



Introduction to Computer Science

Polly Huang
NTU EE

<http://homepage.ntu.edu.tw/~pollyhuang>
pollyhuang@ntu.edu.tw

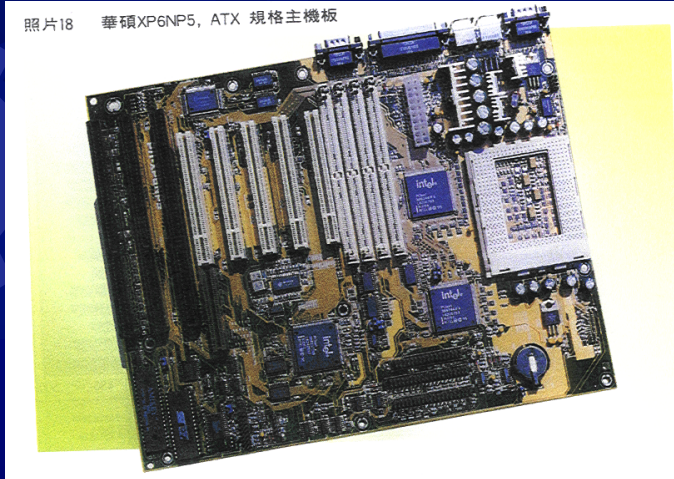


Chapter 2

Data Manipulation
(Machine Architecture)

Basically, about this thing...

照片18 華碩XP6NP5, ATX 規格主機板

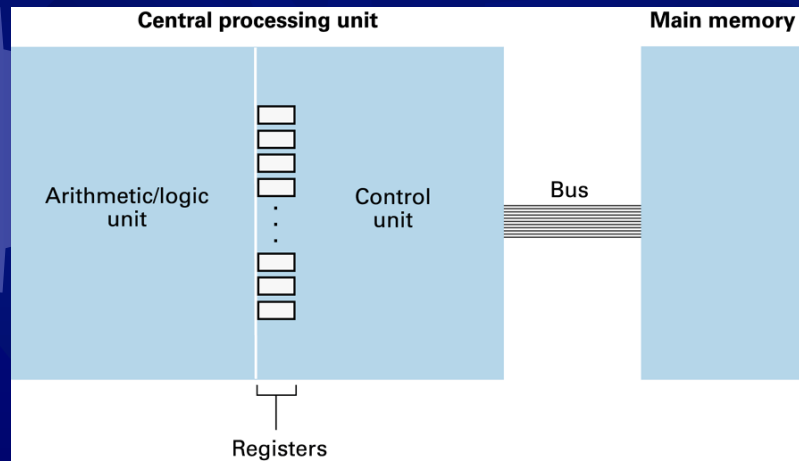


Credit: 陳文欽, 陳兆宏, 97DIY 電腦選購與組裝, 旗標, 1997

Chapter 2: Data Manipulation

- ✦ 2.1 Computer Architecture
- ✦ 2.2 Machine Language
- ✦ 2.3 Program Execution
- ✦ 2.4 Arithmetic/Logic Instructions
- ✦ 2.5 Communicating with Other Devices
- ✦ 2.6 Programming Data Manipulation
- ✦ 2.7 Other Architectures

A Simple Machine



Polly Huang, NTU EE

Chapter 2

5

Computer Architecture

- ☀ Central Processing Unit (CPU) or processor
 - Arithmetic/Logic Unit (ALU)
 - Control Unit
 - Registers
 - (Cache Memory)
- ☀ Bus
- ☀ Main Memory
- ☀ Motherboard

Polly Huang, NTU EE

Chapter 2

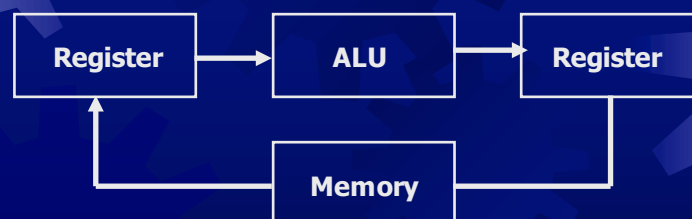
6

Central Processing Unit

- ☀ Circuitry that performs **operations** on data
- ☀ Two major operations
 - Arithmetic/logic unit (ALU)
 - Performs data manipulation
 - Boolean operations for example
 - Control unit
 - Coordinates the machine's activities

Registers

- ☀ Temporary **holding places** for data being manipulated by the CPU



Bus



Credit: Jill Kuhlman

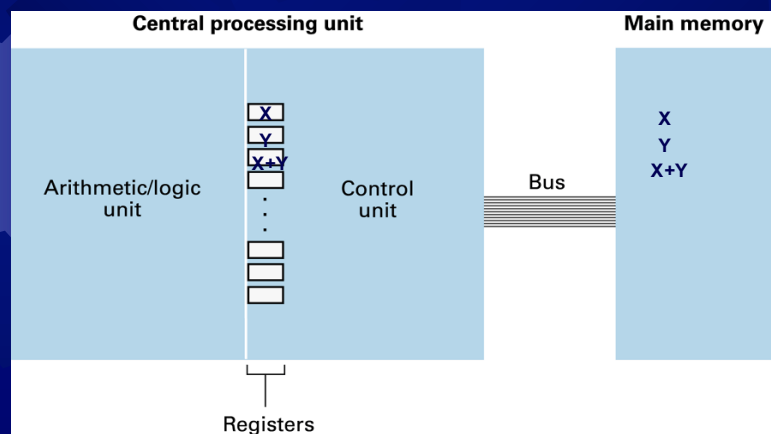
Bus

- Through the bus, the CPU is able to
 - Extract data from main memory
 - By supplying the address of the memory cell
 - Along with a **read** signal
- Similarly, the CPU can
 - Place data in memory
 - By providing the address of the destination cell and the data to be stored
 - With a **write** signal

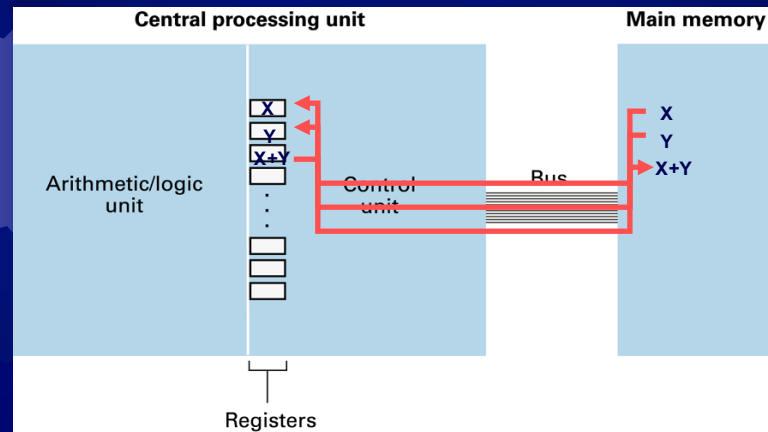
Adding from Data in Memory

- Step 1.** Get one of the values to be added from memory and place it in a register.
- Step 2.** Get the other value to be added from memory and place it in another register.
- Step 3.** Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.
- Step 4.** Store the result in memory.
- Step 5.** Stop.

