

计算机组成和体系结构

第二讲

四川大学网络空间安全学院

2020年3月2日

*封面图片来自Pexels.com，作者Jeremy Waterhouse

版权声明

课件中所使用的图片、视频等资源版权归原作者所有。

课件原创内容采用 [创作共用署名—非商业使用—相同方式共享4.0国际版许可证\(Creative Commons BY-NC-SA 4.0 International License\)](#) 授权使用。

Copyright@四川大学网络空间安全学院计算机组成与体系结构课程组，2020



上期内容回顾

- 初识计算机系统
 - 现代通用计算机系统的主要组成部分
 - 常见性能指标
 - 前缀乘数
- 计算机发展历史
 - 四代计算机：电子管计算机，晶体管计算机，集成电路计算机，超大规模集成电路计算机
 - 芯片集成度、摩尔定律、罗克定律
- 计算机层次化结构与云计算
 - 用户层、高级编程语言层、汇编语言层、系统软件层、机器层（指令集架构层）、控制层、数字逻辑层
 - 常见云计算类型与层次化结构的对应关系
- 冯-诺依曼模型
 - 冯-诺依曼模型的基本组成和主要特征
 - 取-译码-执行的过程

本期学习目标

- 绪论5: 非冯-诺依曼模型和并行计算
 - 了解几类常见的非冯-诺依曼模型
 - 初步了解一些并行计算系统的组成和体系结构
- 数字电子技术回顾
 - 复习进制转换、整型和浮点数的二进制表示、字符编码和错误检测
 - 复习布尔代数
- 计算机模型
 - 掌握CPU基本知识和组成结构

中英文缩写对照表

英文缩写	英文全称	中文全称
ALU	Arithmetic-Logic Unit	算数逻辑单元
APU	Accelerated Processing Unit	加速处理器
APU	AI Processing Unit	人工智能处理器
NPU	Neural Processing Unit	神经网络处理器
TPU	Tensor Processing Unit	TensorFlow处理器
CPU	Central Processing Unit	中央处理器
DSP	Digital Signal Processor	数字信号处理器
GPU	Graphic Processing Unit	图形处理器
NPU	Network Processing Unit	网络处理器
SIMD	Single-Instruction, Multiple-Data	单指令流多数据流
MIMD	Multiple-Instruction, Multiple-Data	多指令流多数据流
MUX	Multiplexer	多路复用器

绪论5: 非冯-诺依曼模型和并行计算

单处理器非冯-诺依曼模型

冯-诺依曼模型的特征

- 由CPU、存储器和I/O设备构成
- 指令和数据放在同一个存储器中
- 读取指令和传送数据采用同一个总线
- 顺序执行指令

扩展冯-诺依曼模型

- 系统总线模型中采用三条总线
 - 数据总线、地址总线和控制总线

非冯-诺依曼模型

- 哈佛架构 (Harvard Architecture) 及其变体
- 并行处理器
- 领域专用计算芯片

上述分类依据的是不同的特征，因此并不是互斥的：例如GPU既属于并行处理器，同时又属于领域专用计算芯片

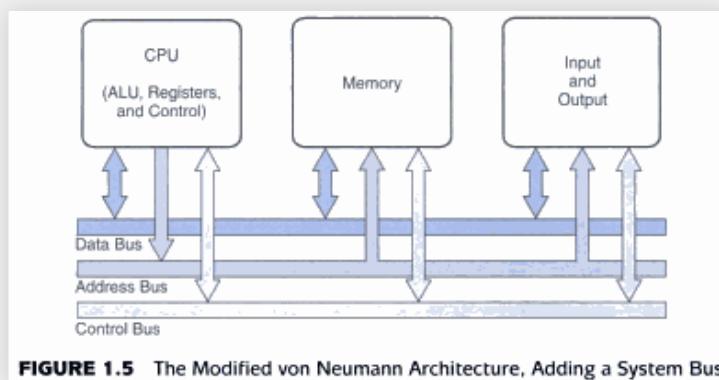
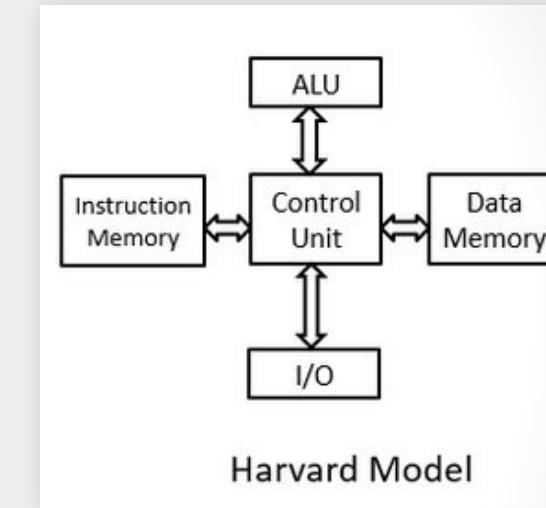


FIGURE 1.5 The Modified von Neumann Architecture, Adding a System Bus

哈佛结构及变体

- 指令和数据分开存储
- 采用两个总线分别传输指令和数据
- 系统总线模型也可以看作是哈佛架构的一种变体：没有分开存储指令和数据



Harvard Architecture

图片来源：<https://www.polytechnichub.com/difference-harvard-architecture-von-neumann-architecture/>

并行处理器

按照指令和数据的并行计算能力，将**单个处理器**分为下面四类

S I S D	S I M D
Single Instruction stream Single Data stream	Single Instruction stream Multiple Data stream
M I S D	M I M D
Multiple Instruction stream Single Data stream	Multiple Instruction stream Multiple Data stream

费林分类 (*Flynn's Taxonomy*,
Michael Flynn, 1966)

图片来源：Lawrence Livermore National Laboratory, <https://hpc.llnl.gov/tutorials/introduction-parallel-computing/flynns-classical-taxonomy>

并行处理器

按照指令和数据的并行计算能力，将**单个处理器**分为下面四类

- 单指令流单数据流 (Single-instruction, single-data, SISD)

S I S D	S I M D
Single Instruction stream Single Data stream	Single Instruction stream Multiple Data stream
M I S D	M I M D
Multiple Instruction stream Single Data stream	Multiple Instruction stream Multiple Data stream

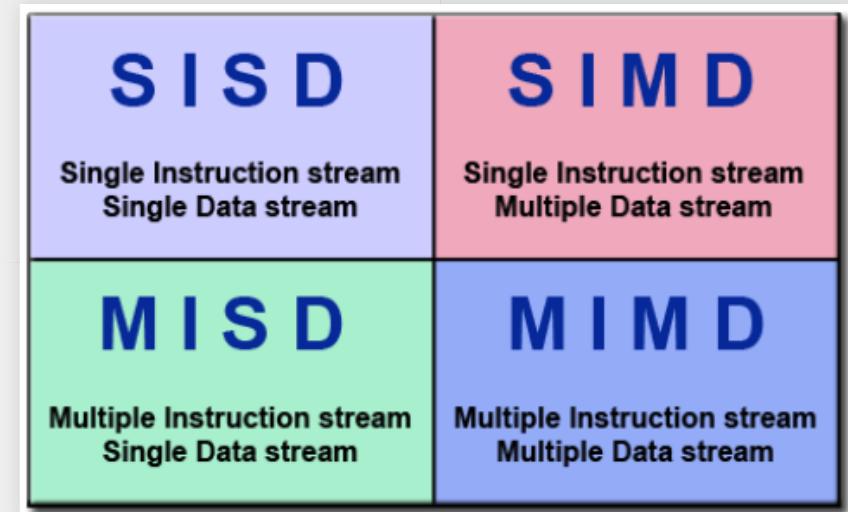
费林分类 (*Flynn's Taxonomy*,
Michael Flynn, 1966)

图片来源：Lawrence Livermore National Laboratory, <https://hpc.llnl.gov/tutorials/introduction-parallel-computing/flynns-classical-taxonomy>

并行处理器

按照指令和数据的并行计算能力，将**单个处理器**分为下面四类

- 单指令流单数据流 (Single-instruction, single-data, SISD)
 - 冯-诺依曼模型和哈佛架构都属于SISD



费林分类 (*Flynn's Taxonomy*,
Michael Flynn, 1966)

图片来源：Lawrence Livermore National Laboratory, <https://hpc.llnl.gov/tutorials/introduction-parallel-computing/flynns-classical-taxonomy>

并行处理器

按照指令和数据的并行计算能力，将**单个处理器**分为下面四类

- 单指令流单数据流 (Single-instruction, single-data, SISD)
 - 冯-诺依曼模型和哈佛架构都属于SISD
- 多指令流单数据流 (Multiple-instruction, single-data, MISD)

S I S D	S I M D
Single Instruction stream Single Data stream	Single Instruction stream Multiple Data stream
M I S D	M I M D
Multiple Instruction stream Single Data stream	Multiple Instruction stream Multiple Data stream

费林分类 (*Flynn's Taxonomy*,
Michael Flynn, 1966)

图片来源：Lawrence Livermore National Laboratory, <https://hpc.llnl.gov/tutorials/introduction-parallel-computing/flynns-classical-taxonomy>

并行处理器

按照指令和数据的并行计算能力，将**单个处理器**分为下面四类

- 单指令流单数据流 (Single-instruction, single-data, SISD)
 - 冯-诺依曼模型和哈佛架构都属于SISD
- 多指令流单数据流 (Multiple-instruction, single-data, MISD)
- 单指令流多数据流 (Single-instruction, multiple-data, SIMD)

S I S D	S I M D
Single Instruction stream Single Data stream	Single Instruction stream Multiple Data stream
M I S D	M I M D
Multiple Instruction stream Single Data stream	Multiple Instruction stream Multiple Data stream

费林分类 (*Flynn's Taxonomy*,
Michael Flynn, 1966)

图片来源：Lawrence Livermore National Laboratory, <https://hpc.llnl.gov/tutorials/introduction-parallel-computing/flynns-classical-taxonomy>

并行处理器

按照指令和数据的并行计算能力，将**单个处理器**分为下面四类

- 单指令流单数据流 (Single-instruction, single-data, SISD)
 - 冯-诺依曼模型和哈佛架构都属于SISD
- 多指令流单数据流 (Multiple-instruction, single-data, MISD)
- 单指令流多数据流 (Single-instruction, multiple-data, SIMD)
 - 现代CPU通常支持SIMD操作：例如Intel的AVX-512

S I S D	S I M D
Single Instruction stream Single Data stream	Single Instruction stream Multiple Data stream
M I S D	M I M D
Multiple Instruction stream Single Data stream	Multiple Instruction stream Multiple Data stream

费林分类 (*Flynn's Taxonomy*,
Michael Flynn, 1966)

图片来源：Lawrence Livermore National Laboratory, <https://hpc.llnl.gov/tutorials/introduction-parallel-computing/flynns-classical-taxonomy>

并行处理器

按照指令和数据的并行计算能力，将**单个处理器**分为下面四类

- 单指令流单数据流 (Single-instruction, single-data, SISD)
 - 冯-诺依曼模型和哈佛架构都属于SISD
- 多指令流单数据流 (Multiple-instruction, single-data, MISD)
- 单指令流多数据流 (Single-instruction, multiple-data, SIMD)
 - 现代CPU通常支持SIMD操作：例如Intel的AVX-512
- 多指令流多数据流 (Multiple-instruction, multiple-data, MIMD)

S I S D	S I M D
Single Instruction stream Single Data stream	Single Instruction stream Multiple Data stream
M I S D	M I M D
Multiple Instruction stream Single Data stream	Multiple Instruction stream Multiple Data stream

费林分类 (*Flynn's Taxonomy*,
Michael Flynn, 1966)

图片来源：Lawrence Livermore National Laboratory, <https://hpc.llnl.gov/tutorials/introduction-parallel-computing/flynns-classical-taxonomy>

并行处理器

按照指令和数据的并行计算能力，将**单个处理器**分为下面四类

- 单指令流单数据流 (Single-instruction, single-data, SISD)
 - 冯-诺依曼模型和哈佛架构都属于SISD
- 多指令流单数据流 (Multiple-instruction, single-data, MISD)
- 单指令流多数据流 (Single-instruction, multiple-data, SIMD)
 - 现代CPU通常支持SIMD操作：例如Intel的AVX-512
- 多指令流多数据流 (Multiple-instruction, multiple-data, MIMD)

后续课程中会详细介绍并行处理器相关的知识

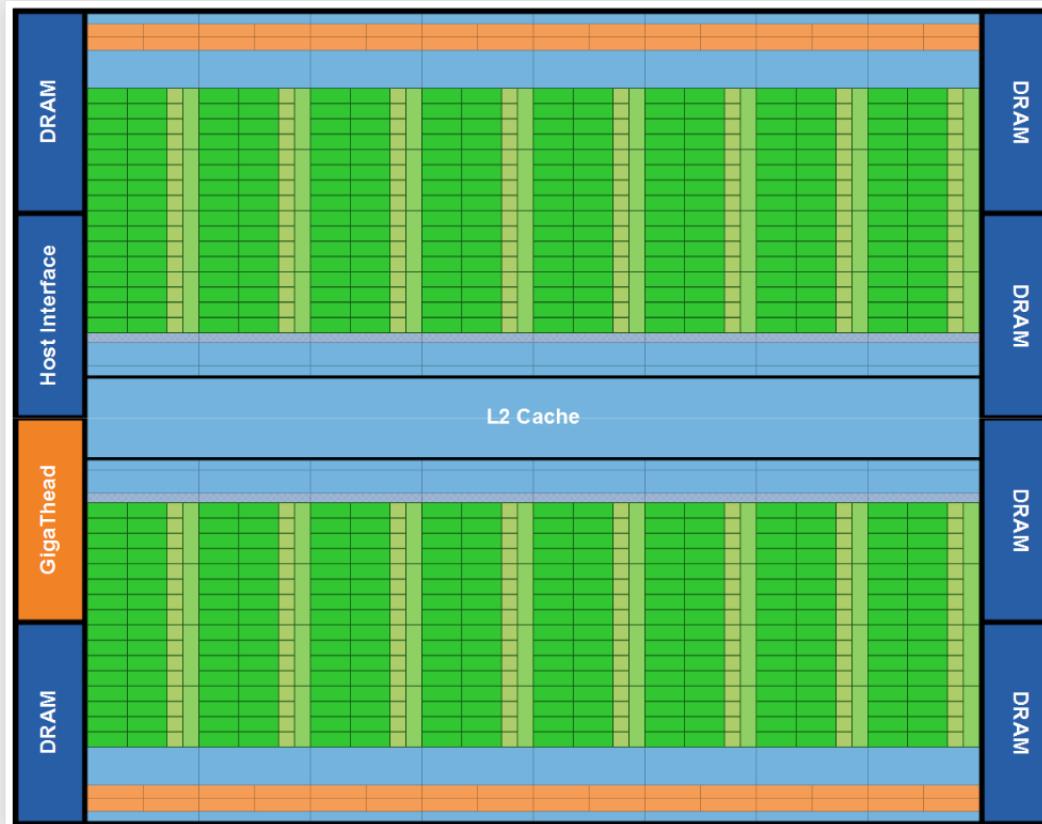
S I S D	S I M D
Single Instruction stream Single Data stream	Single Instruction stream Multiple Data stream
M I S D	M I M D
Multiple Instruction stream Single Data stream	Multiple Instruction stream Multiple Data stream

费林分类 (*Flynn's Taxonomy*,
Michael Flynn, 1966)

图片来源：Lawrence Livermore National Laboratory, <https://hpc.llnl.gov/tutorials/introduction-parallel-computing/flynns-classical-taxonomy>

领域专用计算芯片 (1)

