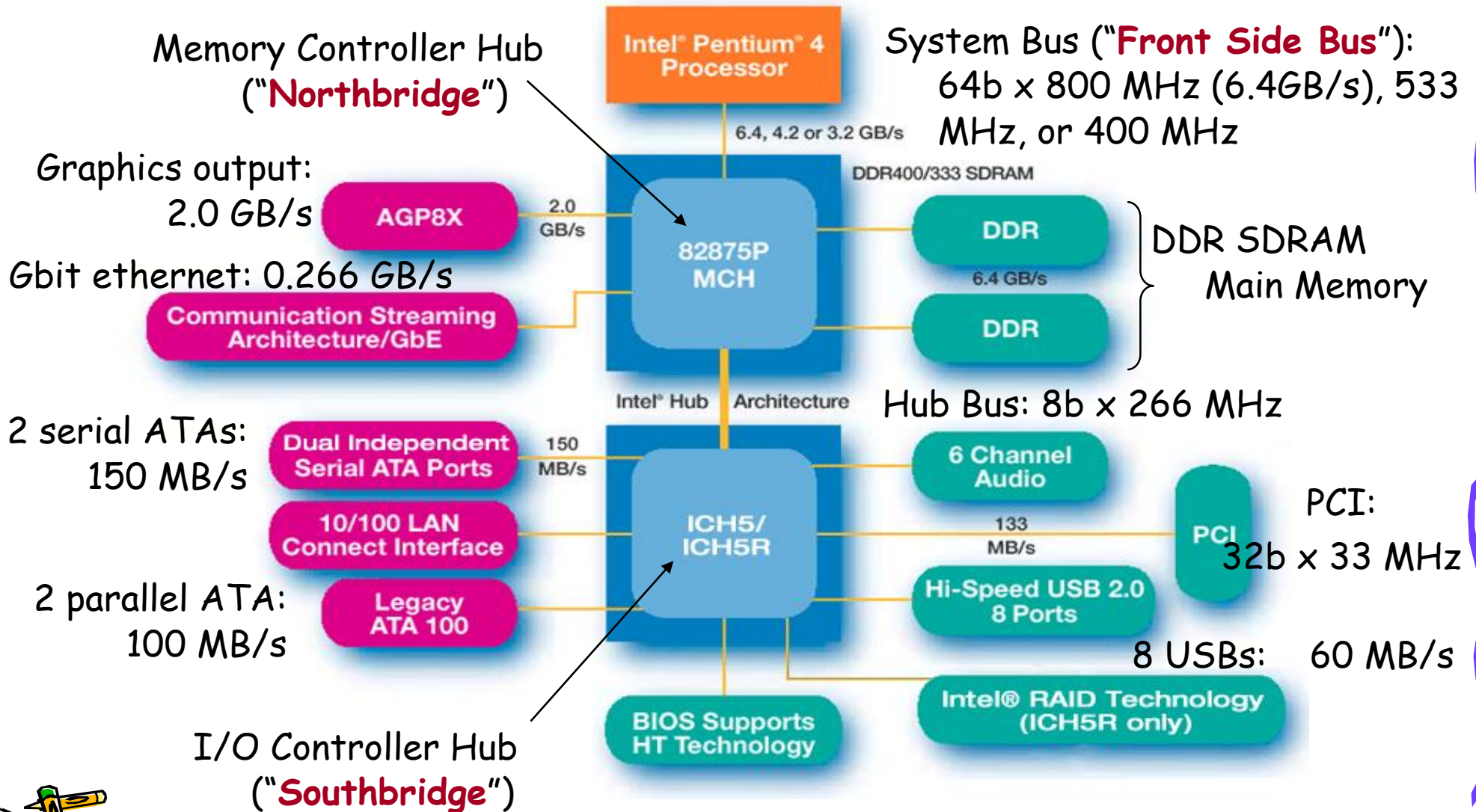


# Centralized Parallel Arbitration (集中並列方式)

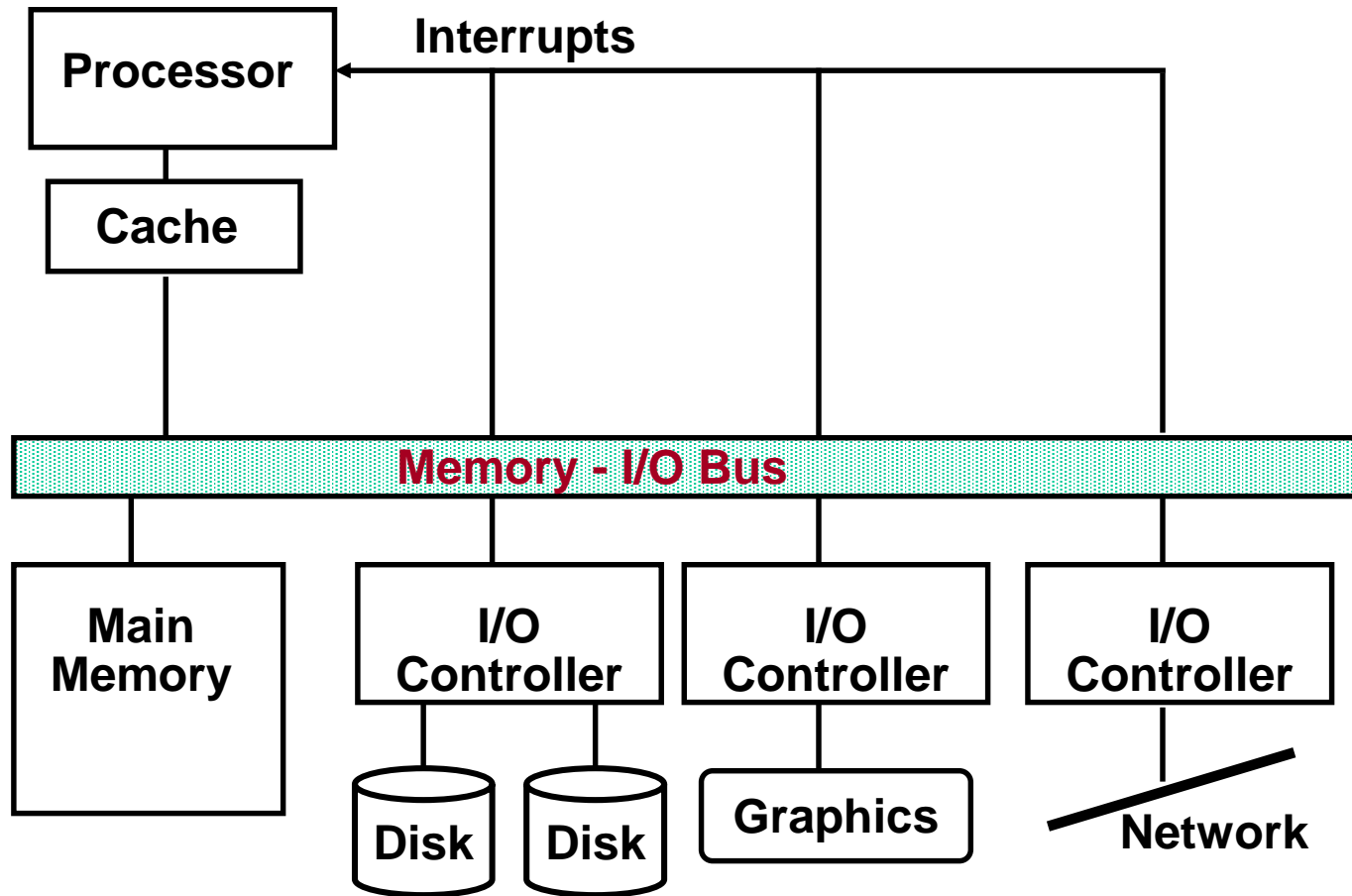


- Advantages: flexible, can assure fairness
- Disadvantages: more complicated arbiter hardware
- Used in essentially all processor-memory buses and in high-speed I/O buses

# Example: The Pentium 4's Buses



# A Typical I/O System and interrupts



# Communication of I/O Devices and Processor (1)

- How the processor directs the I/O devices
  - **Memory-mapped I/O**
    - Portions of the high-order memory address space are assigned to each I/O device
    - Read and writes to those memory addresses are interpreted as commands to the I/O devices
    - Load/stores to the I/O address space can only be done by the OS
  - **Special I/O instructions**