The Flow of Adding X and Y



The Flow of Adding X and Y



Stored Program Concept

- ☀ A program is just a special type of data.
 - A program can be stored in main memory.
- ☀ One general-purpose machine can run many different programs.
- ☀ The standard approach to build today's computers

Representing Programs

- ☀ A program is composed by instructions
- ☀ An instruction contains an operation and data to operate
- ☀ Data are represented as numbers
 - From Chapter 1
- ☀ Operations are coded by numbers
 - Section 2.2

Stored Program

- ☀ Programs are stored in memory to be read or written just like numbers
- ☀ Flexible and reliable
- ☀ No need to have special hardware to store programs

Chapter 2: Data Manipulation

- ☀ 2.1 Computer Architecture
- ☀ 2.2 Machine Language
- ☀ 2.3 Program Execution
- ☀ 2.4 Arithmetic/Logic Instructions
- ☀ 2.5 Communicating with Other Devices
- ☀ 2.6 Programming Data Manipulation
- ☀ 2.7 Other Architectures

Machine Language

- ☀ **Machine instruction**
 - An instruction coded as a bit pattern directly recognizable by the CPU
- ☀ **Machine language**
 - The set of all instructions recognized by a machine

Machine Instruction Types

- ☀ Data Transfer
 - Copy data between CPU and main memory
 - E.g., LOAD, STORE, I/O (Section 2.5)
- ☀ Arithmetic/Logic
 - Use existing data values to compute a new data value
 - E.g., AND, OR, XOR, ADD
 - SHIFT (to the right), ROTATE (fill from the left)
- ☀ Control
 - Direct the execution of the program
 - E.g., JUMP (to instructions not in sequence)

Dividing Values

- Step 1.** LOAD a register with a value from memory.
- Step 2.** LOAD another register with another value from memory.
- Step 3.** If this second value is zero, JUMP to Step 6.
- Step 4.** Divide the contents of the first register by the second register and leave the result in a third register.
- Step 5.** STORE the contents of the third register in memory.
- Step 6.** STOP.

Machine Language Philosophies

- ☀ Reduced Instruction Set Computing
 - RISC
 - Minimalist's approach
- ☀ Complex Instruction Set Computing
 - CISC
 - Opportunist's approach

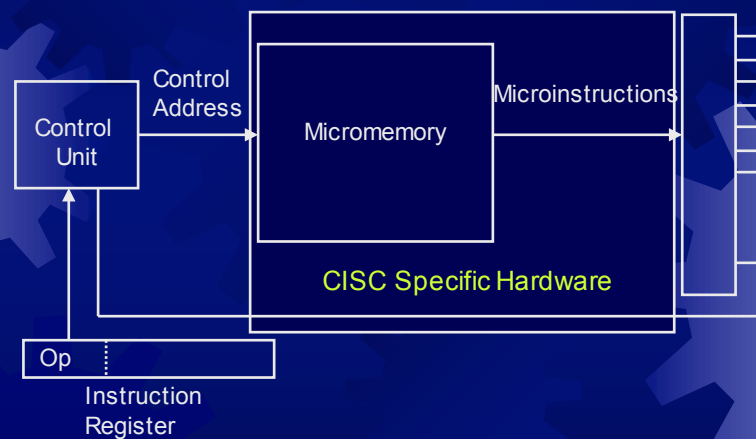
RISC

- ☀ Minimalist's approach
- ☀ Few, simple instructions
- ☀ Advantages
 - Efficient and fast
- ☀ Disadvantages
 - One way to code anything, longer programs
- ☀ Example
 - PowerPC from Apple/IBM/Motorola

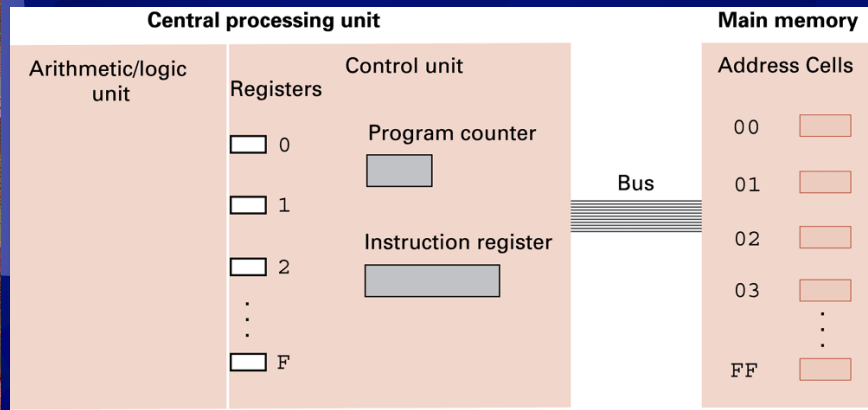
CISC

- Opportunist's approach
- Many, convenient, and powerful instructions
- Advantage
 - Multiple ways to code, easier to program, shorter programs
- Disadvantages
 - Less efficient and slow
 - Extra hardware
- Example
 - Pentium from Intel

Extra Hardware in CISC



A Simple Machine (Appendix C)



A Simple Machine

- ★ Machine Hardware
 - Program counter
 - Two types of registers
 - Special-purpose instruction register
 - 16 general-purpose registers
 - 256 memory cells with 8-bit each
- ★ Machine Language
 - Each instruction is 16 bits (FFFF)
 - Size of the instruction register
 - 12 basic instructions
 - Appendix C (page 581-582)

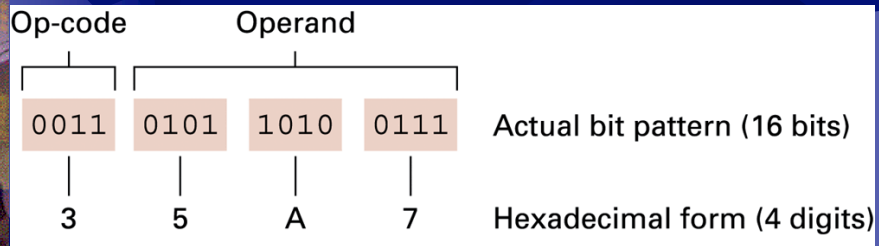
PC and IR

- ☀ Program counter
 - Contains the address of the next instruction to be executed
 - Keep track of where it is in the program
- ☀ Instruction register
 - Hold the instruction being executed