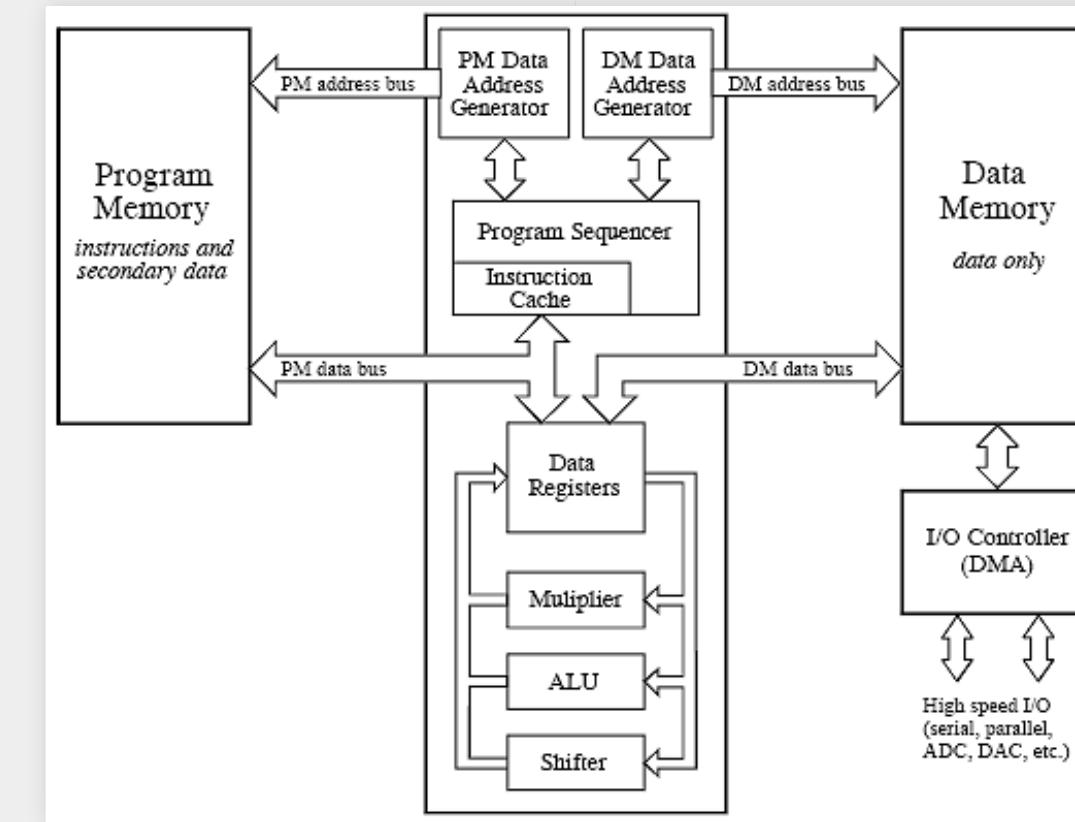
图形处理器 (GPU)



Nvidia, Fermi Architecture

图片来源：Nvidia, NVIDIA's Next Generation CUDA Compute Architecture: Fermi

数字信号处理器 (DSP)

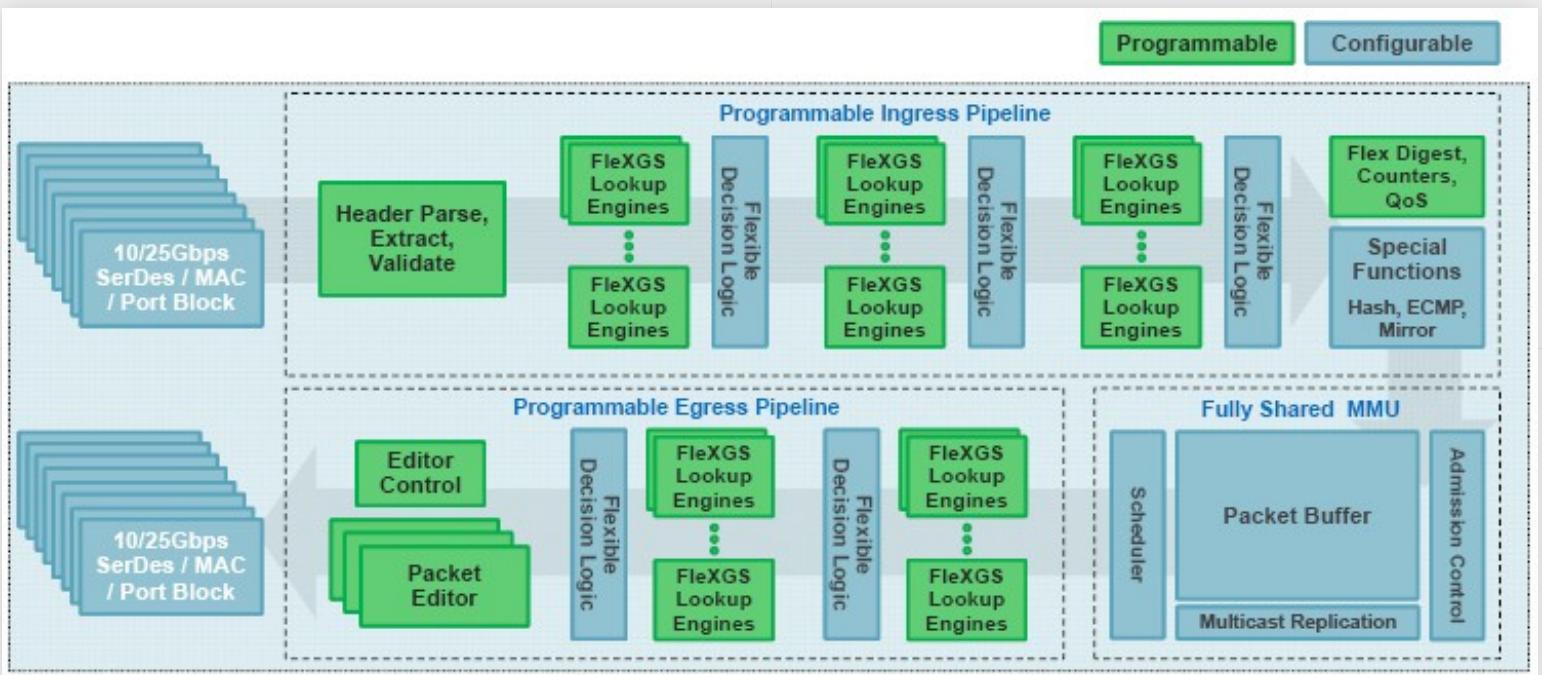


简化的DSP架构

图片来源：http://www.byclb.com/TR/Tutorials/dsp_appl_spc/ch3_1.htm

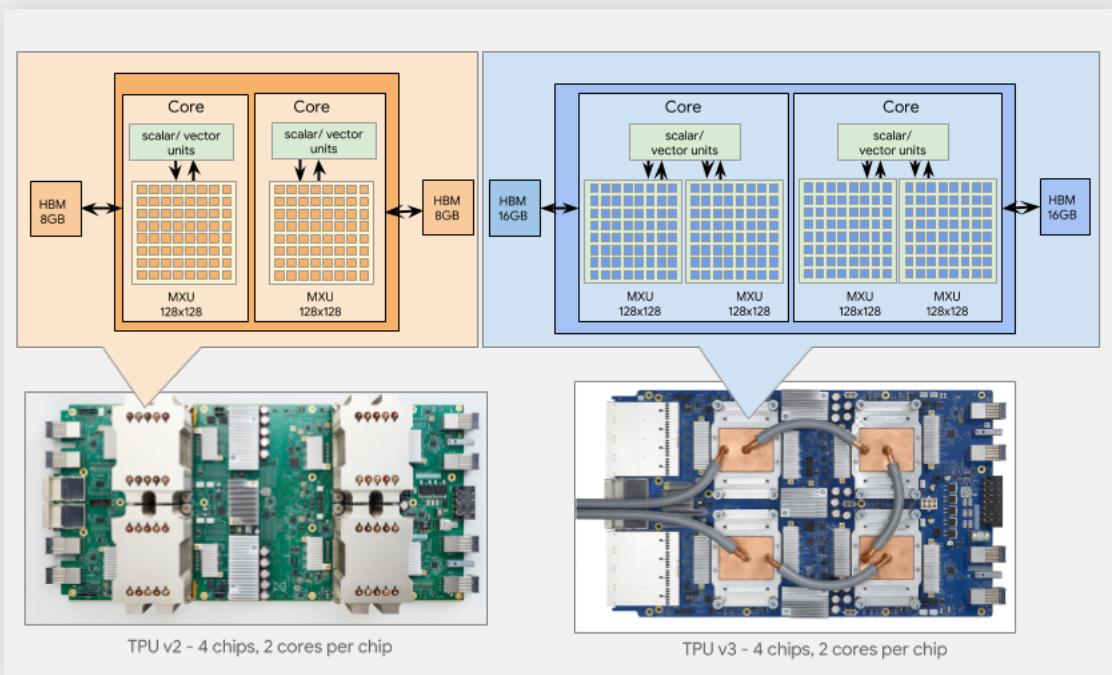
领域专用计算芯片 (2)

网络处理器 (NPU)



Broadcom, Trident 3

人工智能加速处理器 (APU)



Google TPU

图片来源：<https://www.nextplatform.com/2017/08/04/making-mainstream-ethernet-switches-malleable/>

图片来源：<https://cloud.google.com/tpu/docs/system-architecture>

领域专用计算芯片 (3)

小结：针对一些特定的计算，通用计算机处理器的架构并不是最佳选择（性能、价格）

- GPU
 - 目前，许多GPU主要采用SIMD并行
 - 一个GPU上通常具有上百甚至上千个微处理单元
- DSP
 - DSP通常采用哈佛架构
- NPU
 - 基于数据流，指令相对固定
 - 在存储介质上，主要采用TCAM和SRAM
- APU
 - 基于数据流

并行计算

并行计算允许多个处理单元同时执行一个计算任务。按照计算单元的类型分类如下：

- 同构并行计算 (Homogeneous Parallel Computing)
 - 各计算单元是相同的体系结构
 - ALU级别 (SIMD) , 核级别 (多核计算) , 处理器级别 (多处理器计算) , 机器级别 (集群计算等)
- 异构并行计算 (Heterogeneous Parallel Computing)
 - 计算单元分成不同类型 (通常是CPU和其它领域专用芯片)
 - CPU卸载 (Offloading) 利用异构并行计算将特定的计算子任务分配给专用芯片执行

同构并行计算

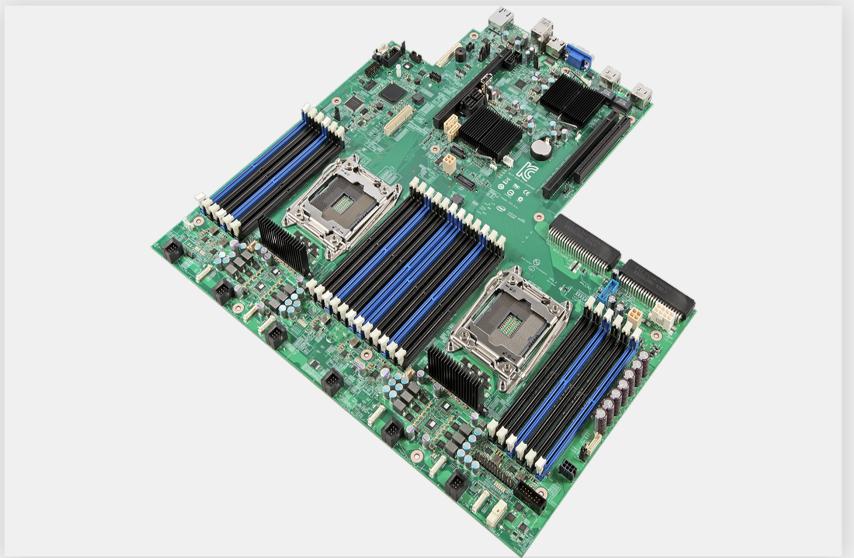
多核 (MULTI-CORE)



Intel Core 2 Quad Q6600

- 一个处理器中有多个核
- 所有的核具有独立的ALU和寄存器组，但是共享内存和其它一些资源

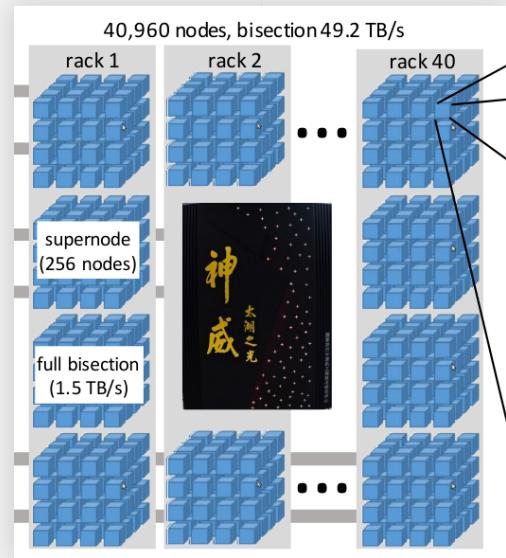
多处理器 (MULTI-PROCESSOR)



Intel s2600wt2 主板

- 一个机器上装有多个独立的处理器
- 处理器之间既可能共享内存，也可能拥有独立的内存

集群处理 (CLUSTER)



神威太湖之光

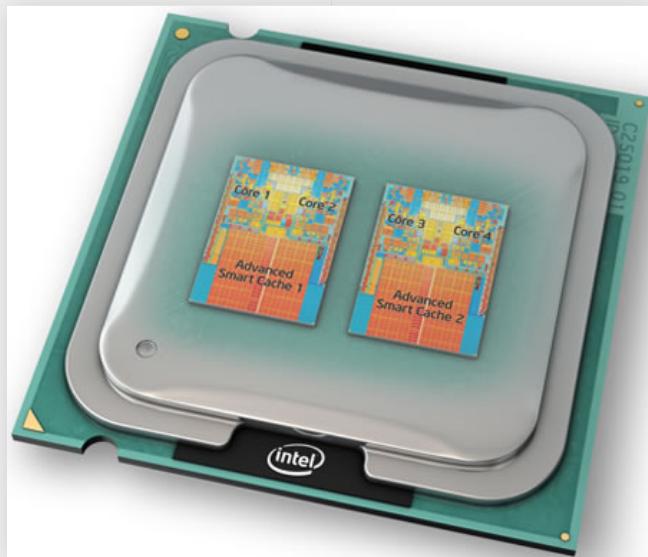
- 一个计算集群通过多个计算机构成
- 机器之间通过高速网络连接

图片来源：

1. <http://www.tech-faq.com/wp-content/uploads/images/Quad-Core-Processor.jpg>
2. <https://servermarketinglibrary.intel.com/wp-content/uploads/assets/s2600wt2-angle.png>
3. http://pacman.cs.tsinghua.edu.cn/~cwg/papers_cwg/a56-lin.pdf

同构并行计算

多核 (MULTI-CORE)



Intel Core 2 Quad Q6600

- 一个处理器中有多个核
- 所有的核具有独立的ALU和寄存器组，但是共享内存和其它一些资源

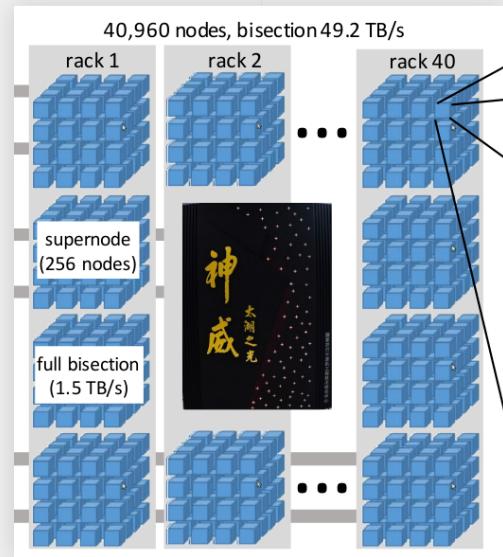
多处理器 (MULTI-PROCESSOR)



Intel s2600wt2 主板

- 一个机器上装有多个独立的处理器
- 处理器之间既可能共享内存，也可能拥有独立的内存

集群处理 (CLUSTER)