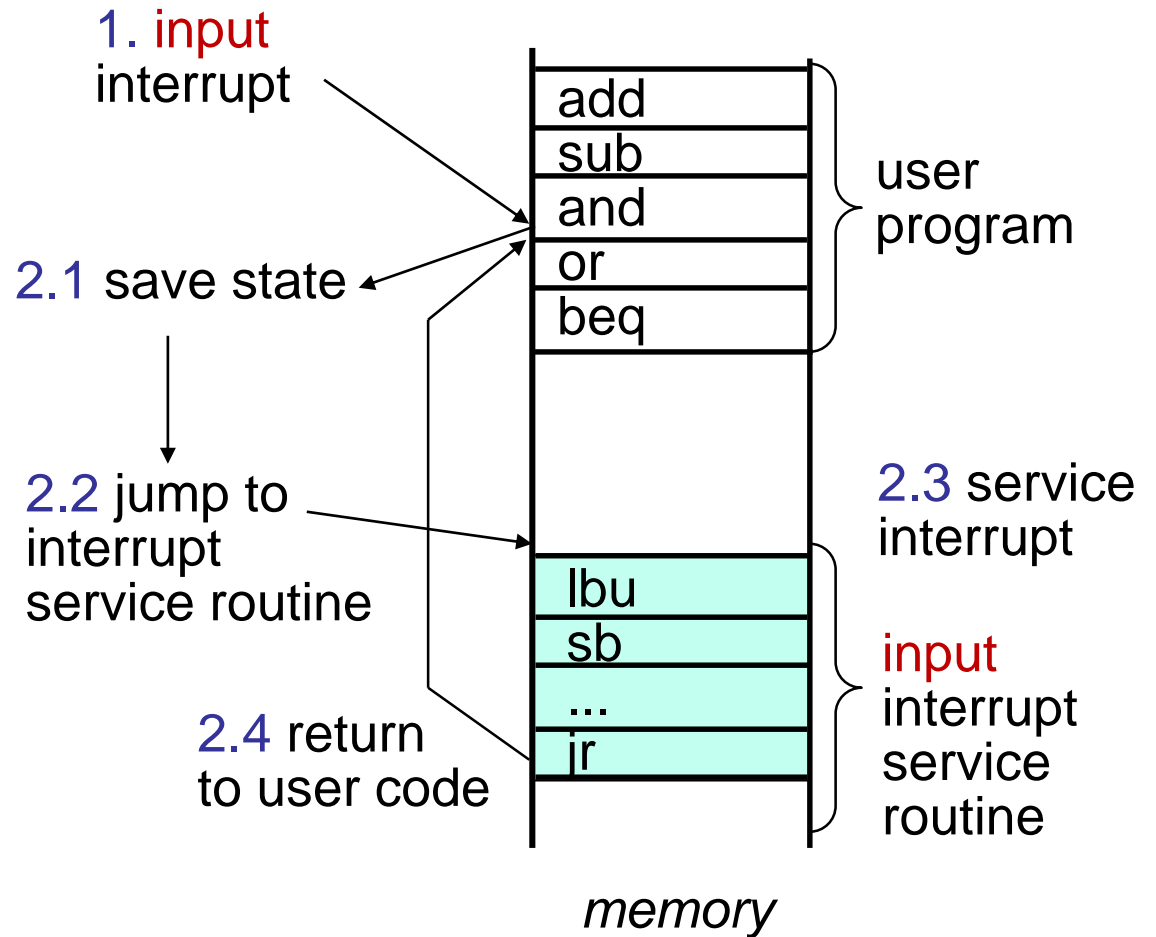
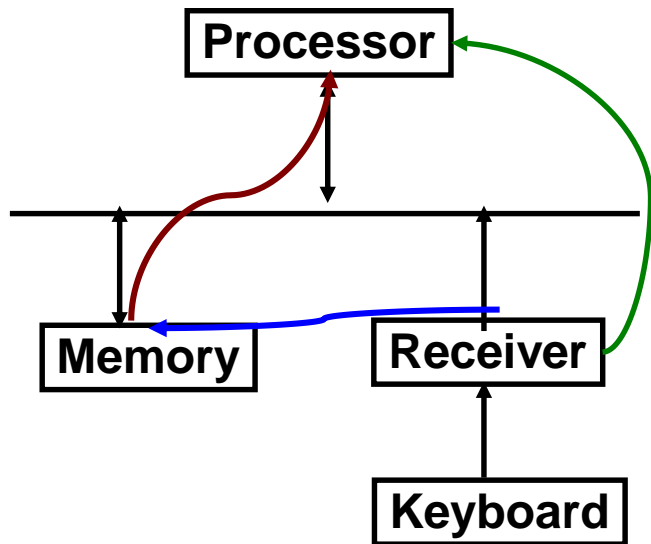