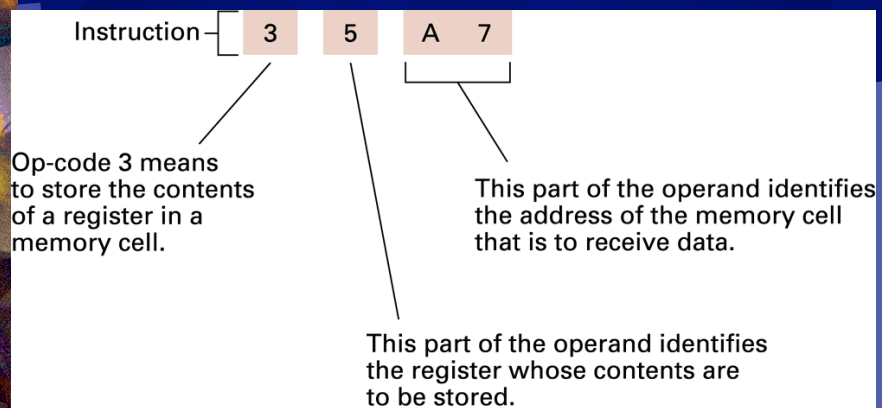
Parts of a machine instruction

- ☀ Op-code
 - Machine operation to execute
 - One per instruction
- ☀ Operand
 - More detailed information about this operation
 - Number of operands varies depending on op-code

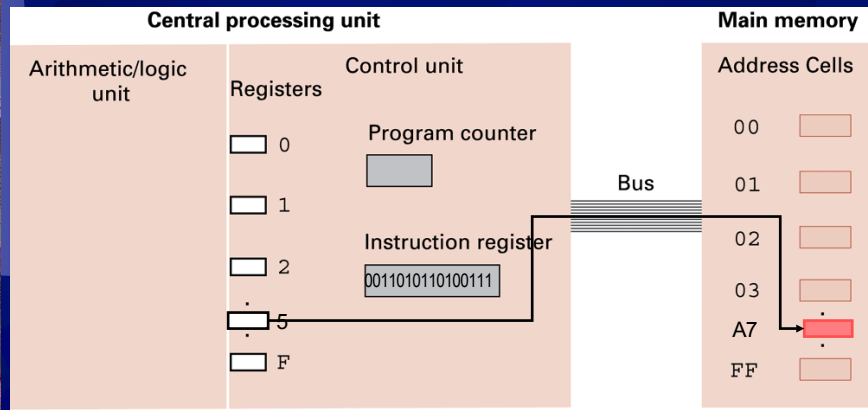
An instruction



Decoding the Instruction 35A7



The Action of 35A7



Appendix C Language

- ☀ 1RXY – $R \leftarrow XY$
- ☀ 2RXY – $R \leftarrow (XY)$
- ☀ 3RXY – $XY \leftarrow R$
- ☀ 40RS – $S \leftarrow R$
- ☀ 5RST – $R \leftarrow S +_i T$
- ☀ 6RST – $R \leftarrow S +_f T$

Appendix C Language (cont.)

- ☀ 7RST – $R \leftarrow \text{OR}(S,T)$
- ☀ 8RST – $R \leftarrow \text{AND}(S,T)$
- ☀ 9RST – $R \leftarrow \text{XOR}(S,T)$
- ☀ AR0X – $R \leftarrow \text{ROTATE}(R,X \text{ bits})$
- ☀ BRXY – If $R0=R$, $\text{PC} \leftarrow XY$
- ☀ C000 – HALT

Example Program (A+B=C)

Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.



Double Quiz Time!



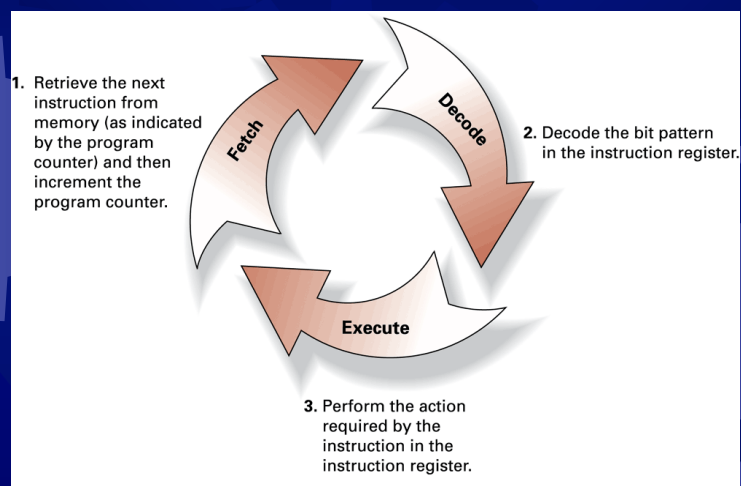
Chapter 2: Data Manipulation

- ☀ 2.1 Computer Architecture
- ☀ 2.2 Machine Language
- ☀ 2.3 Program Execution
- ☀ 2.4 Arithmetic/Logic Instructions
- ☀ 2.5 Communicating with Other Devices
- ☀ 2.6 Programming Data Manipulation
- ☀ 2.7 Other Architectures

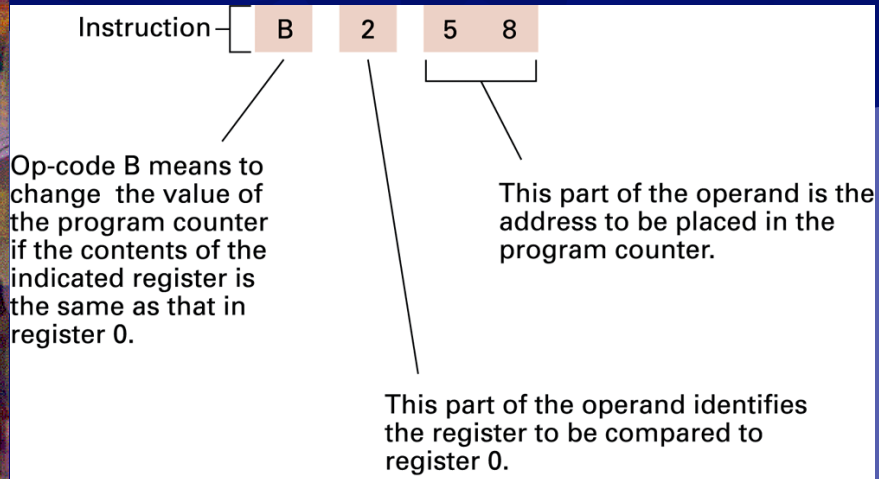
Program Execution

- ☀ Controlled by two special-purpose registers
 - Program counter: address of next instruction
 - Instruction register: current instruction
- ☀ Steps performed by control unit
 - Fetch
 - Decode
 - Execute