神威太湖之光

- 一个计算集群通过多个计算机构成
- 机器之间通过高速网络连接

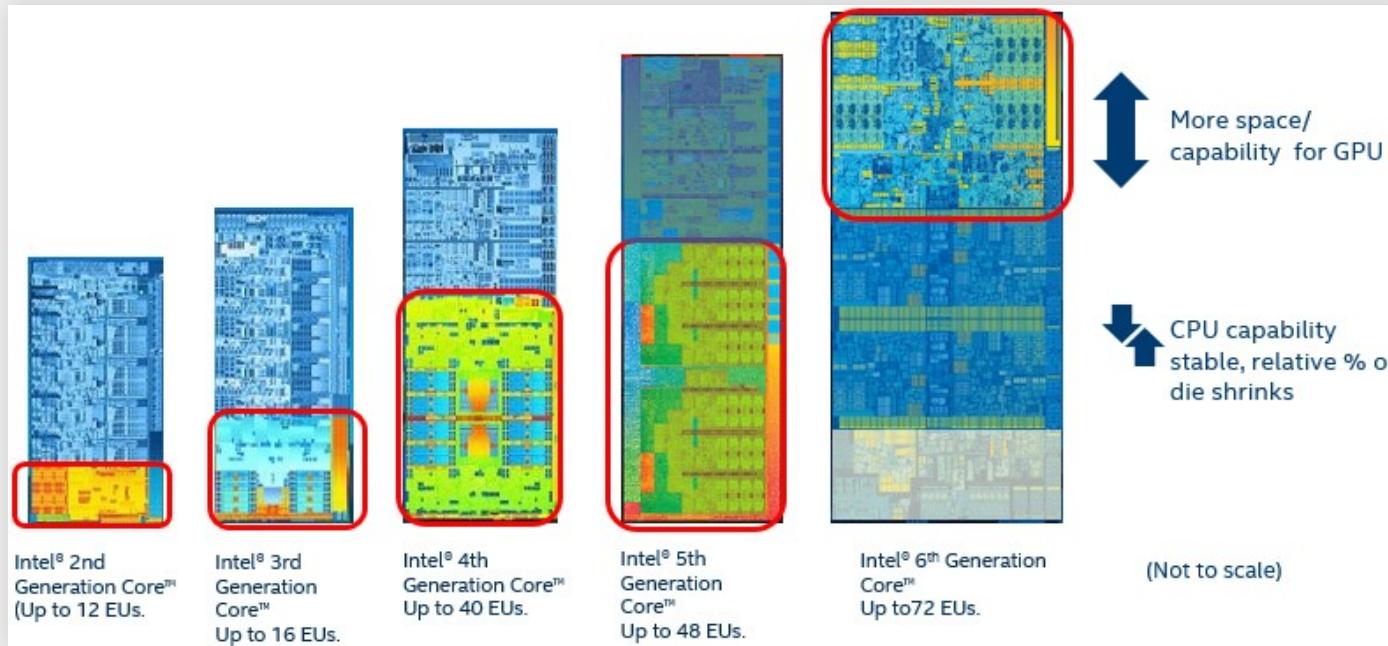
多线程 (Multi-threading) 、多进程 (Multi-process) 请选修操作系统

图片来源：

1. <http://www.tech-faq.com/wp-content/uploads/images/Quad-Core-Processor.jpg>
2. <https://servermarketinglibrary.intel.com/wp-content/uploads/assets/s2600wt2-angle.png>
3. http://pacman.cs.tsinghua.edu.cn/~cwg/papers_cwg/a56-lin.pdf

异构并行计算

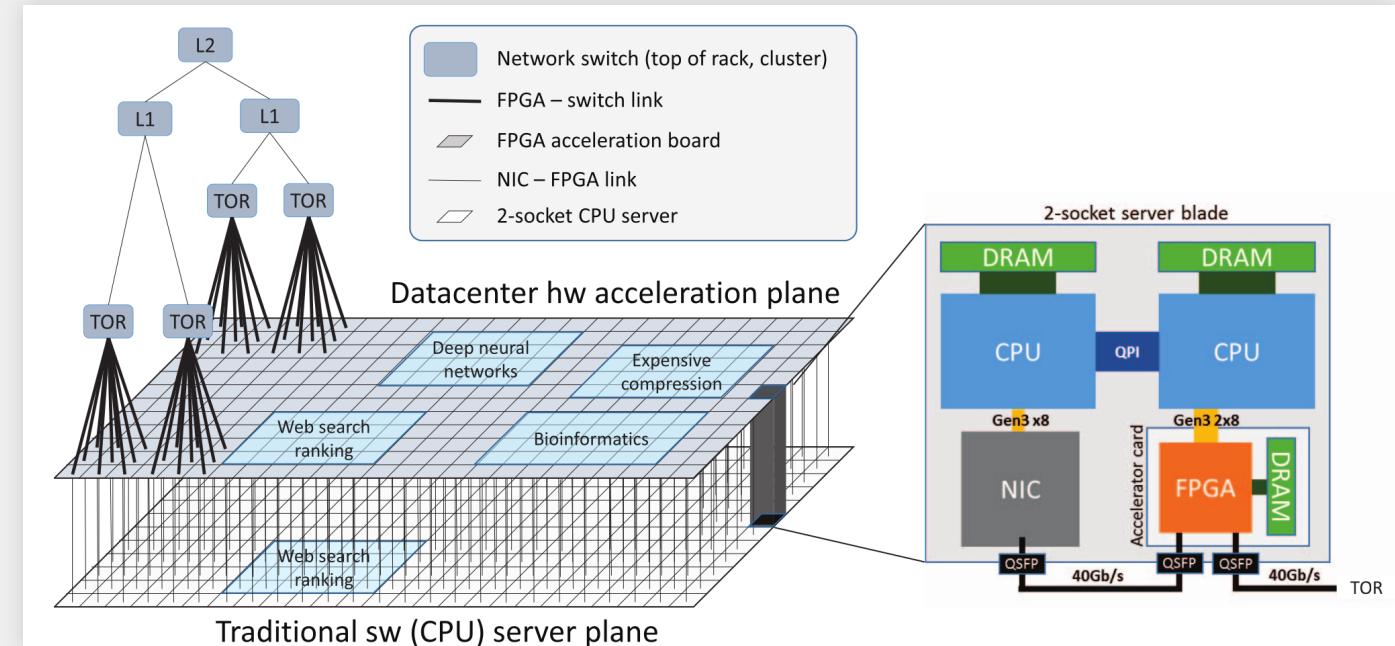
GPU



Intel GPU Offloading

将图形计算任务从CPU卸载到GPU上

FPGA



Microsoft Azure FPGA Cloud

将数据分析任务、虚拟网络管理等任务卸载到FPGA上

图片来源：

1. https://software.intel.com/sites/default/files/managed/f7/f8/cpugpu_0.jpg
2. <https://www.usenix.org/conference/nsdi18/presentation/firestone>



数字电子技术回顾1: 数据表示

按位记数法和进制转换

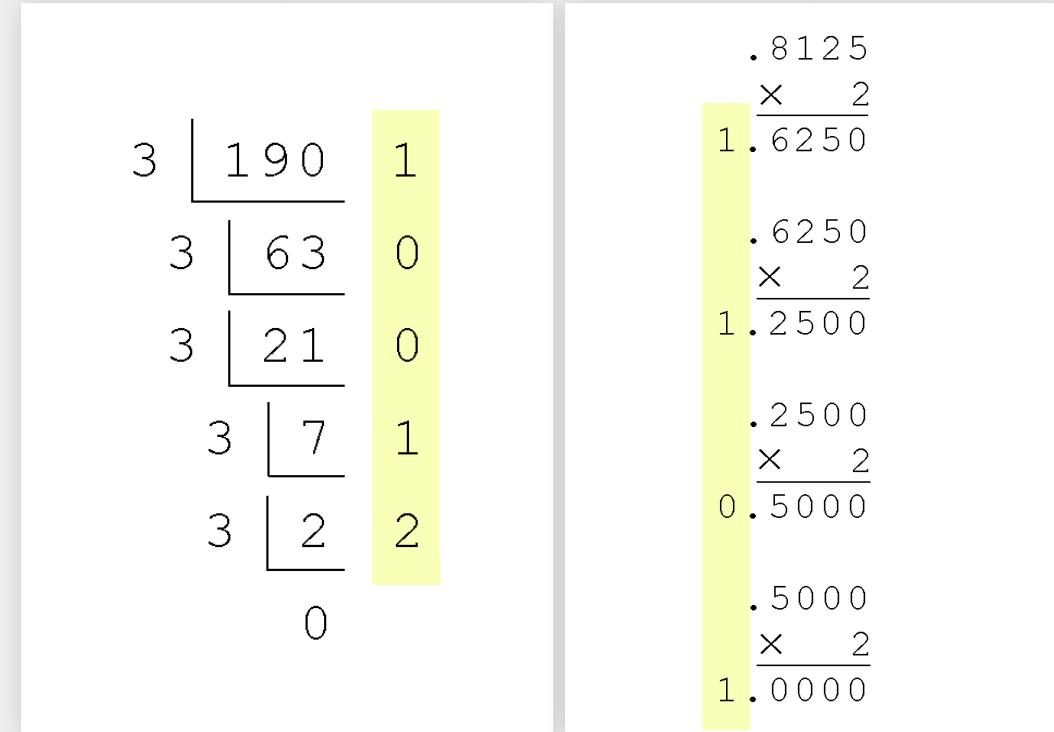
- 按位计数法：将数字用基数（base）的幂加权求和形式表示

- 用 x_b 的形式表示以整数 b 为基的数字
- x 是一个数字序列 $x_U, \dots, x_0, x_{-1}, \dots, x_{-L}$ ，其中的每一位取值为 $[0, b)$ 中的整数

- $x_b = \sum_{i=-L}^U x_i b^i$

- 计算机系统中常用的基数

- 二进制、十六进制（A-F分别代表10-15）
- 例： $100_{10} = (01100100)_2 = 64_{16}$, $(64)_{16}$ 也记作 $0x64$



$$192_{10} = 10012_3, 0.8125_{10} = 0.1101_2$$

计算机系统中的数字表示和算术运算

计算机系统的特点：数字表示的位数（字长）是有限的

计算机系统中的数字表示和算术运算

计算机系统的特点：数字表示的位数（字长）是有限的

- 无符号整数：一般采用对应的二进制表示

以字长等于8为例

$$134_{10} = (01000100)_2$$

计算机系统中的数字表示和算术运算

计算机系统的特点：数字表示的位数（字长）是有限的

- 无符号整数：一般采用对应的二进制表示
- 有符号整数：原码、补码和移码以及对应的算术运算

以字长等于8为例

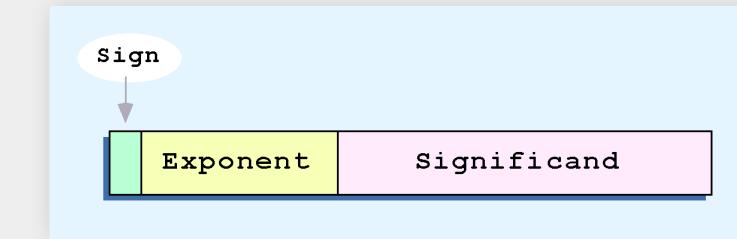
- 原码 (signed magnitude)：
 $-121_{10} = -|121_{10}| = (1|1111001)_2$
- 反码 (diminished radix complement)：
 $121_{10} \Rightarrow (2^7 - 1) - 121 = 6_{10} = (0000110)_2$
- 一的补码 (One's complement)
 - 非负数用原码： $121_{10} = (0|1111001)_2$
 - 负数用反码： $-121_{10} = (1|0000110)_2$
- 二的补码 (Two's complement)
 - 非负数用原码： $121_{10} = (0|1111001)_2$
 - 负数用反码加1： $-121_{10} = (1|0000111)_2$
- 偏移值为M的移码 (Excess-M)：假设M=127，则
 $121_{10} \Rightarrow (121 + 127) = -120_{10} = (1|1111000)_2$

计算机系统中的数字表示和算术运算

计算机系统的特点：数字表示的位数（字长）是有限的

- 无符号整数：一般采用对应的二进制表示
- 有符号整数：原码、补码和移码以及对应的算术运算
- 浮点数：浮点数的表示和算术运算

以字长等于14为例



图片来源：Linda Null, Julia Lobur, *The Essentials of Computer Organization and Architecture*, 4t

- 假设1个符号位，5个指数位，8个有效数位
- 指数位采用移码：消除指数位中的符号位
- 有效数位采用规格化（Normalization）：最左边默认有一个值为1的隐藏位
- 例 $19.5_{10} = (10011.1)_2 = (1.00111) * 2^4$
 - 符号位为0
 - 指数位是M=15的移码： $4_{10} \Rightarrow (4 + 15) = 19_{10} = (10011)_2$
 - 有效数位是去掉1之后的小数位：00111000
 - 最终表示：(0|10011|00111000)

数字电子技术回顾2: 布尔代数到数字电路

布尔代数及基本门电路

布尔代数及基本门电路

- 基本布尔逻辑运算符：AND/OR/NOT

X AND Y		
X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y		
X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X	
X	X'
0	1
1	0

图片来源：Linda Null, Julia Lobur, *The Essentials of Computer Organization and Architecture*, 4th Edition

布尔代数及基本门电路

- 基本布尔逻辑运算符：AND/OR/NOT
- 布尔表达式和布尔函数
 - 布尔积表达式（对应AND）： $x y$
 - 布尔和表达式（对应OR）： $x + y$
 - 布尔非表达式（对应NOT）： \bar{x} 或者 x'
 - 运算符优先级：NOT > AND > OR
 - 布尔函数：由布尔变量和逻辑运算符组成的表达式

■ 例：

			$F(x, y, z) = xz' + y$		
x	y	z	z'	xz'	$xz' + y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

图片来源：Linda Null, Julia Lobur, *The Essentials of Computer Organization and Architecture*, 4th Edition

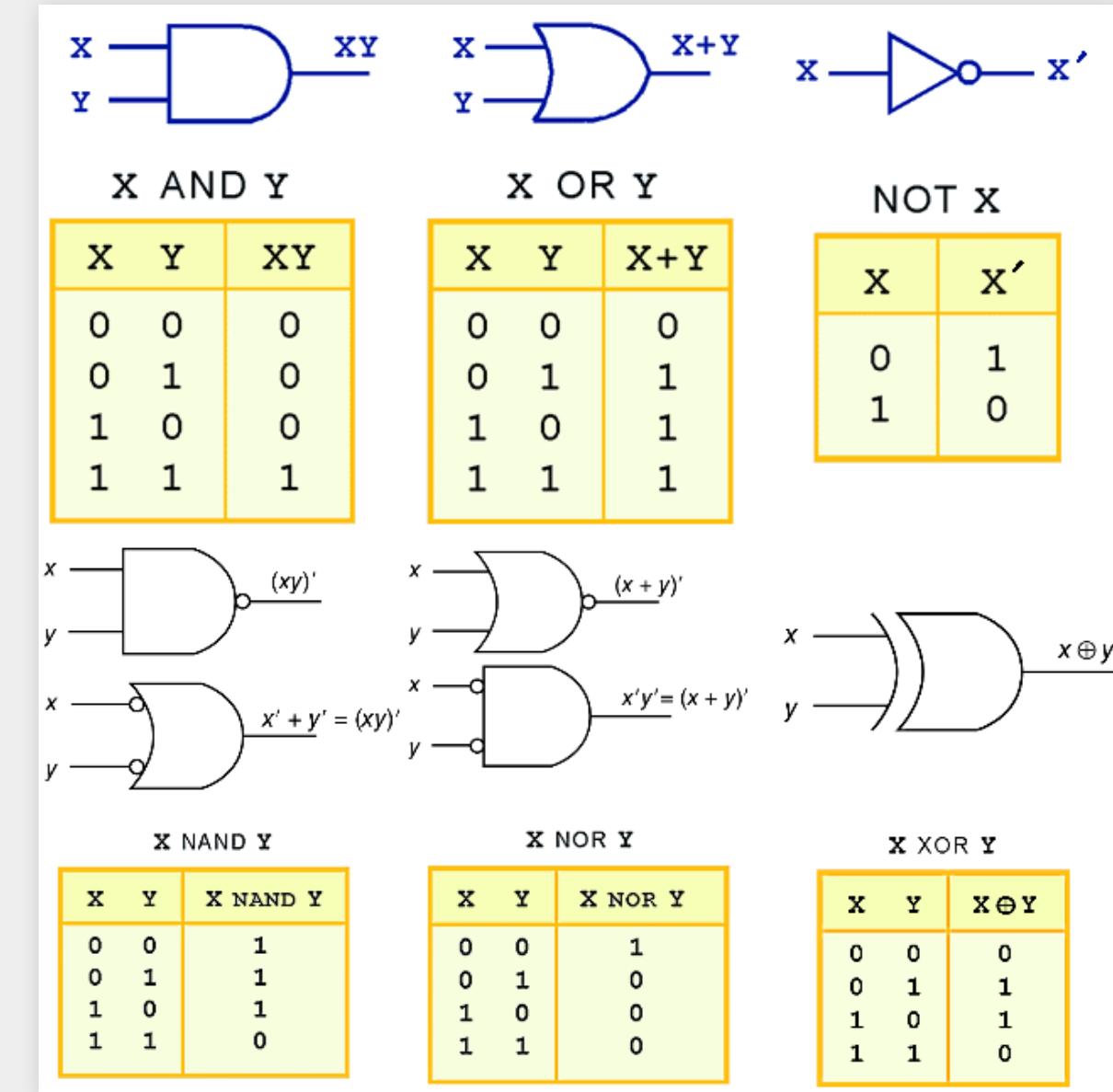
布尔代数及基本门电路

- 基本布尔逻辑运算符：AND/OR/NOT
- 布尔表达式和布尔函数
- 布尔代数基本定律

定律名称	AND	OR
同一律 Identify Law	$1x = x$	$0 + x = x$
零律 Null Law	$0x = 0$	$1 + x = 1$
幂等律 Idempotent Law	$xx = x$	$x + x = x$
逆等律 Inverse Law	$xx' = 0$	$x + x' = 1$
交换律 Commutative Law	$xy = yx$	$x + y = y + x$
结合律 Associative Law	$(xy)z = x(yz)$	$(x + y) + z = x + (y + z)$
分配律 Distributive Law	$x + (yz) = (x + y)(x + z)$	$x(y + z) = xy + xz$
吸收律 Absorption Law	$x(x + y) = x$	$x + xy = x$
德摩根定律 DeMorgan's Law	$(xy)' = x' + y'$	$(x + y)' = x'y'$
双重否定律 Double Complement Law	$x'' = x$	