# Communication of I/O Devices and Processor (2)

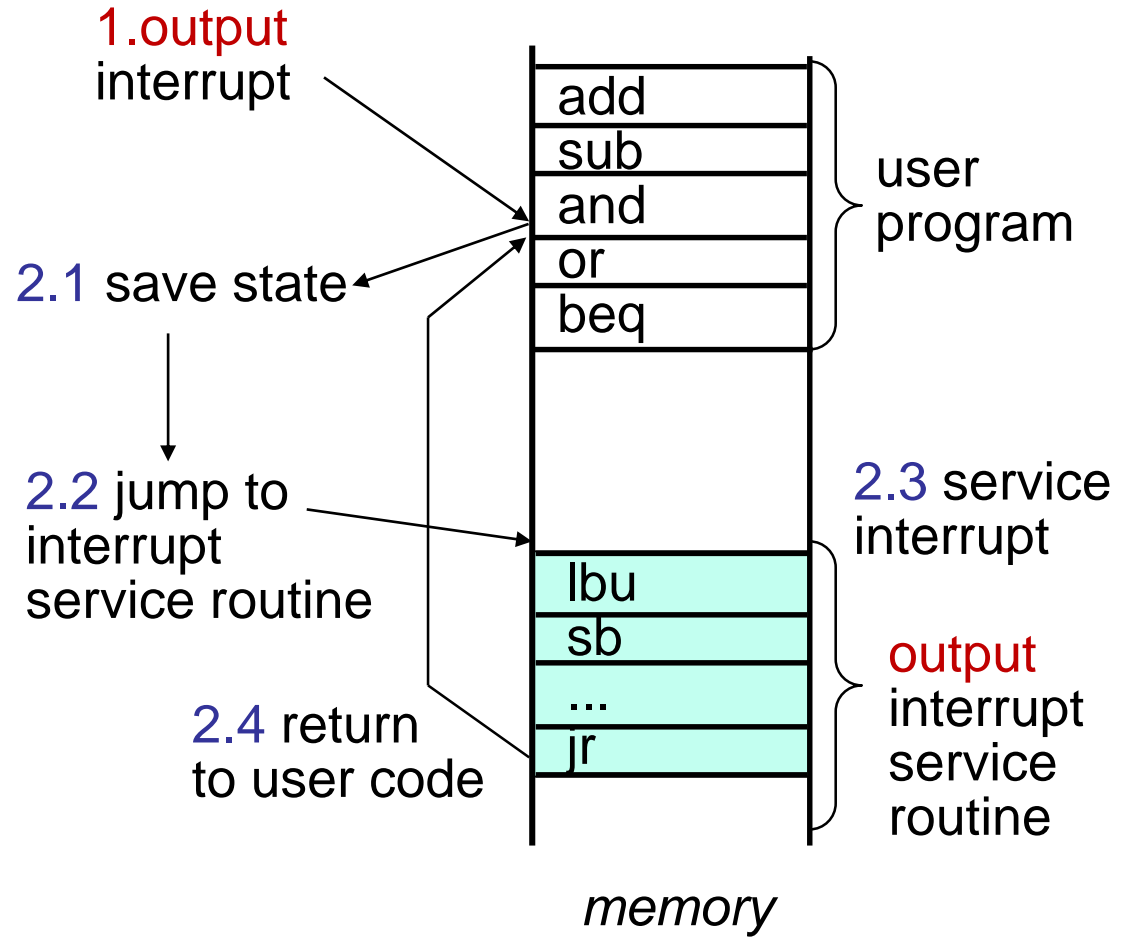
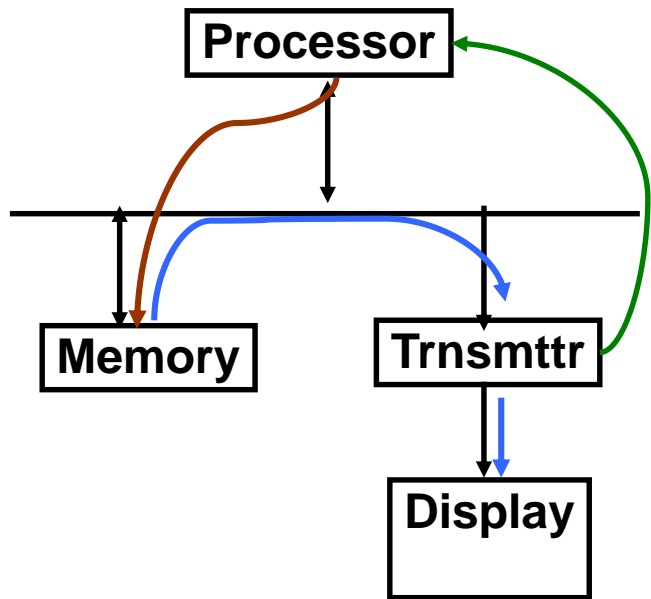
- How the I/O device communicates with the processor
  - **Polling** – the processor periodically checks the status of an I/O device to determine its need for service
    - Processor is totally in control – but does all the work
    - Can waste a lot of processor time due to speed differences
  - **Interrupt-driven I/O** – the I/O device issues an interrupts to the processor to indicate that it needs attention