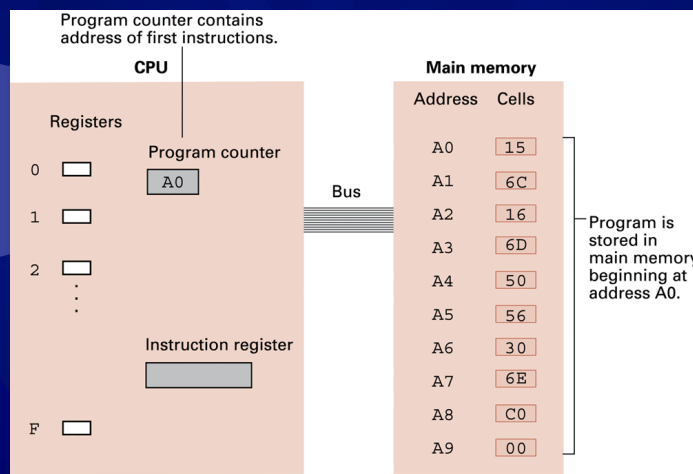
The Machine Cycle



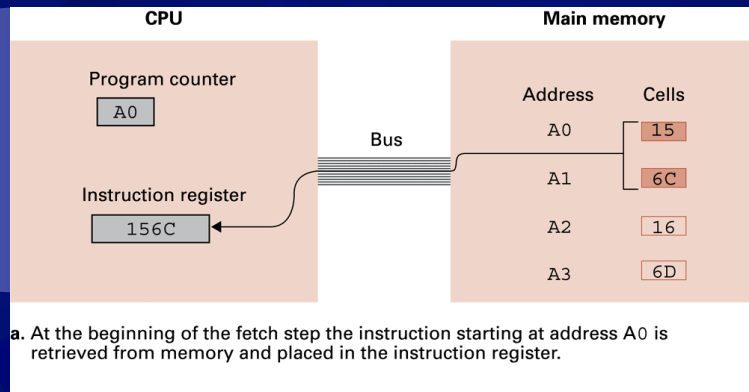
Decoding the Instruction B258



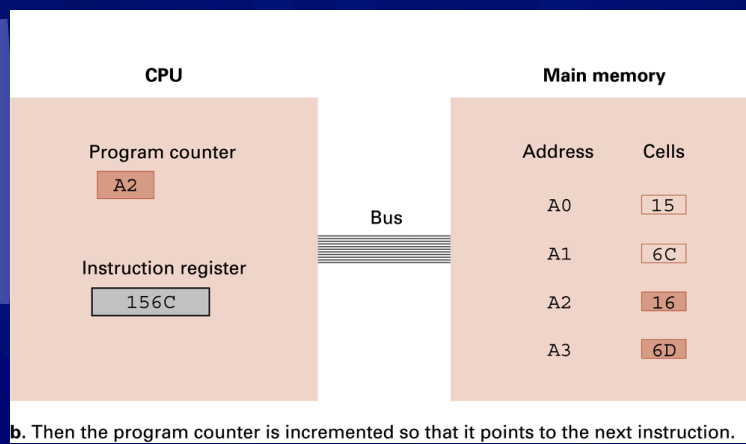
The ADD program in Memory



Fetch



Fetch (cont'd)



Execution of the Program

Fetch		Decode & Execute
IR	PC	
	A0	
156C	A2	$R5 \leftarrow 6C$
166D	A4	$R6 \leftarrow 6D$
5056	A6	$R0 \leftarrow R5+R6$
306E	A8	$6E \leftarrow R0$
C000	AA	HALT

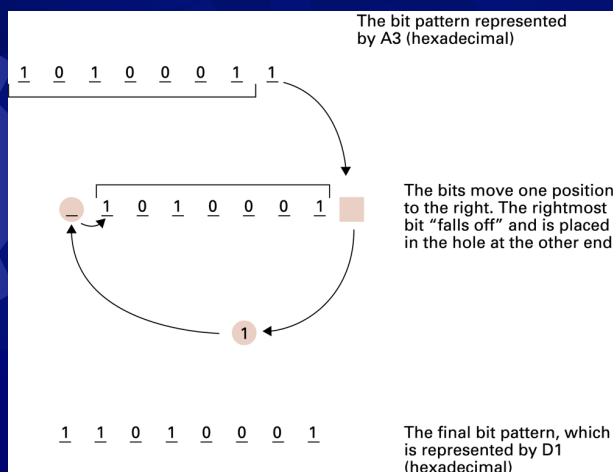
Chapter 2: Data Manipulation

- ☀ 2.1 Computer Architecture
- ☀ 2.2 Machine Language
- ☀ 2.3 Program Execution
- ☀ 2.4 Arithmetic/Logic Instructions
- ☀ 2.5 Communicating with Other Devices
- ☀ 2.6 Programming Data Manipulation
- ☀ 2.7 Other Architectures

Arithmetic/Logic Operations

- ☀ Logic
 - AND, OR, XOR
- ☀ Rotate and Shift
 - Rotate: Circular shift
 - Shift: Logical shift (left shift)
- ☀ Arithmetic
 - Add, subtract, multiply, divide
 - Often separate instructions for different types of data

Rotating A3 1 Bit to the Right





Double Quiz Time!



Chapter 2: Data Manipulation

- ☀ 2.1 Computer Architecture
- ☀ 2.2 Machine Language
- ☀ 2.3 Program Execution
- ☀ 2.4 Arithmetic/Logic Instructions
- ☀ 2.5 Communicating with Other Devices
- ☀ 2.6 Programming Data Manipulation
- ☀ 2.7 Other Architectures

Communicating with Devices

- ☀ Controller

- Intermediary device that handles communication between the computer and a peripheral device.

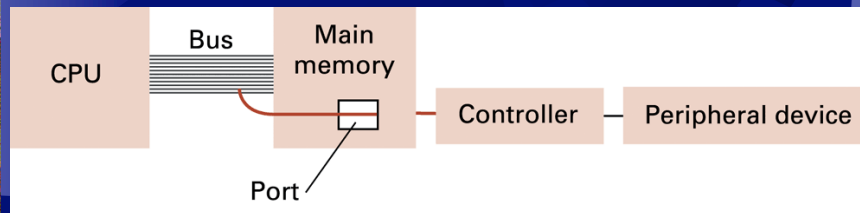
- ☀ Port

- Set of addresses assigned to a device.
- E.g., COM1, COM2

Two I/O Access Methods

- ☀ Memory-mapped I/O: CPU reads from or writes to addresses of the controller's port
- ☀ Bus I/O: CPU reads from or writes directly to the controller