布尔代数及基本门电路

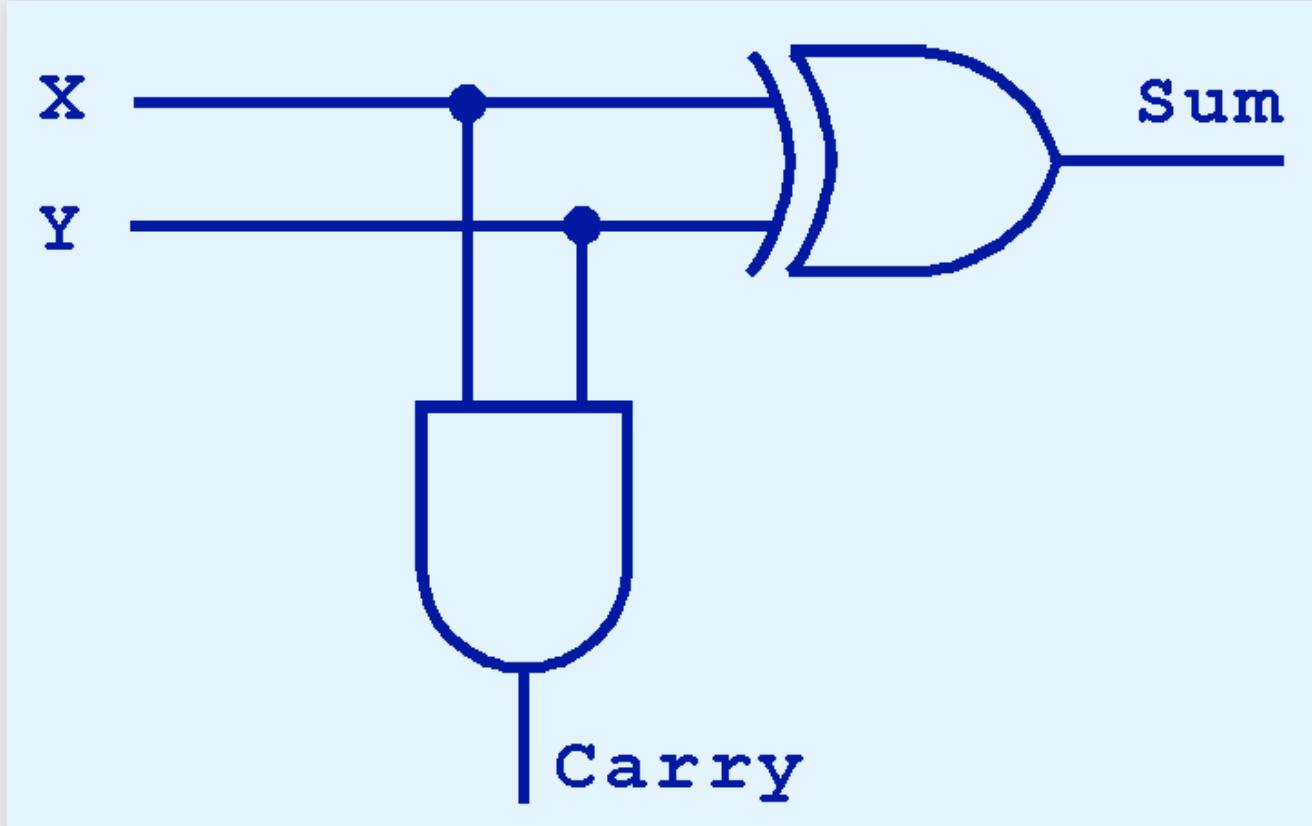
- 基本布尔逻辑运算符：AND/OR/NOT
- 布尔表达式和布尔函数
- 布尔代数基本定律
- 布尔代数运算符对应的门电路及常用门电路



常用组合逻辑电路

常用组合逻辑电路

半加器 (Half Adder)

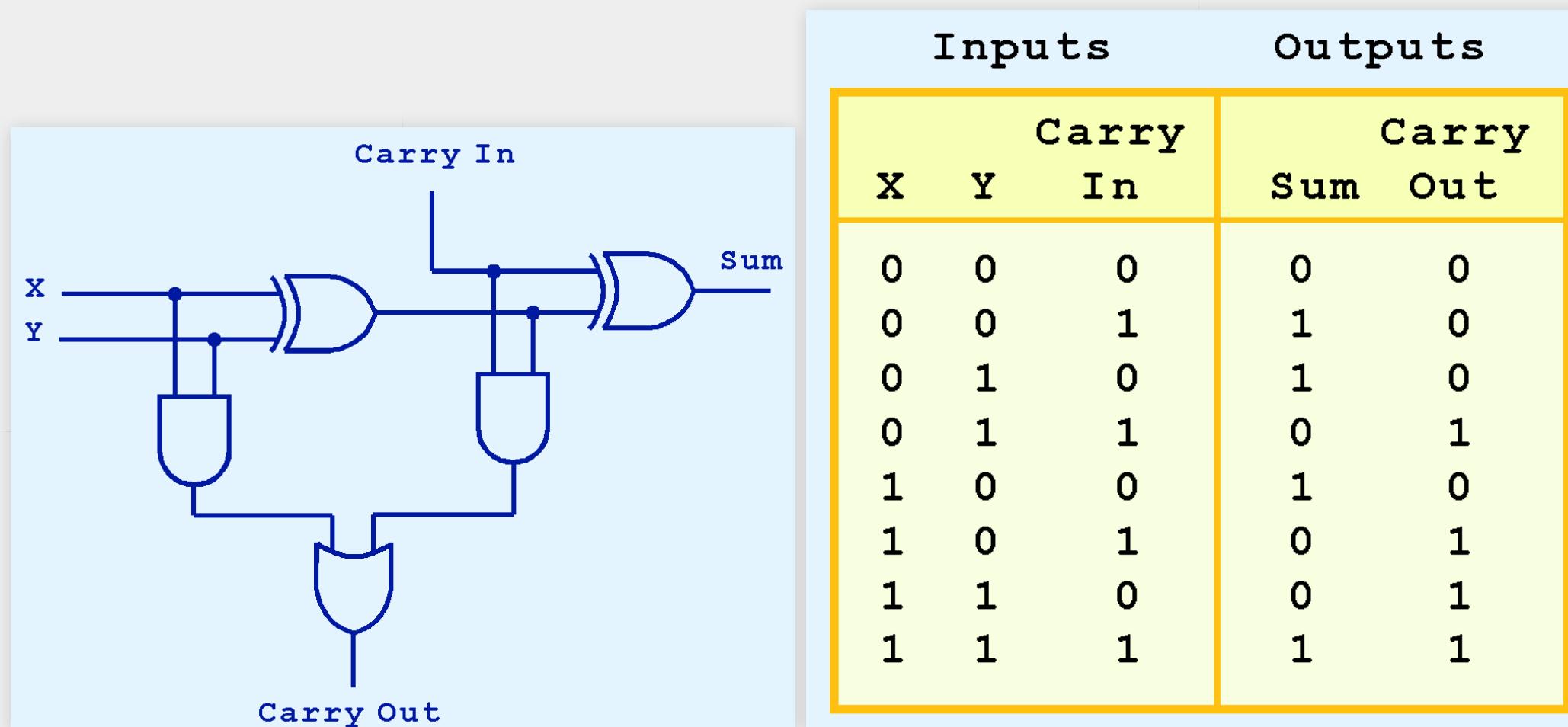


Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

图片来源：Linda Null, Julia Lobur, *The Essentials of Computer Organization and Architecture*, 4th Edition

常用组合逻辑电路

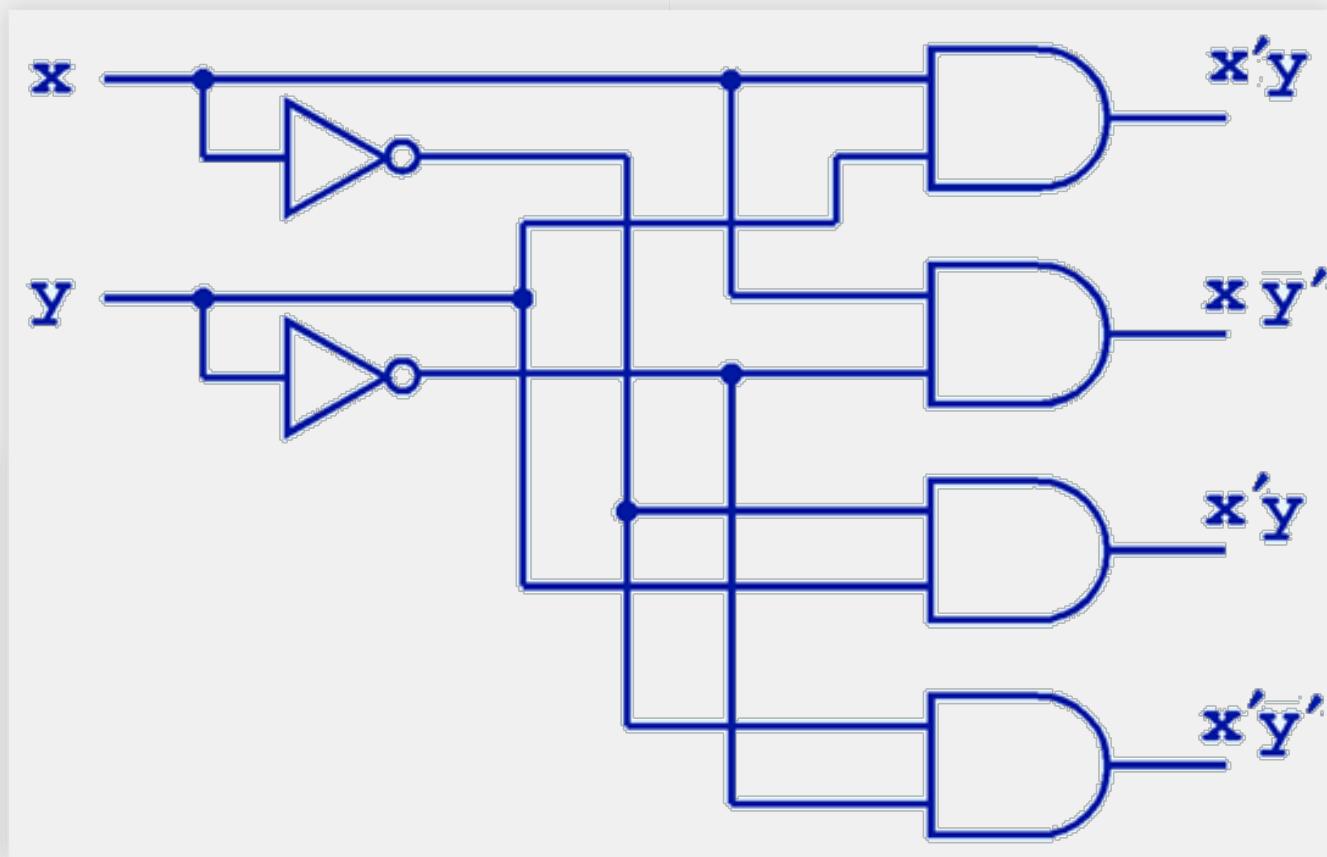
全加器 (Full Adder)



图片来源：Linda Null, Julia Lobur, *The Essentials of Computer Organization and Architecture*, 4th Edition

常用组合逻辑电路

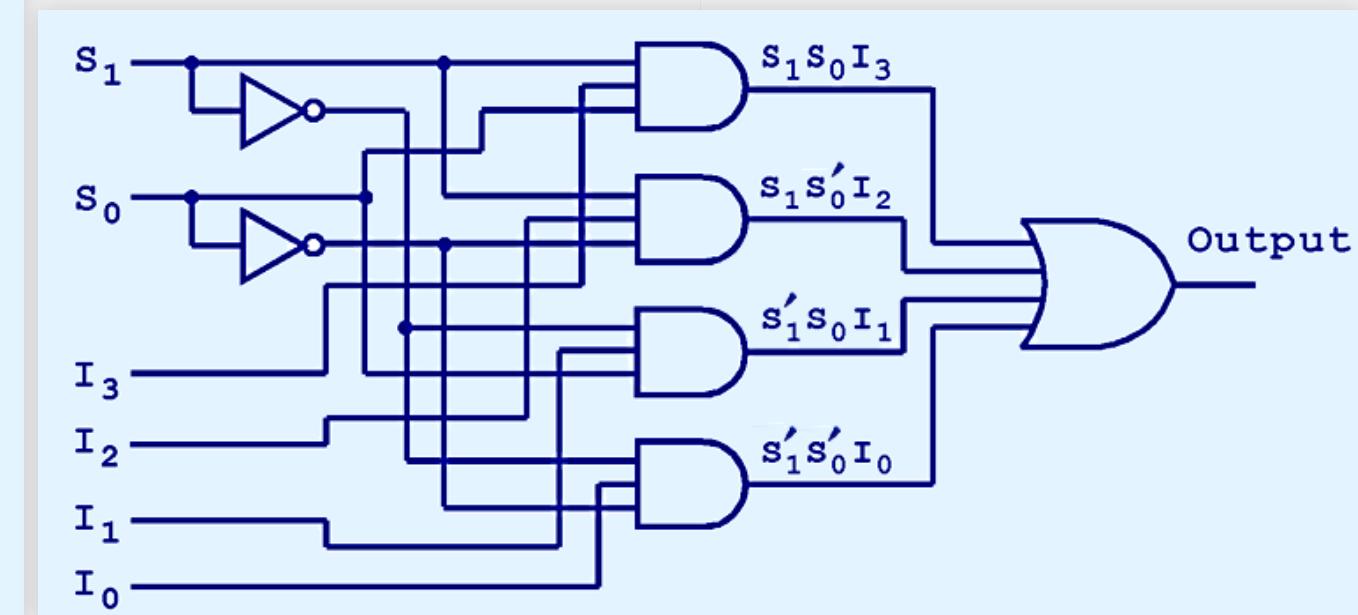
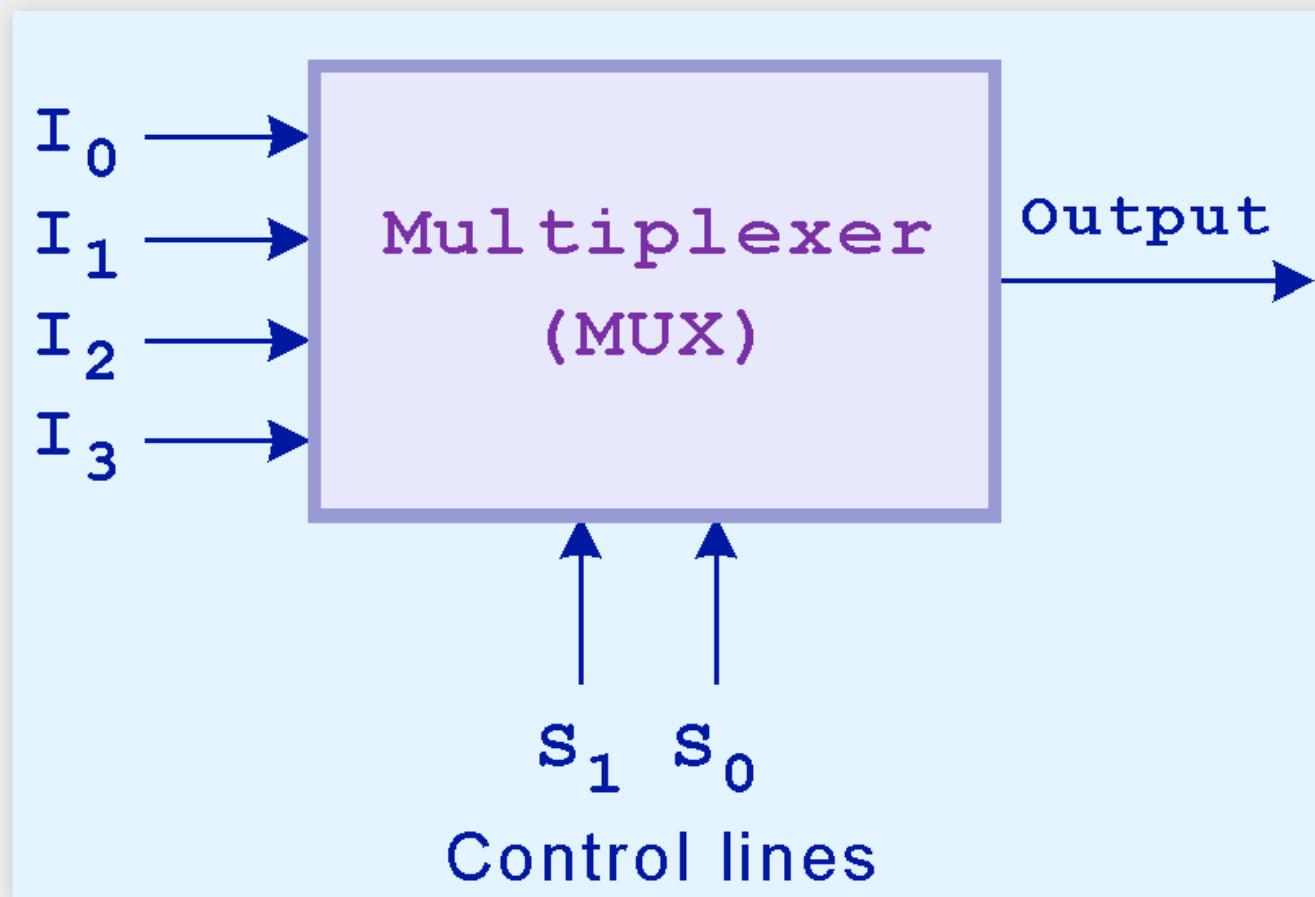
译码器 (Decoder)