# Interrupt-Driven Input



# Interrupt-Driven Output



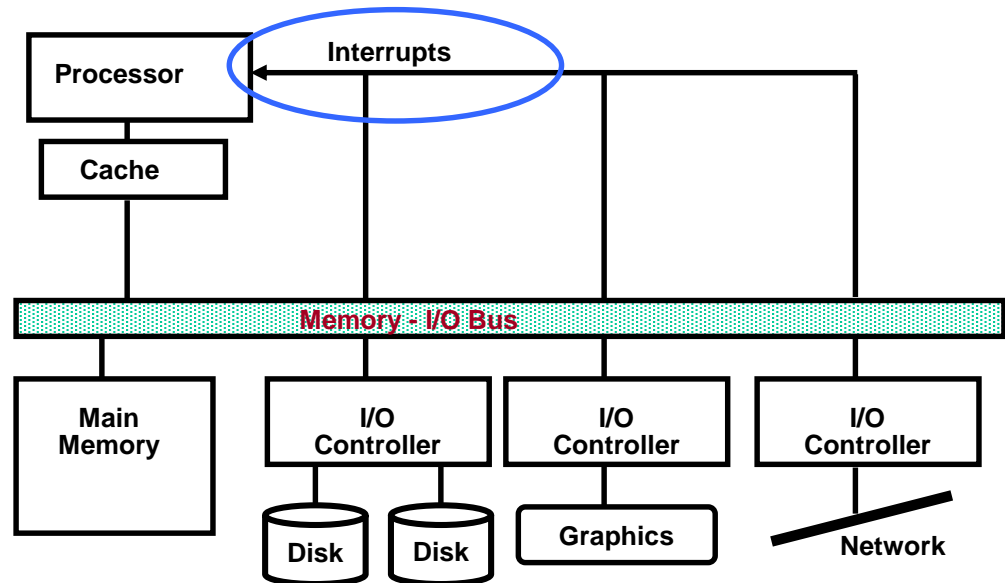
# Interrupt-Driven I/O

- An I/O interrupt is **asynchronous**
  - Is not associated with any instruction so doesn't prevent any instruction from completing
    - You can pick your own convenient point to handle the interrupt
- With I/O interrupts
  - Need a way to identify the device generating the interrupt
  - Can have different urgencies (so may need to be prioritized)
- **Advantages** of using interrupts
  - No need to continuously poll for an I/O event; user program progress is only suspended during the actual transfer of I/O data to/from user memory space
- **Disadvantage** – special hardware is needed to
  - Cause an interrupt (I/O device) and detect an interrupt and save the necessary information to resume normal processing after servicing the interrupt (processor)



# Direct Memory Access (DMA)

- For high-bandwidth devices (like disks) **interrupt-driven I/O** would consume a lot of processor cycles
- DMA – the I/O controller has the ability to transfer data **directly** to/from the memory without involving the processor
- There may be multiple DMA devices in one system



# Direct Memory Access (DMA) how to?

1. The processor initiates the DMA transfer by supplying
  1. the I/O device address
  2. the operation to be performed
  3. the memory address destination/source
  4. the number of bytes to transfer.
2. The I/O DMA controller manages the entire transfer arbitrating for the bus
3. When the DMA transfer is complete, the I/O controller **interrupts** the processor to let it know that the transfer is complete

- **Cache Coherence**



# I/O and the Operating System

- The operating system acts as the interface between the I/O hardware and the program requesting I/O
  - To protect the **shared I/O resources**, the user program is not allowed to communicate directly with the I/O device
- Thus OS must be able to give commands to I/O devices, handle interrupts generated by I/O devices, provide fair access to the shared I/O resources, and schedule I/O requests to enhance system throughput
  - I/O interrupts result in a transfer of processor control to the **supervisor (OS) process**