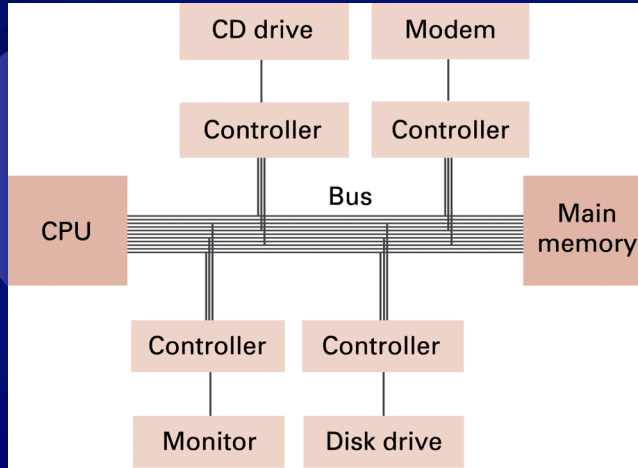
Memory-Mapped I/O



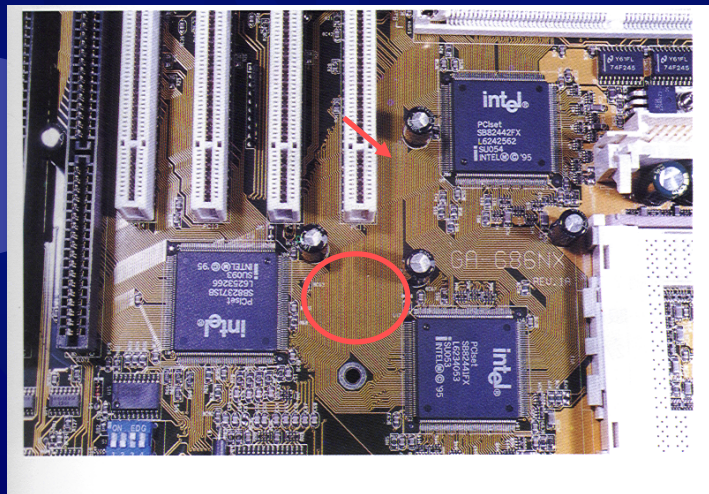
Bus I/O

- ☀ Direct memory access (DMA): main memory access by a controller over the bus

Controllers Attaching to Bus



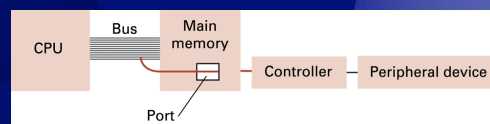
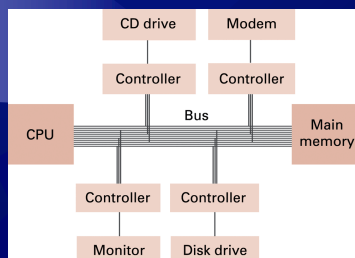
Buses Everywhere



Quiz Time!

Consider this Situation

- ☀ Concurrently trying to do these:
 1. Moving data from hard disk to memory
 2. Moving data from memory to CPU



Performance Bottleneck

- ☀ Von Neumann Bottleneck
 - Insufficient bus/memory speed degrades performance
- ☀ Solution
 - Handshaking
 - The process of controlling the transfer of data between components at different speeds

CPU and Peripheral Devices

- ☀ Two-way dialog is needed
- ☀ Status word
 - Bit pattern generated by the peripheral device and sent to the controller
 - Reflects the condition of the device
 - Example: printer is out of paper or ready for additional data
 - Controller respond to status word itself or make it available to the CPU

Data Communication Rates

- ☀ Measurement units
 - bps = bits per second
 - kbps = kilo-bps, or 1,000 bps
 - mbps = mega-bps, or 1,000,000 bps
 - gbps = giga-bps, or 1,000,000,000 bps
- ☀ Bandwidth = maximum available rate
- ☀ Baud rate
 - At which the communication line transfers states
 - A state may be composed of several bits
 - Status words

Improving Efficiency

- ☀ I.e., making the transmission time shorter
- ☀ Data compression
 - Make it a less amount of data to send
 - Talked briefly about this in Chapter 1
- ☀ Higher speed transmission
 - Send more data per unit time

Communication Path Types

- ☀ Serial
 - One line transfers one bit at a time
 - E.g., modem, mouse, keyboard
- ☀ Parallel
 - Several lines transfer different bits simultaneously
 - E.g., printer
- ☀ USB 2.0/3.0
- ☀ Modem/DSL/Cable/Fiber

Chapter 2: Data Manipulation

- ☀ 2.1 Computer Architecture
- ☀ 2.2 Machine Language
- ☀ 2.3 Program Execution
- ☀ 2.4 Arithmetic/Logic Instructions
- ☀ 2.5 Communicating with Other Devices
- ☀ 2.6 Programming Data Manipulation
- ☀ 2.7 Other Architectures

Programming Data Manipulation

- ☀ Programming languages shields users from details of the machine:
 - A single `Python` statement might map to one, tens, or hundreds of machine instructions
 - Programmer does not need to know if the processor is RISC or CISC
 - Assigning variables surely involves `LOAD`, `STORE`, and `MOVE` op-codes

2-63

Bitwise Problems

- ☀ `bin(0b10011010 & 0b11001001)`
- ☀ `bin(0b10011010 | 0b11001001)`
- ☀ `bin(0b10011010 ^ 0b11001001)`

Polly Huang, NTU EE

Chapter 2

64

Printing

```
print(bin(0b10011010 & 0b11001001))  
# Prints '0b10001000'  
  
print(bin(0b10011010 | 0b11001001))  
# Prints '0b11011011'  
  
print(bin(0b10011010 ^ 0b11001001))  
# Prints '0b1010011'
```

2-65

Control Structures

☀ If statement

```
if (water_temp > 140):  
    print('Bath water too hot!')
```

☀ While statement

```
while (n < 10):  
    print(n)  
    n = n + 1
```

Polly Huang, NTU EE

Chapter 2

66

Functions

☀ Function

- A series of operations that should be performed on the given parameter(s)

☀ Function call

- Appearance of a function in an expression or statement

```
x = 1034
y = 1056
z = 2078
biggest = max(x, y, z) # function call
```

2 Kinds

☀ Fruitful functions

- Return a value

☀ void functions (or procedures)

- Not return a value

```
sideA = 3.0
sideB = 4.0
# Calculate third side via Pythagorean Theorem
hypotenuse = math.sqrt(sideA**2 + sideB**2)
print(hypotenuse)
```

Input / Output

```
# Calculates the hypotenuse of a right
triangle
import math

# Inputting the side lengths, first try
sideA = int(input('Length of side A? '))
sideB = int(input('Length of side B? '))

# Calculate third side via Pythagorean Theorem
hypotenuse = math.sqrt(sideA**2 + sideB**2)
print(hypotenuse)
```