图片来源：Linda Null, Julia Lobur, *The Essentials of Computer Organization and Architecture*, 4th Edition

常用组合逻辑电路

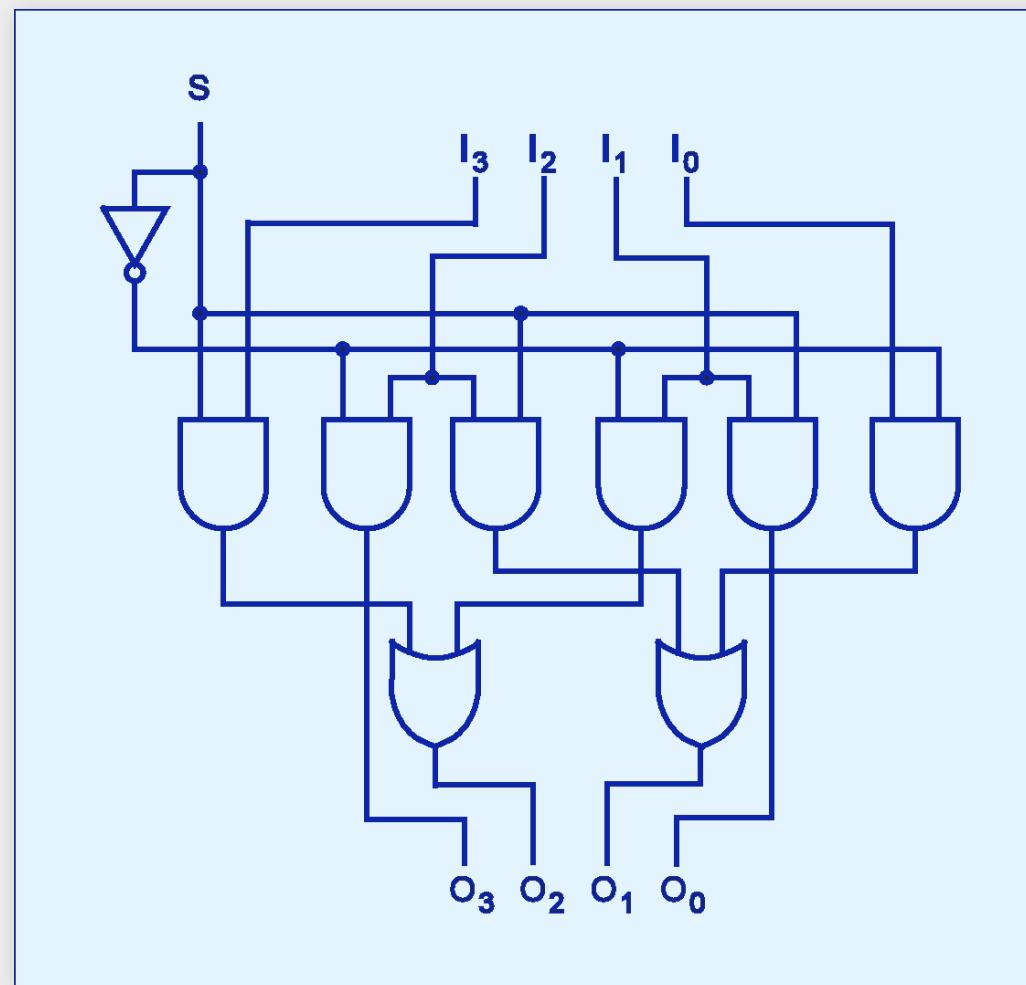
多路复用器 (Multiplexer, MUX)



图片来源：Linda Null, Julia Lobur, *The Essentials of Computer Organization and Architecture*, 4th Edition

常用组合逻辑电路

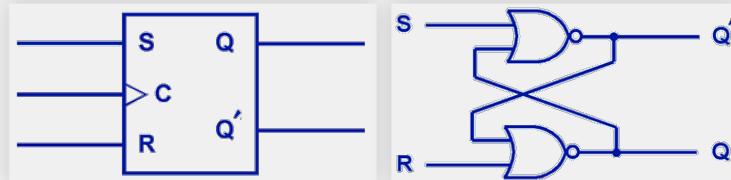
移位器 (Shifter)



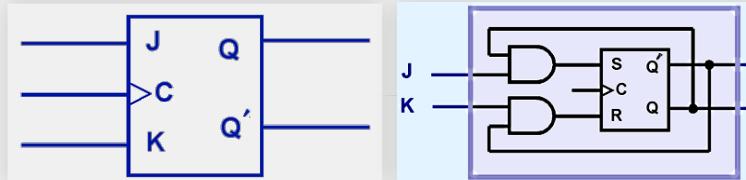
图片来源：Linda Null, Julia Lobur, *The Essentials of Computer Organization and Architecture*, 4th Edition

常用时序逻辑电路

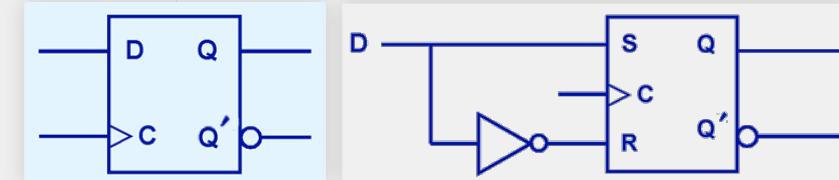
SR触发器



JK触发器



D触发器



Present State			Next State
S	R	$Q(t)$	$Q(t+1)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	undefined
1	1	1	undefined

J	K	$Q(t+1)$
0	0	$Q(t)$ (no change)
0	1	0 (reset to 0)
1	0	1 (set to 1)
1	1	$Q'(t)$

D	$Q(t+1)$
0	0
1	1

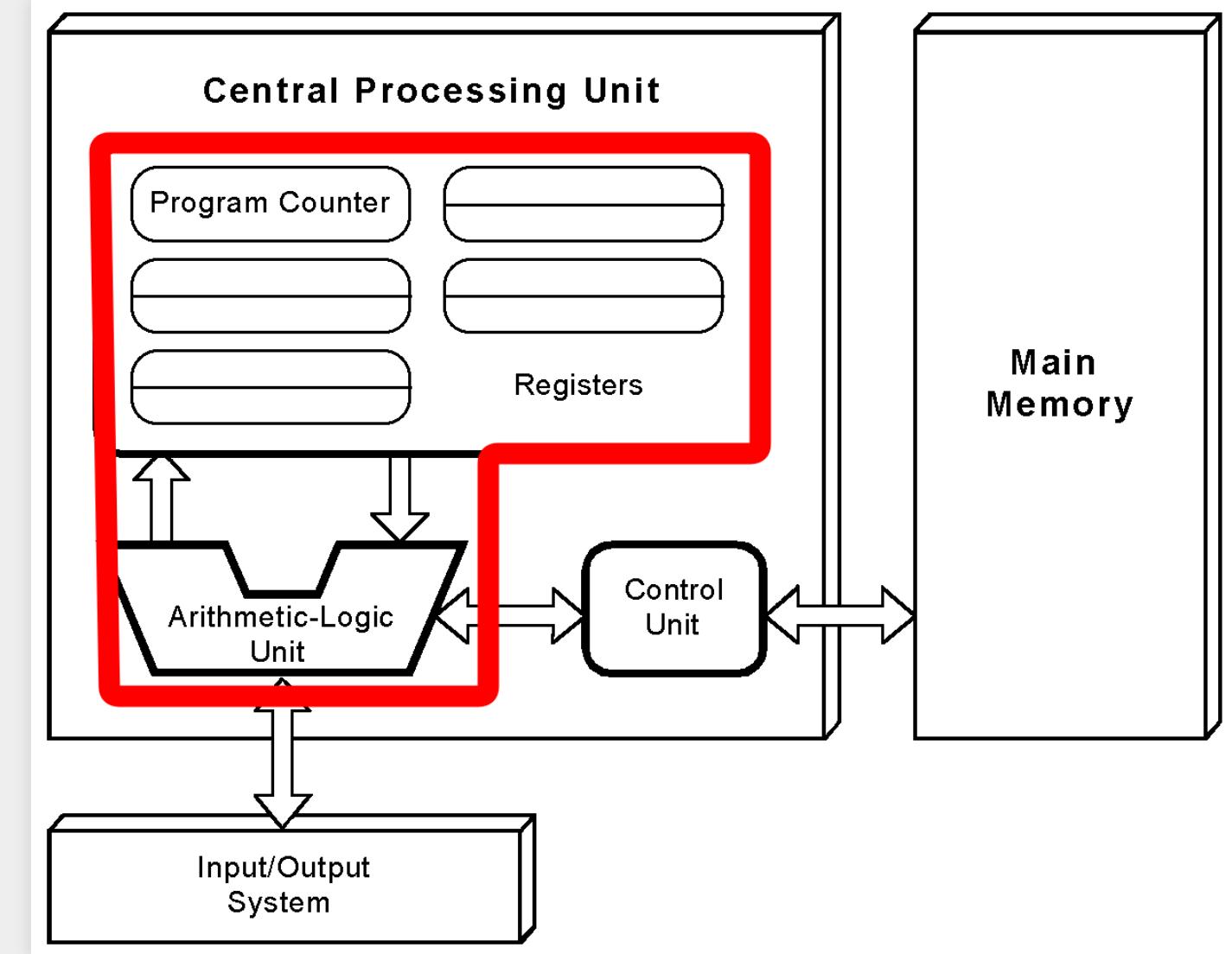
图片来源：Linda Null, Julia Lobur, *The Essentials of Computer Organization and Architecture*, 4th Edition

计算机体系结构

CPU基本知识和组织结构

CPU基本知识和组织结构

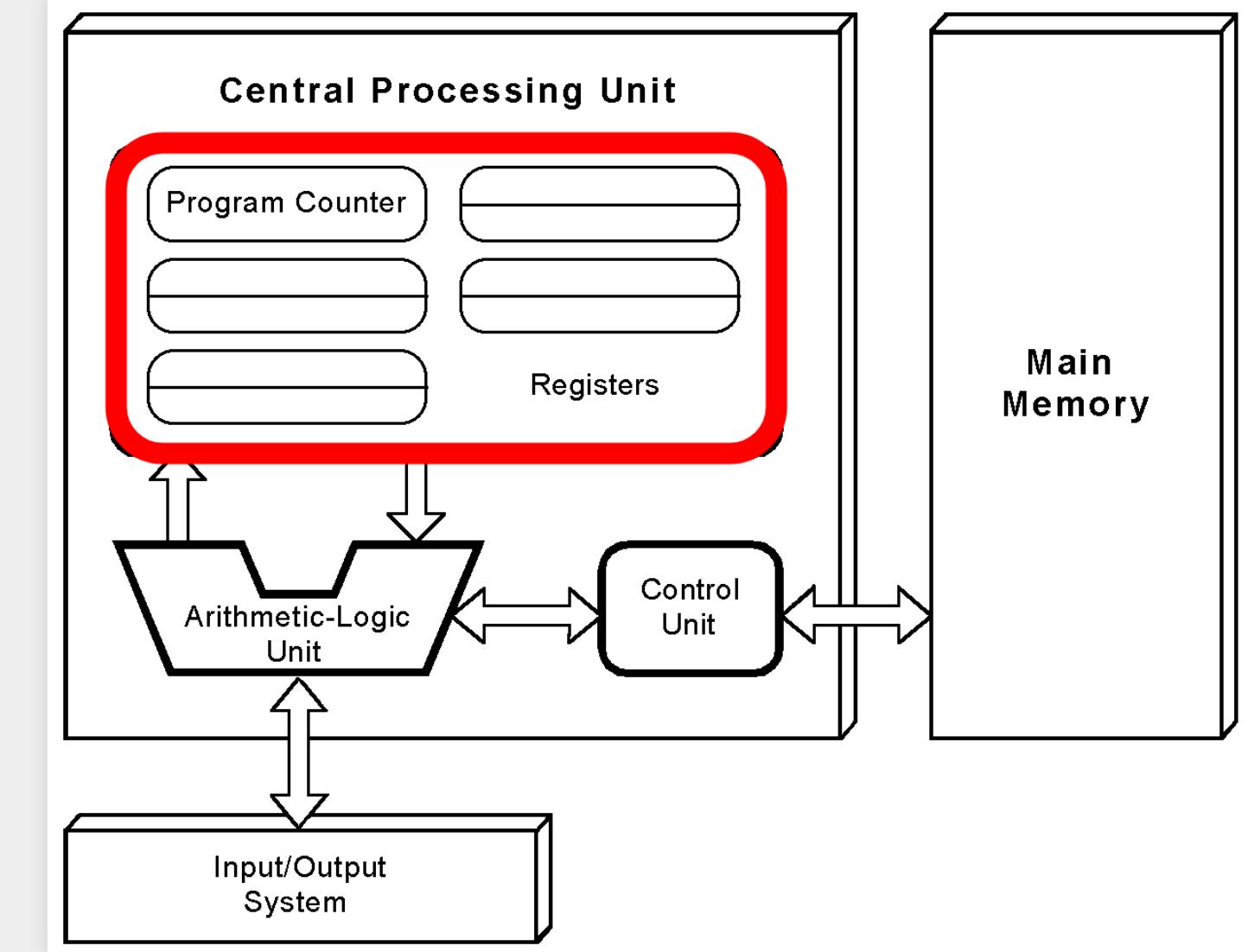
- 数据通路 (Datapath)



图片来源：Linda Null, Julia Lobur, *The Essentials of Computer Organization and Architecture*, 4th Edition

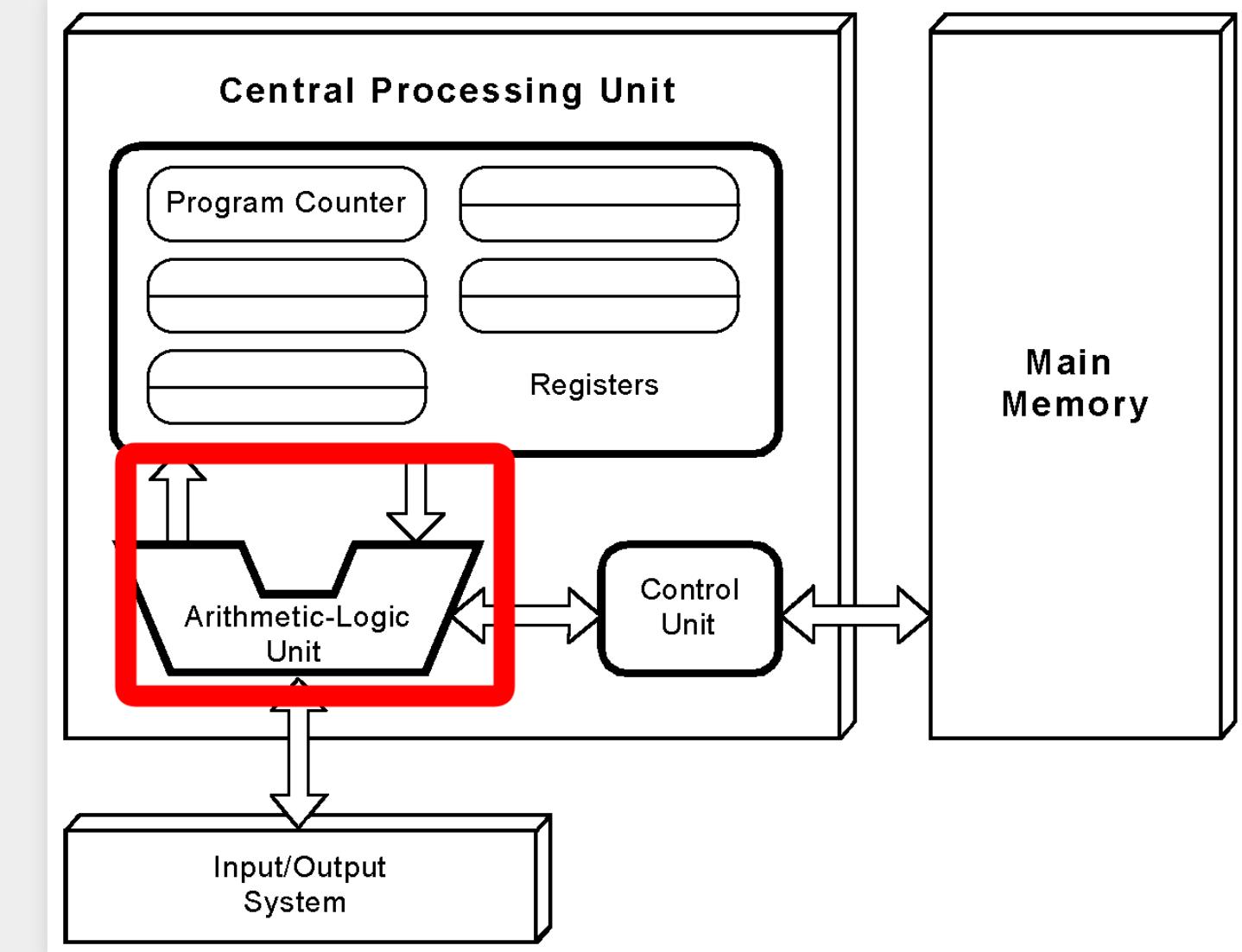
CPU基本知识和组织结构

- 数据通路 (Datapath)
 - 寄存器：CPU内部的数据储存单元



CPU基本知识和组织结构

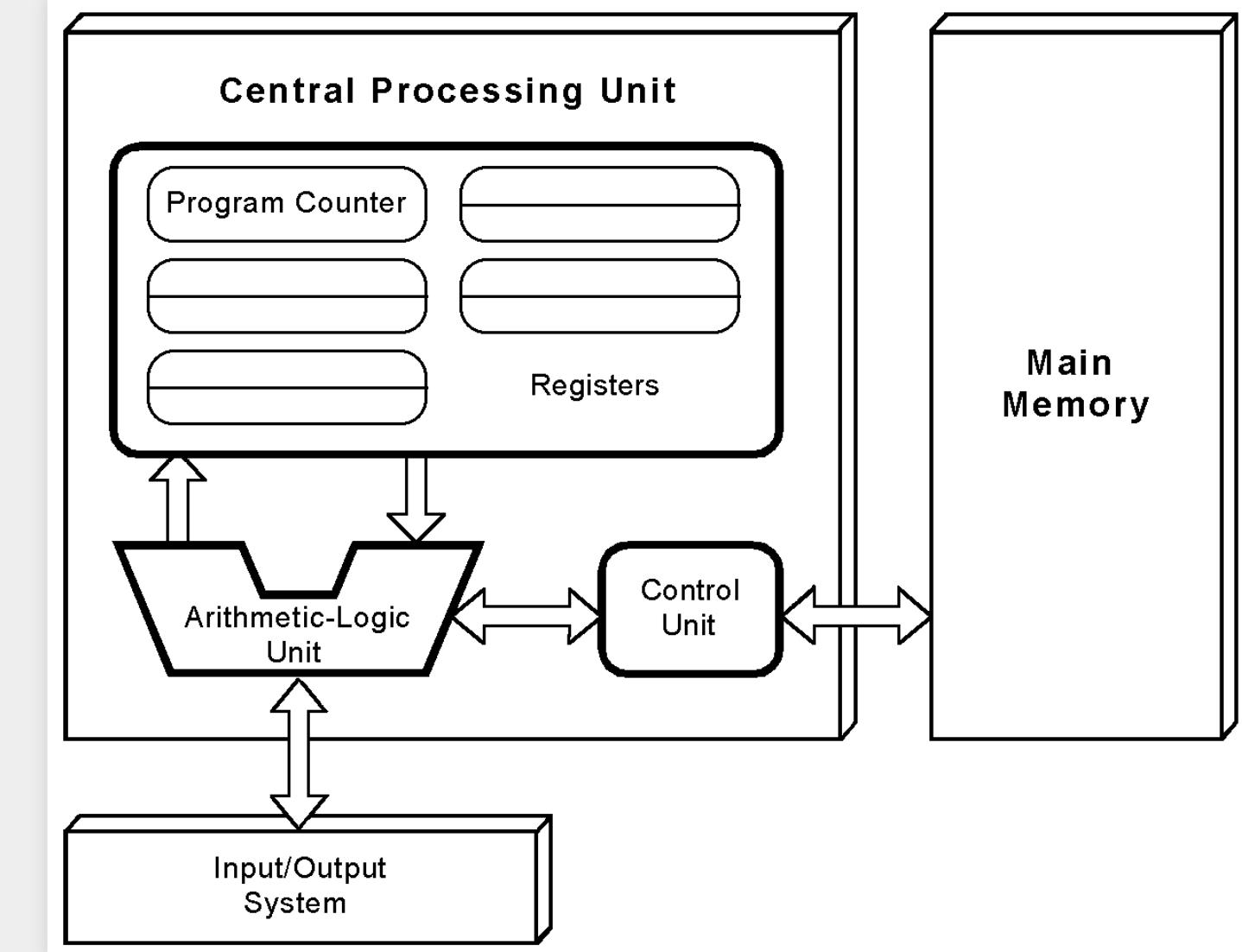
- 数据通路 (Datapath)
 - 寄存器：CPU内部的数据储存单元
 - 算数逻辑单元 (ALU) : 执行实际运算



CPU基本知识和组织结构

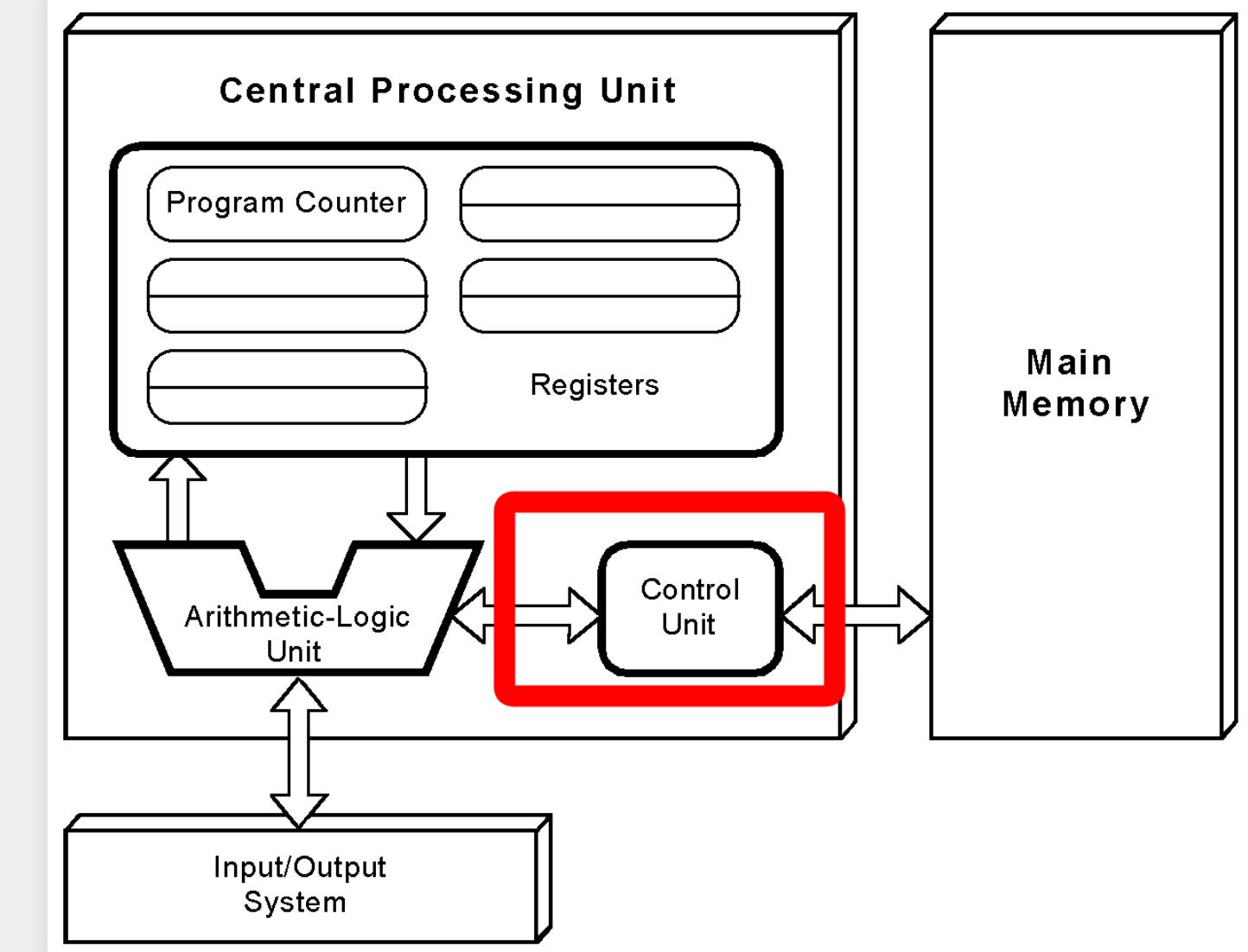
- 数据通路 (Datapath)

- 寄存器：CPU内部的数据储存单元
- 算数逻辑单元 (ALU)：执行实际运算
- 总线：连接CPU内部的存储单元和计算单元、也连接CPU和主存储、I/O设备等



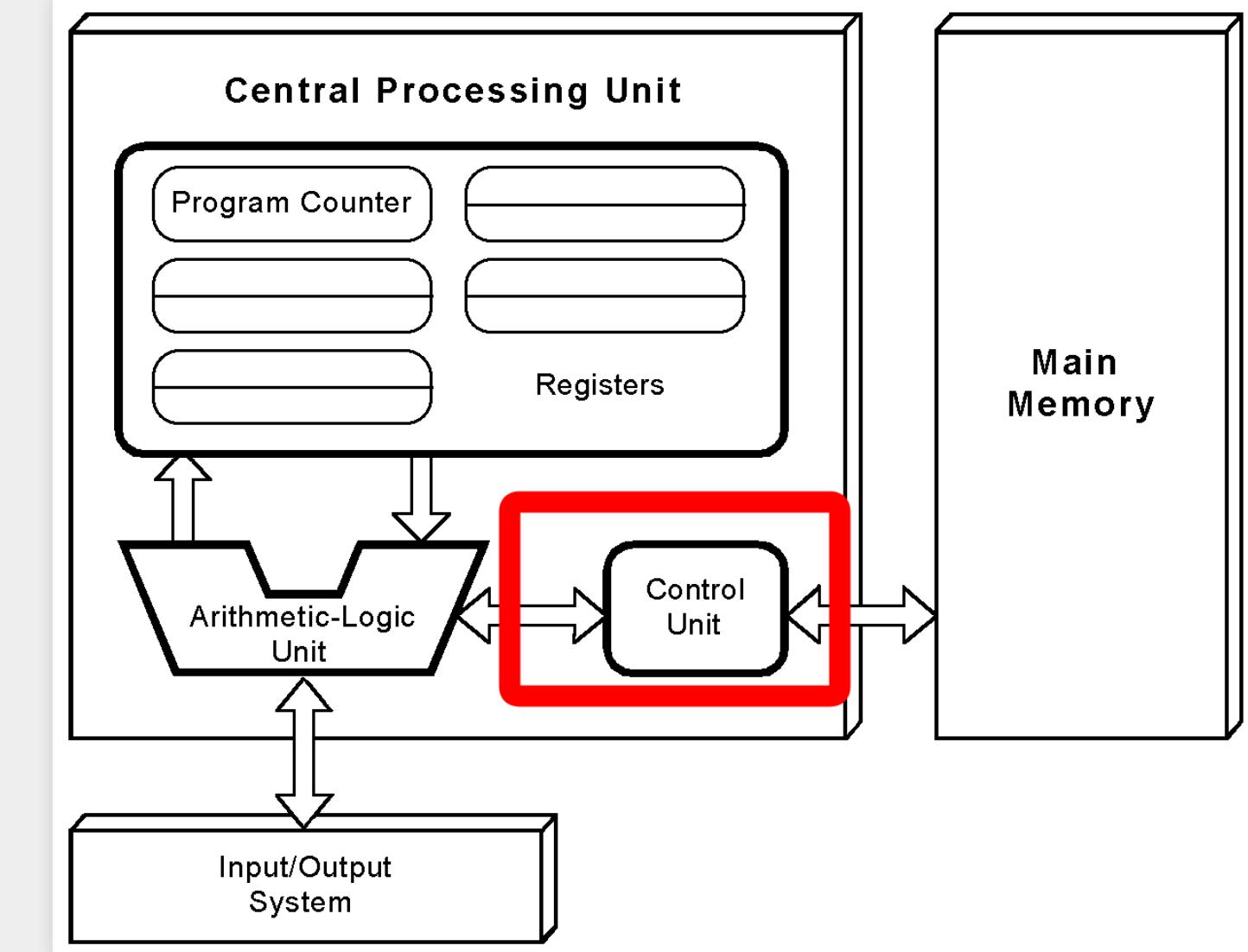
CPU基本知识和组织结构

- 数据通路 (Datapath)
 - 寄存器：CPU内部的数据储存单元
 - 算数逻辑单元 (ALU)：执行实际运算
 - 总线：连接CPU内部的存储单元和计算单元、也连接CPU和主存储、I/O设备等
- 控制单元 (Control Unit)



CPU基本知识和组织结构

- 数据通路 (Datapath)
 - 寄存器：CPU内部的数据储存单元
 - 算数逻辑单元 (ALU)：执行实际运算
 - 总线：连接CPU内部的存储单元和计算单元、也连接CPU和主存储、I/O设备等
- 控制单元 (Control Unit)
 - 监视、调度、控制数据通路中的数据传输