2-69

Marathon Training Assistant I

```
# Marathon training assistant.
import math

# This function converts a number of minutes and
# seconds into just seconds.
def total_seconds(min, sec):
    return min * 60 + sec

# This function calculates a speed in miles per hour given
# a time (in seconds) to run a single mile.
def speed(time):
    return 3600 / time
```

2-70

Marathon Training Assistant II

```
# Prompt user for pace and mileage.
pace_minutes = int(input('Minutes per mile? '))
pace_seconds = int(input('Seconds per mile? '))
miles = int(input('Total miles? '))

# Calculate and print speed.
mph = speed(total_seconds(pace_minutes, pace_seconds))
print('Your speed is ' + str(mph) + ' mph')

# Calculate elapsed time for planned workout.
total = miles * total_seconds(pace_minutes, pace_seconds)
elapsed_minutes = total // 60
elapsed_seconds = total % 60

print('Your elapsed time is ' + str(elapsed_minutes) +
      ' mins ' + str(elapsed_seconds) + ' secs')
```

2-71

Example Marathon Training Data

Time Per Mile				Total Elapsed Time	
Minutes	Seconds	Miles	Speed (mph)	Minutes	Seconds
9	14	5	6.49819494584	46	10
8	0	3	7.5	24	0
7	45	6	7.74193548387	46	30
7	25	1	8.08988764044	7	25

2-72



Quiz Time!



Chapter 2: Data Manipulation

- ☀ 2.1 Computer Architecture
- ☀ 2.2 Machine Language
- ☀ 2.3 Program Execution
- ☀ 2.4 Arithmetic/Logic Instructions
- ☀ 2.5 Communicating with Other Devices
- ☀ 2.6 Programming Data Manipulation
- ☀ **2.7 Other Architectures**

Limitation of Transmission

- ☀ Electric pulses travel through a wire no faster than the speed of light
- ☀ The distance between CPU and memory limits the speed of fetching and executing executions
- ☀ Increasing the execution speed becomes a miniaturization problem
 - Shortening the distance

Other Architectures

- ☀ Throughput = (total work) / (total time)
- ☀ Technologies to increase throughput
 - Pipelining
 - Parallel processing

Pipelining

- ☀ Allowing the steps in the machine cycle to overlap
- ☀ Total throughput is increased



Polly Huang, NTU EE

Chapter 2

77

Drawback of Pipelining

- ☀ Whenever there are JUMP instructions
- ☀ Gain by pre-fetching vanishes

Polly Huang, NTU EE

Chapter 2

78

A presentation slide with a dark blue background featuring a pattern of interlocking gears. The text "Quiz Time!" is centered in a light yellow font. At the bottom, the text "Polly Huang, NTU EE", "Chapter 2", and "79" is displayed in a small white font. A vertical decorative strip on the left side shows a close-up of colorful gears.

Quiz Time!

Polly Huang, NTU EE Chapter 2 79

A presentation slide with a dark blue background featuring a pattern of interlocking gears. The text "Side Bar..." is centered in a light yellow font, and "Modern Computers" is centered below it in a white font. At the bottom, the text "Polly Huang, NTU EE", "Chapter 2", and "80" is displayed in a small white font. A vertical decorative strip on the left side shows a close-up of colorful gears.

Side Bar...

Modern Computers

Polly Huang, NTU EE Chapter 2 80

Parallel Processing

- ☀ Performance of several activities at the same time
- ☀ Requires more than one processing unit
- ☀ Multi-processor machines

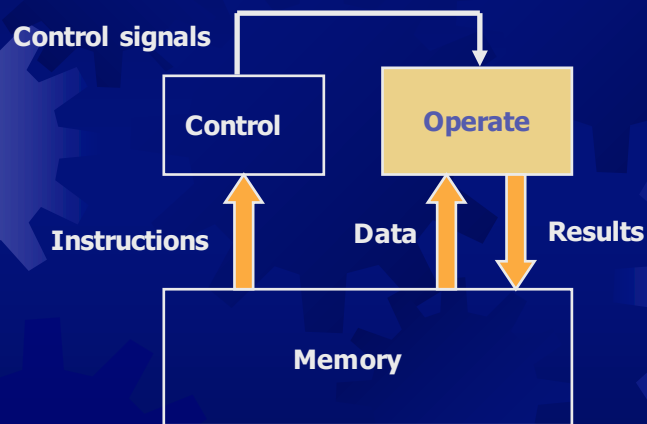


● Processor:
CPU+Memory

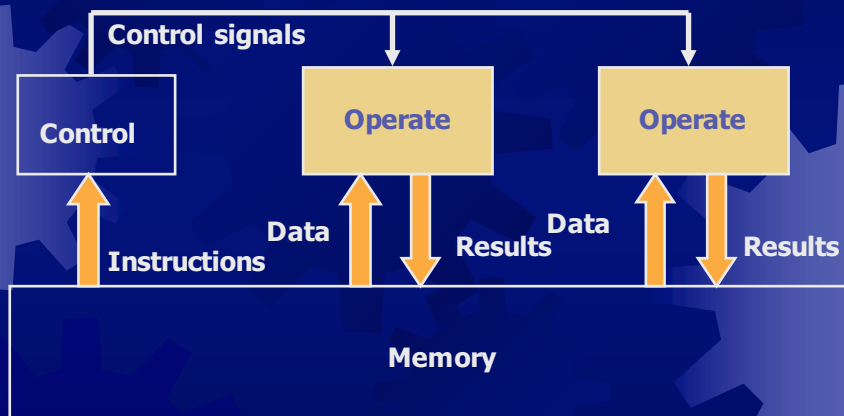
Classification of Architectures

- ☀ SISD
 - Single-instruction stream, single-data stream
- ☀ SIMD
 - Single-instruction stream, multiple-data stream
- ☀ MIMD
 - Multiple-instruction stream, multiple-data stream
- ☀ Other architectures
 - Dataflow, neural networks, special-purpose