寄存器

寄存器

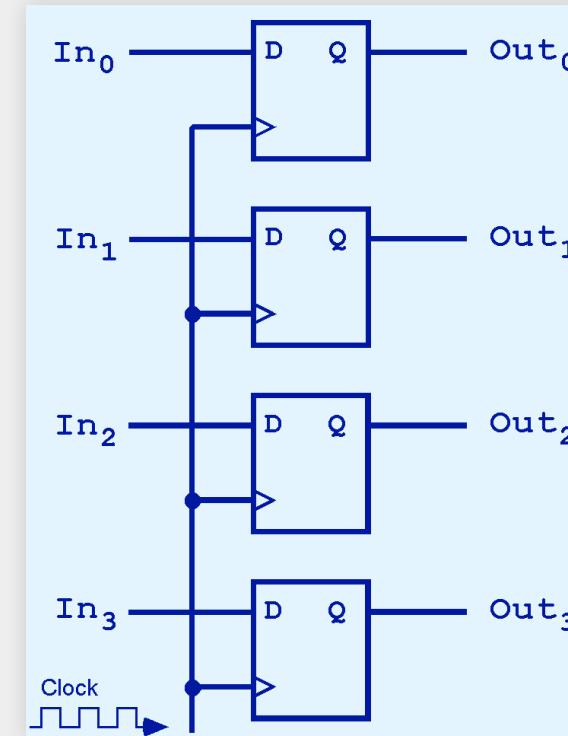
- 通用寄存器可以通过D触发器实现

寄存器

- 通用寄存器可以通过D触发器实现
 - 一个D触发器相当于1-bit的寄存器

寄存器

- 通用寄存器可以通过D触发器实现
 - 一个D触发器相当于1-bit的寄存器
 - D触发器通过并联实现多位寄存器（共享时钟）



4位寄存器的实现

图片来源：Linda Null, Julia Lobur, *The Essentials of Computer Organization and Architecture*, 4th Edit

寄存器

- 通用寄存器可以通过D触发器实现
 - 一个D触发器相当于1-bit的寄存器
 - D触发器通过并联实现多位寄存器（共享时钟）
- 寄存器的大小：现有计算机通常是16/32/64位

64-Bit Mode ¹		
Name	Number	Size (bits)
RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8–R15	16	64
YMM0–YMM15 ³	16	256
XMM0–XMM15 ³	16	128
MMX0–MMX7 ⁴	8	64
FPR0–FPR7 ⁴	8	80
RIP	1	64
RFLAGS	1	64
—		64

AMD64 架构64位模式下寄存器集

图片来源：*AMD64 Architecture Programmer's Manual*, <https://www.amd.com/system/files/TechDocs/>

寄存器

- 通用寄存器可以通过D触发器实现
 - 一个D触发器相当于1-bit的寄存器
 - D触发器通过并联实现多位寄存器（共享时钟）
- 寄存器的大小：现有计算机通常是16/32/64位
- 寄存器的数量：通常是2的倍数，常见的数量是16/32/64

64-Bit Mode ¹		
Name	Number	Size (bits)
RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8–R15	16	64
YMM0–YMM15 ³	16	256
XMM0–XMM15 ³	16	128
MMX0–MMX7 ⁴	8	64
FPR0–FPR7 ⁴	8	80
RIP	1	64
RFLAGS	1	64
—		64

AMD64 架构64位模式下寄存器集

图片来源：*AMD64 Architecture Programmer's Manual*, <https://www.amd.com/system/files/TechDocs/>

寄存器

- 通用寄存器可以通过D触发器实现
 - 一个D触发器相当于1-bit的寄存器
 - D触发器通过并联实现多位寄存器（共享时钟）
- 寄存器的大小：现有计算机通常是16/32/64位
- 寄存器的数量：通常是2的倍数，常见的数量是16/32/64
- 寄存器的内容：数据、地址和控制信息

64-Bit Mode ¹		
Name	Number	Size (bits)
RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8–R15	16	64
YMM0–YMM15 ³	16	256
XMM0–XMM15 ³	16	128
MMX0–MMX7 ⁴	8	64
FPR0–FPR7 ⁴	8	80
RIP	1	64
RFLAGS	1	64
—		64

AMD64 架构64位模式下寄存器集

图片来源：*AMD64 Architecture Programmer's Manual*, <https://www.amd.com/system/files/TechDocs/>

寄存器

- 通用寄存器可以通过D触发器实现
 - 一个D触发器相当于1-bit的寄存器
 - D触发器通过并联实现多位寄存器（共享时钟）
- 寄存器的大小：现有计算机通常是16/32/64位
- 寄存器的数量：通常是2的倍数，常见的数量是16/32/64
- 寄存器的内容：数据、地址和控制信息
- 专用寄存器

64-Bit Mode ¹		
Name	Number	Size (bits)
RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8–R15	16	64
YMM0–YMM15 ³	16	256
XMM0–XMM15 ³	16	128
MMX0–MMX7 ⁴	8	64
FPR0–FPR7 ⁴	8	80
RIP	1	64
RFLAGS	1	64
—		64

AMD64 架构64位模式下寄存器集

图片来源：*AMD64 Architecture Programmer's Manual*, <https://www.amd.com/system/files/TechDocs/>

寄存器

- 通用寄存器可以通过D触发器实现
 - 一个D触发器相当于1-bit的寄存器
 - D触发器通过并联实现多位寄存器（共享时钟）
- 寄存器的大小：现有计算机通常是16/32/64位
- 寄存器的数量：通常是2的倍数，常见的数量是16/32/64
- 寄存器的内容：数据、地址和控制信息
- 专用寄存器
 - 仅保存数据、地址或控制信息：例如PC寄存器只保存地址

64-Bit Mode ¹		
Name	Number	Size (bits)
RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8–R15	16	64
YMM0–YMM15 ³	16	256
XMM0–XMM15 ³	16	128
MMX0–MMX7 ⁴	8	64
FPR0–FPR7 ⁴	8	80
RIP	1	64
RFLAGS	1	64
—		64

AMD64 架构64位模式下寄存器集

图片来源：*AMD64 Architecture Programmer's Manual*, <https://www.amd.com/system/files/TechDocs/>

寄存器

- 通用寄存器可以通过D触发器实现
 - 一个D触发器相当于1-bit的寄存器
 - D触发器通过并联实现多位寄存器（共享时钟）
- 寄存器的大小：现有计算机通常是16/32/64位
- 寄存器的数量：通常是2的倍数，常见的数量是16/32/64
- 寄存器的内容：数据、地址和控制信息
- 专用寄存器
 - 仅保存数据、地址或控制信息：例如PC寄存器只保存地址
 - 表示某些特殊含义的寄存器：例如AMD64中RFLAGS寄存器中OF对应的bit位代表了算术运算中是否发生溢出

64-Bit Mode ¹		
Name	Number	Size (bits)
RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8–R15	16	64
YMM0–YMM15 ³	16	256
XMM0–XMM15 ³	16	128
MMX0–MMX7 ⁴	8	64
FPR0–FPR7 ⁴	8	80
RIP	1	64
RFLAGS	1	64
—		64

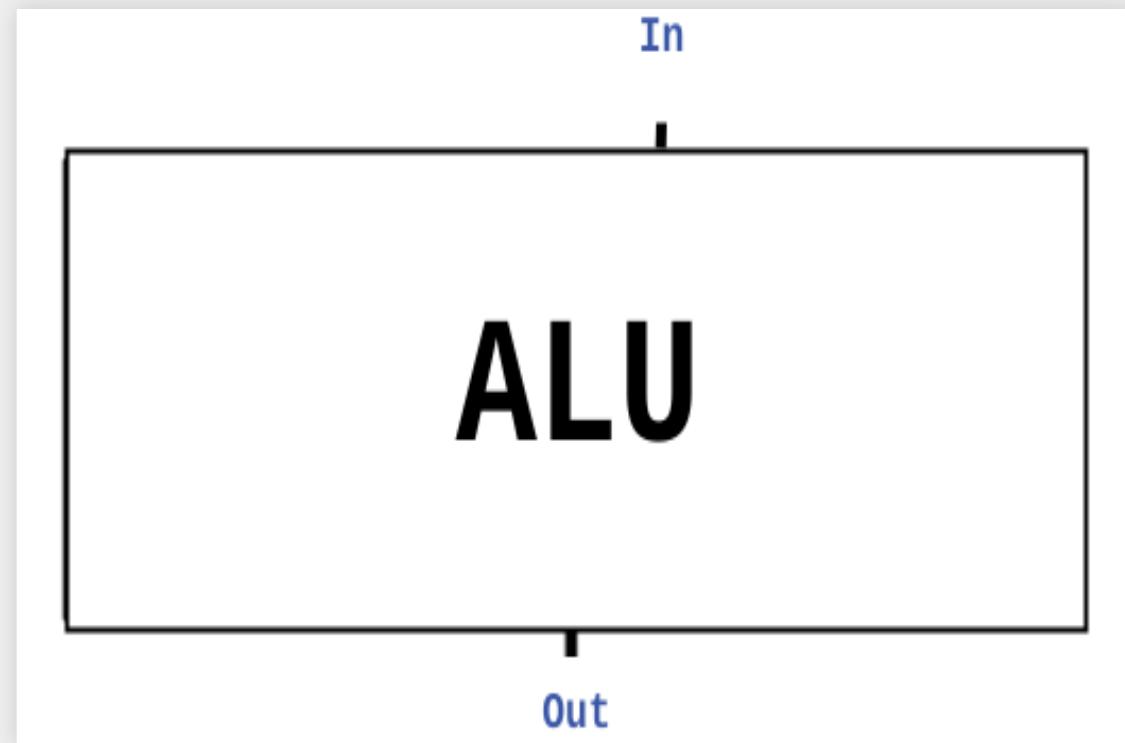
AMD64 架构64位模式下寄存器集

图片来源：*AMD64 Architecture Programmer's Manual*, <https://www.amd.com/system/files/TechDocs/>

算数逻辑单元 (ALU)

算数逻辑单元 (ALU)

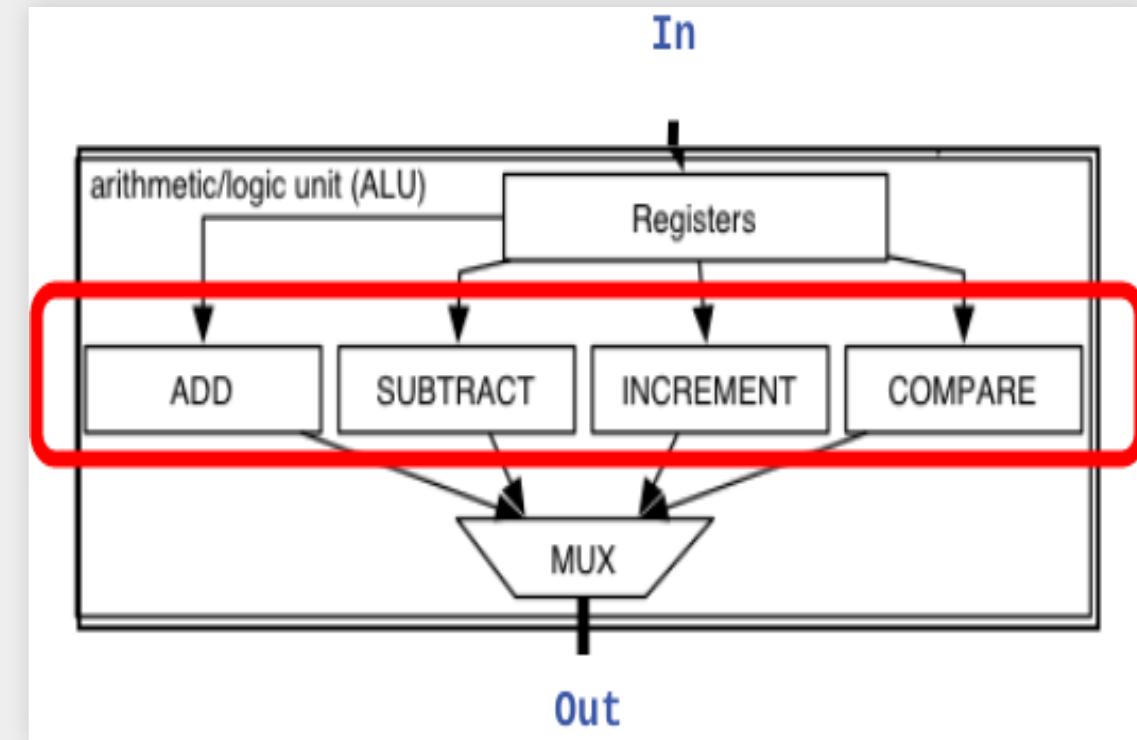
- ALU一般具有多个数据输入和一个数据输出



简化ALU示例

算数逻辑单元 (ALU)

- ALU一般具有多个数据输入和一个数据输出
- ALU的内部实现是多个数字逻辑电路并联

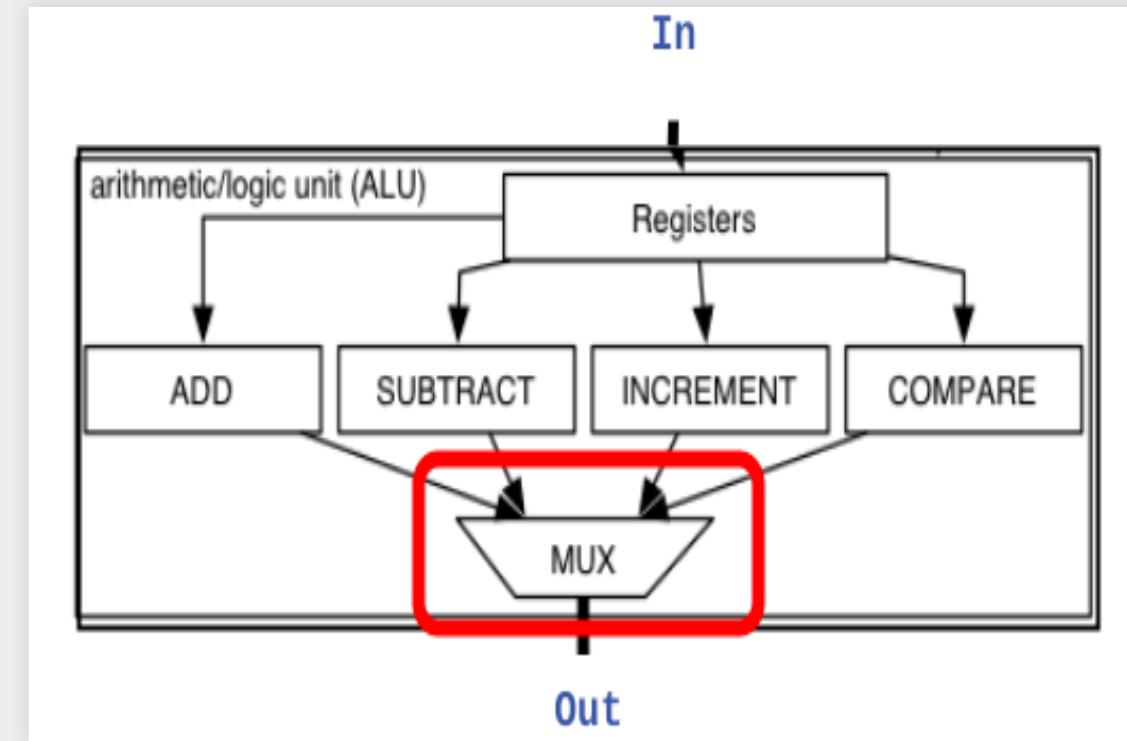


简化ALU示例

图片来源：<https://www.cs.dartmouth.edu/~cbk/classes/4/06x/notes/17-architecture.html>

算数逻辑单元 (ALU)

- ALU一般具有多个数据输入和一个数据输出
- ALU的内部实现是多个数字逻辑电路并联
- 多个数字逻辑电路运算的结果通过一个多路复用器 (MUX) 合并为最终的输出结果



简化ALU示例

控制单元

控制单元

- 寄存器的数据什么时候该写入?
- ALU的最终输出应该被选择哪个子数字逻辑电路的结果? 应该写入哪个寄存器?
- PC寄存器的值应该怎么样变化?
- ...

控制单元

- 寄存器的数据什么时候该写入?
- ALU的最终输出应该被选择哪个子数字逻辑电路的结果? 应该写入哪个寄存器?
- PC寄存器的值应该怎么样变化?
- ...

这些都靠控制单元

控制单元

- 寄存器的数据什么时候该写入?
- ALU的最终输出应该被选择哪个子数字逻辑电
路的结果? 应该写入哪个寄存器?
- PC寄存器的值应该怎么样变化?
- ...

这些都靠控制单元

- 控制单元的输入包括总线、ALU的计算标志位、某些专用寄存器等
- 控制单元的输出是CPU内部数据通路各组件的控制信息，例如
 - 寄存器时钟电位
 - ALU多路复用的选择码等
 - PC寄存器的输入
 - ...