SISD von Neumann Machine



SIMD Processor Array

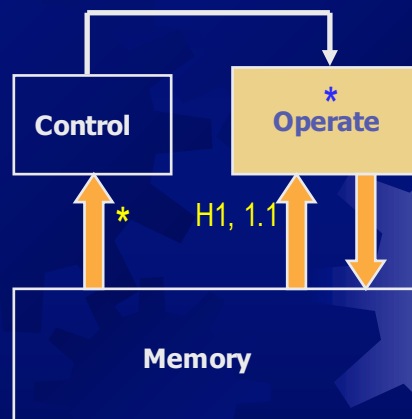


SIMD Exercise

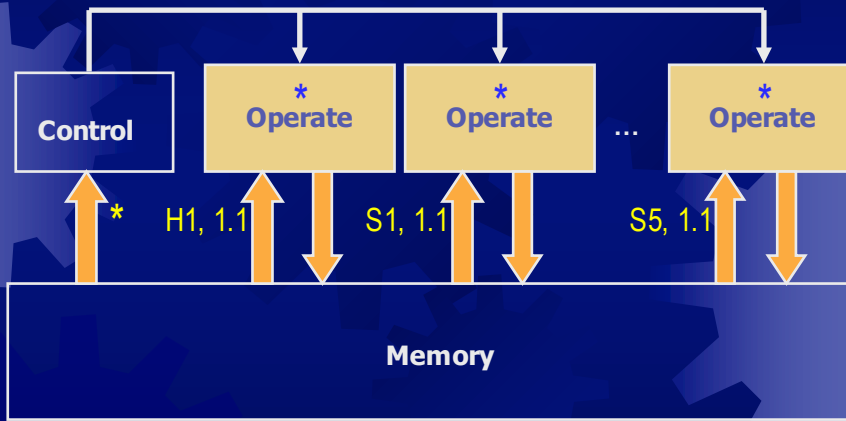
- There's a company with 5 departments
- Each department has a department head and a staff member
- Their annual pays are denoted as H1, S1, H2, S2, ...H5, S5
- Each employee gets an 10% raise annually
- If this company uses a simple PC to handle the payroll, how many computation cycles would it take to compute the new salaries?
- If this company uses a 10-processor SIMD server to handle the payroll, how would you allocate the computation of giving raises so it will save time? And how many computation cycle would it take to compute the new salaries?

SISD Case

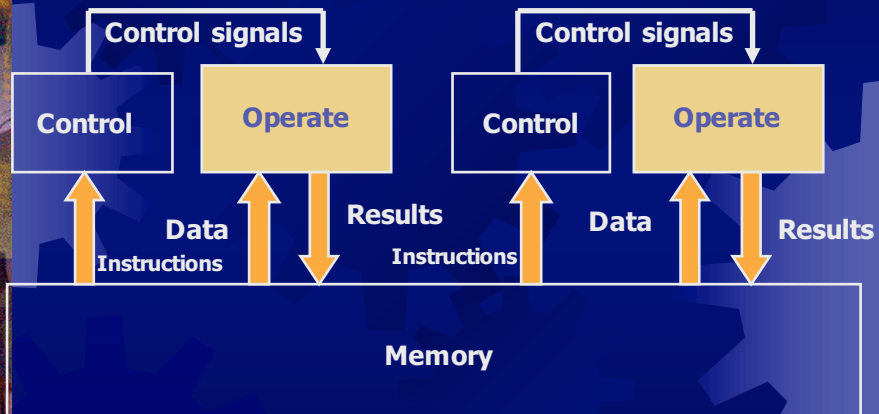
- $H1 * 1.1$
- $S1 * 1.1$
- $H2 * 1.1$
- $S2 * 1.1$
- ...



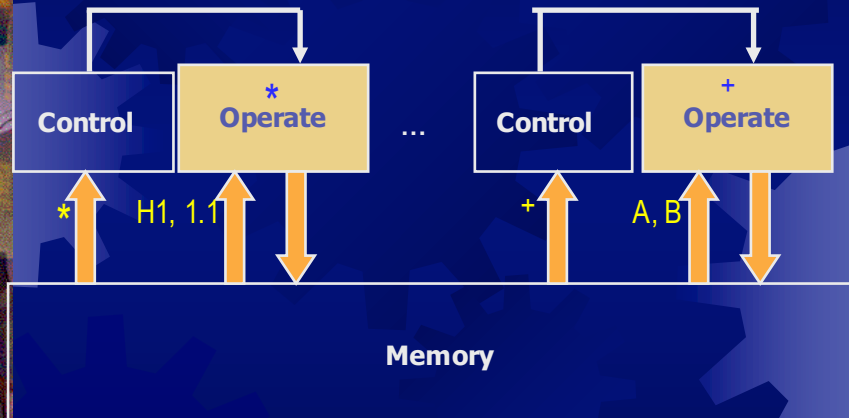
SIMD Case



MIMD Multiprocessor System

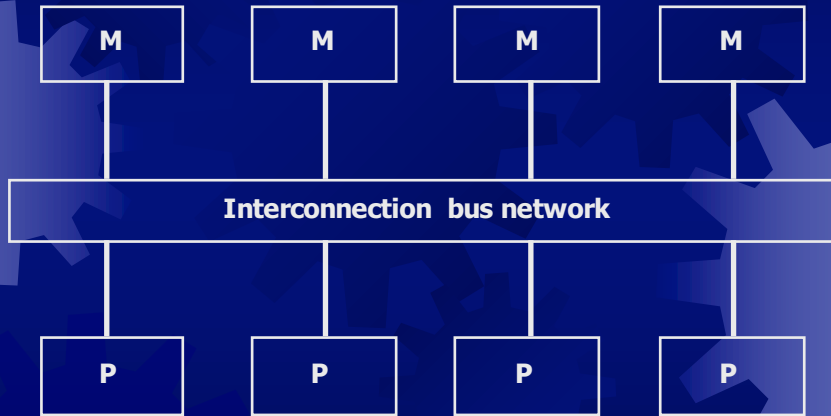


MIMD Case



Quiz Time!

Tightly Coupled MIMD



Polly Huang, NTU EE

Chapter 2

91

Ex: Multi-Processor Server



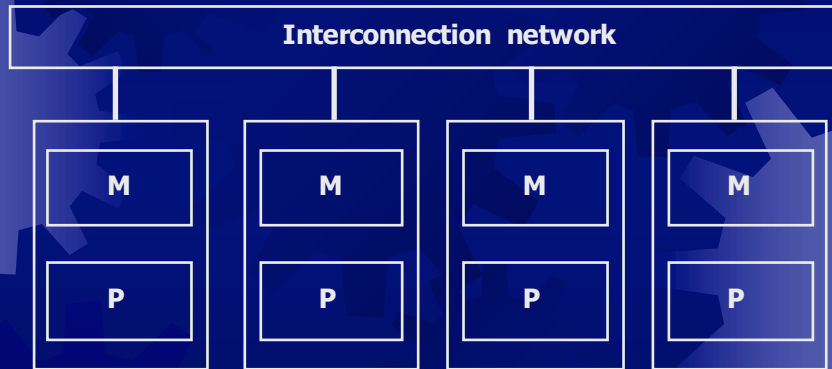
Credit: IBM xServer Series.

Polly Huang, NTU EE

Chapter 2

92

Loosely Coupled MIMD

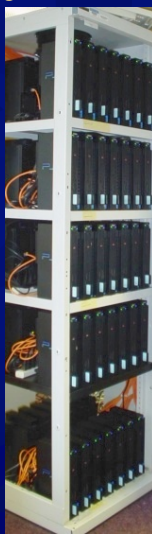


Polly Huang, NTU EE

Chapter 2

93

Ex: PlayStation2 Cluster



Credit: Sony PS2 Linux Cluster

<http://arrakis.ncsa.uiuc.edu/ps2/cluster.php>

Polly Huang, NTU EE

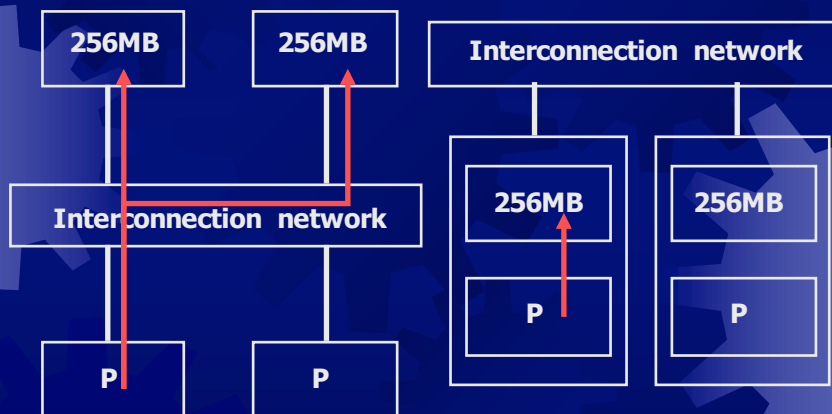
Chapter 2

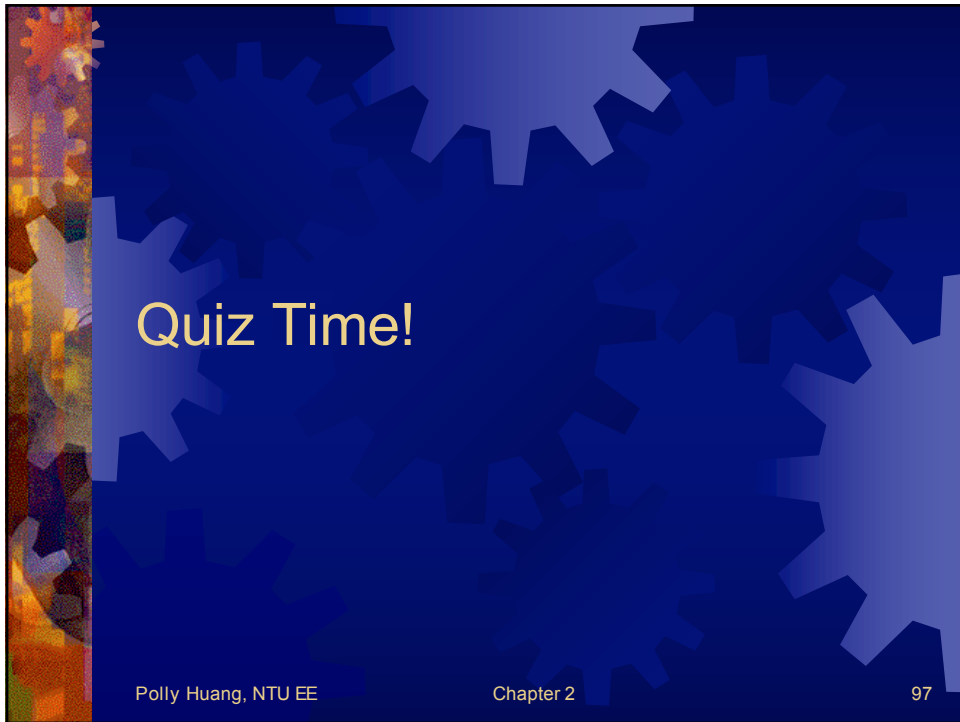
94

Distinctions

- ☀ Loosely coupled
 - Dedicated memory for a processor
 - Cannot do very big jobs that require a high amount of memory
 - Easy to manage
- ☀ Tightly coupled
 - Processors sharing memory space
 - Can do bigger jobs
 - Difficult to manage

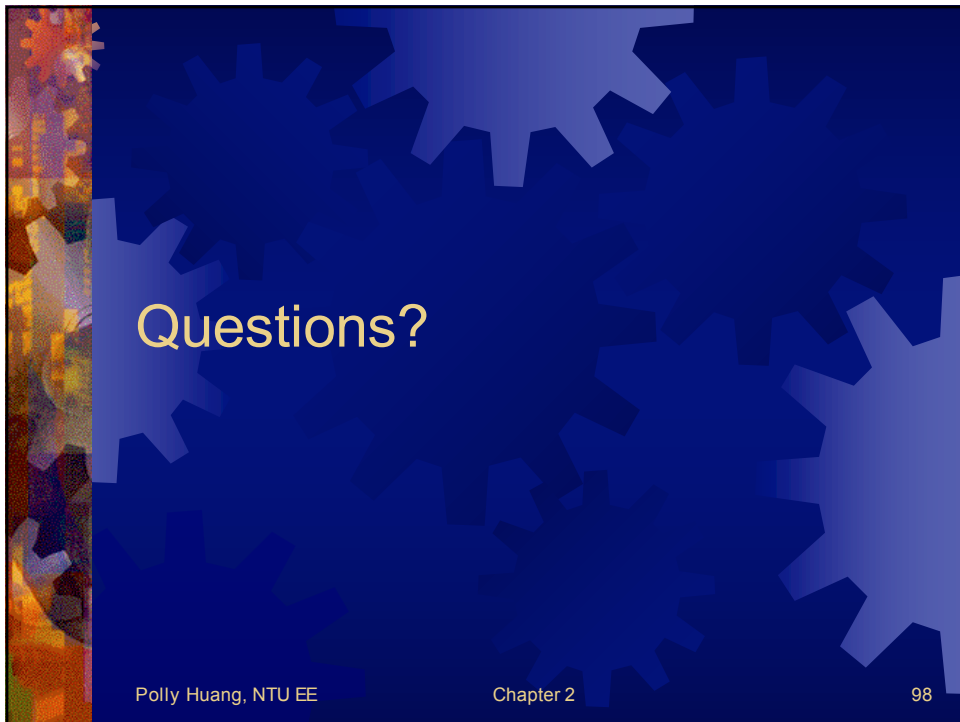
Shared vs. Dedicated Memory



A presentation slide with a dark blue background featuring a pattern of interlocking gears. The text "Quiz Time!" is centered in a light yellow font. At the bottom, there is a footer with the text "Polly Huang, NTU EE", "Chapter 2", and "97".

Quiz Time!

Polly Huang, NTU EE Chapter 2 97

A presentation slide with a dark blue background featuring a pattern of interlocking gears. The text "Questions?" is centered in a light yellow font. At the bottom, there is a footer with the text "Polly Huang, NTU EE", "Chapter 2", and "98".

Questions?

Polly Huang, NTU EE Chapter 2 98