如何设计CPU

思考如下问题：

- 需要多少个寄存器？
- 每个寄存器分别是多少位？
- ALU支持多少个数据输入？
- ALU需要实现哪些算数逻辑电路？
- ALU和寄存器应该如何连接？
- 控制单元有哪些输入？
- 如何决定对应的控制信息？

在下一讲中，我们以MARIE计算机为例来试图回答这些问题

总线和常见的总线类型

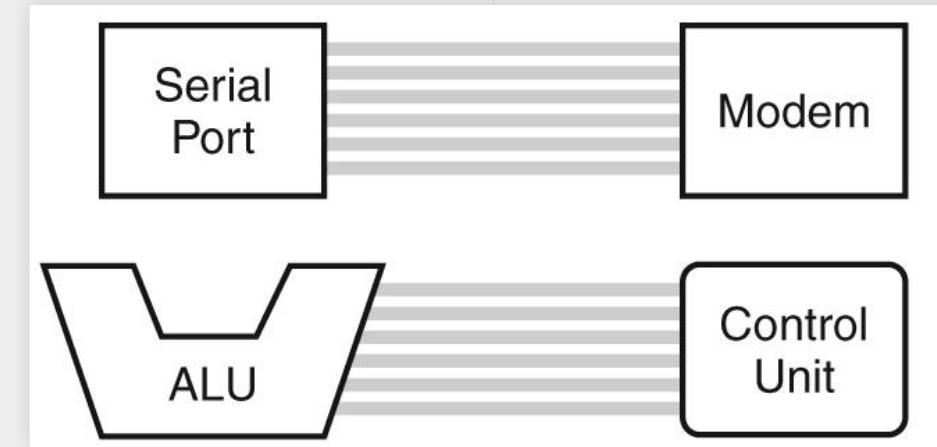
一个总线（Bus）是若干电路连接的集合

总线和常见的总线类型

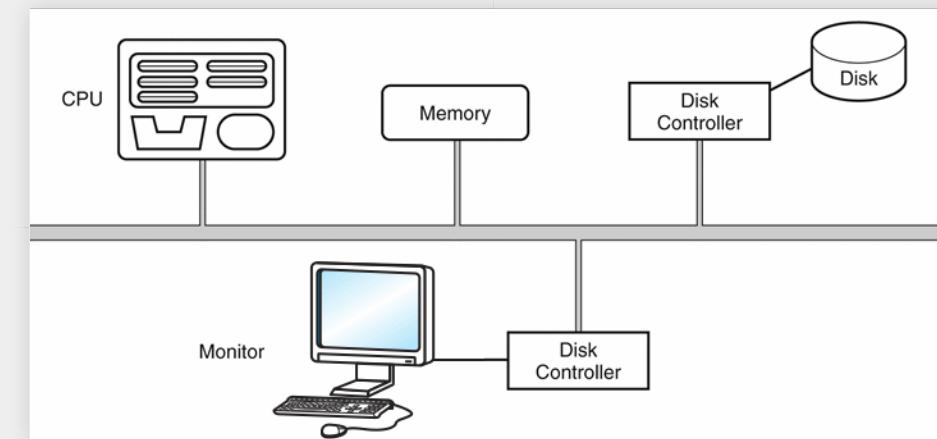
一个总线（Bus）是若干电路连接的集合

按照**连接方式**分类：

- 点对点总线（Point-to-Point Bus）
- 多点总线（Multipoint Bus）



点对点总线



多点总线

总线和常见的总线类型

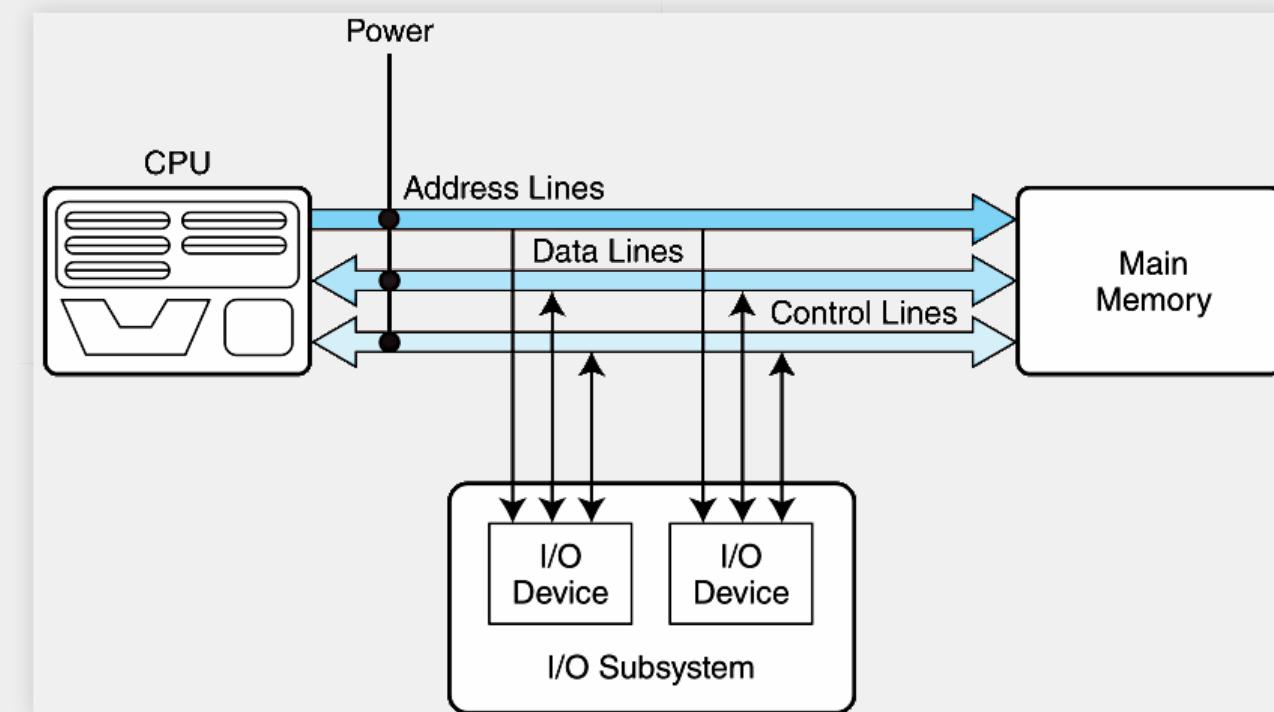
一个总线（Bus）是若干电路连接的集合

按照**连接方式**分类：

- 点对点总线（Point-to-Point Bus）
- 多点总线（Multipoint Bus）

按照**传输内容**分类：

- 数据总线
- 地址总线
- 控制总线

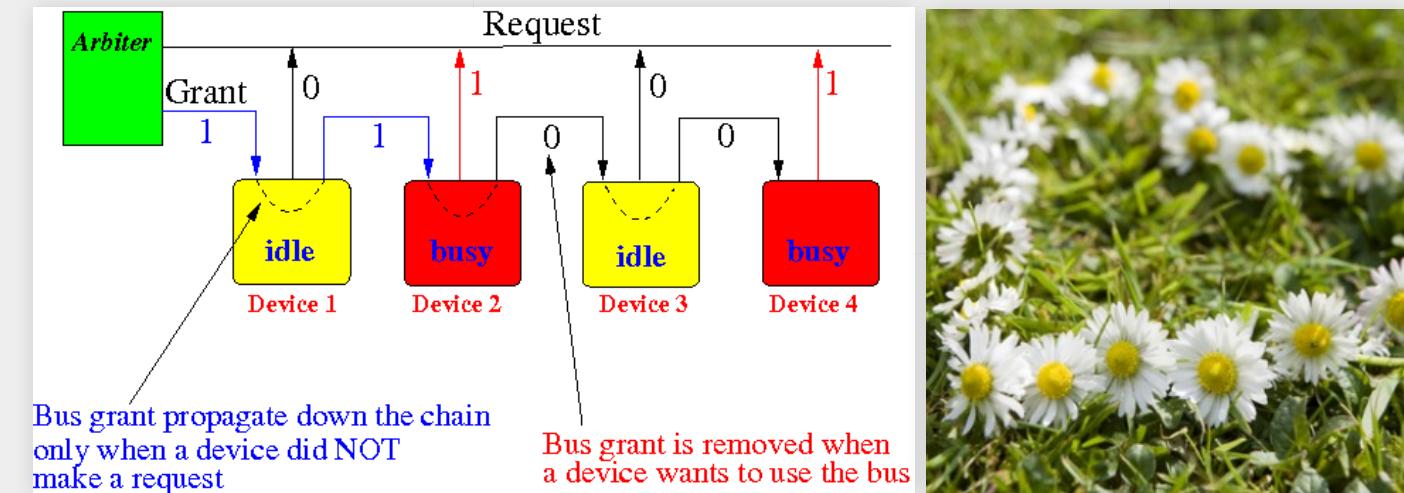


三种类型的总线

总线仲裁机制

总线仲裁机制

- 菊链 (Daisy Chain) 仲裁机制



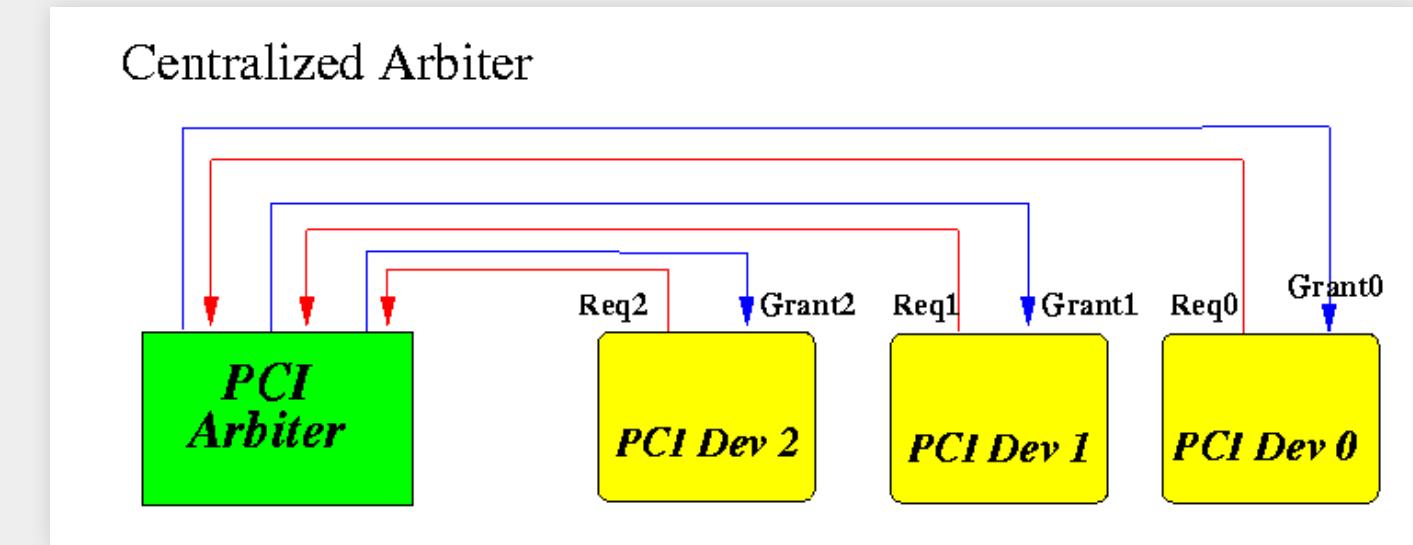
- 所有设备共用一个控制线发送请求
- 控制许可按优先级从高到低传输
- 发送控制请求的设备中优先级最高的设备获得控制许可并清除控制许可

图片来源：<http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus/5-bus/bus-arbiter.html>

图片来源：<http://www.helpmykidlearn.ie/activities/5-7/detail/daisy-chains>

总线仲裁机制

- 菊链 (Daisy Chain) 仲裁机制
- 集中式并行 (Centralized Parallel)

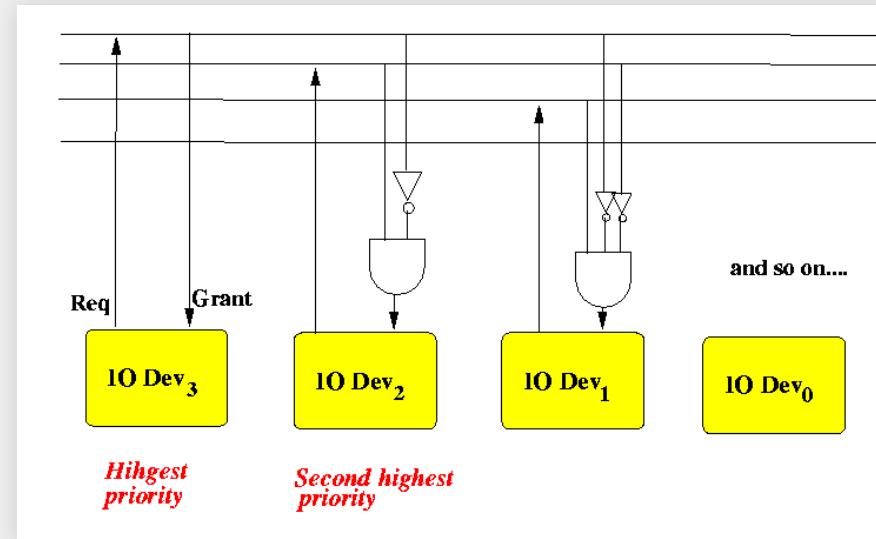


- 由一个中央仲裁根据优先级进行仲裁
- 所有设备和中央仲裁之间有两个控制线，分别发送请求和接收控制许可
- PCI总线采用了这种仲裁方式

图片来源：<http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus/5-bus/bus-arbiter.html>

总线仲裁机制

- 菊链 (Daisy Chain) 仲裁机制
- 集中式并行 (Centralized Parallel)
- 基于自检测的分布式仲裁 (Distributed using Self-Detection)

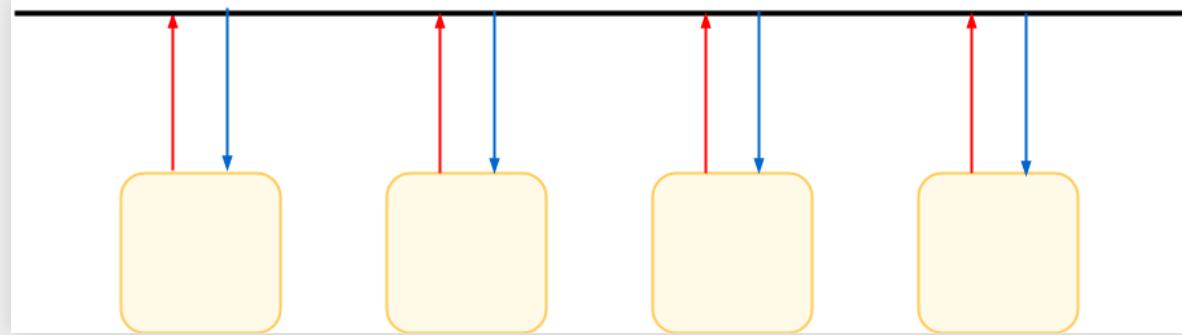


- 每个设备有一个独立的控制线用于发送请求
- 按优先级从高到低排列，第 $i+1$ 个设备监听了前 i 个设备的请求控制线
- 若第 i 个设备发送了请求，当前 $i-1$ 个设备都没有发送请求时，该设备获得总线控制许可

图片来源：<http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus/5-bus/bus-arbiter.html>

总线仲裁机制

- 菊链 (Daisy Chain) 仲裁机制
- 集中式并行 (Centralized Parallel)
- 基于自检测的分布式仲裁 (Distributed using Self-Detection)
- 基于冲突检测的分布式仲裁 (Distributed using Collision-Detection)



- 所有设备共享同一个控制线
- 如果检测到冲突，则重新发送请求，否则发送请求的设备获得控制许可
- 以太网采用了这种仲裁方式

时钟与CPU时间

- 时钟定期产生电子脉冲 (pulse)
- CPU利用时钟脉冲同步各组件之间的数据传输
- 时钟频率代表了时钟脉冲信号产生的频率
- 时钟频率是时钟周期的倒数
- 时钟频率并不能完全决定CPU的性能

$$\text{CPU Time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{avg. cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

CPU时间计算公式

I/O子系统

- I/O子系统并不直接连接CPU，而是通过特定的接口
- 两类常见的连接方式：
 - 基于内存映射 (memory-mapped) 的I/O
 - 基于指令的I/O

更多详细内容我们在后续课程中深入介绍



第二讲结束

本期内容回顾

- 非冯-诺依曼模型
 - 单核非冯-诺依曼模型
 - 并行计算
 - 异构计算
- 数字电路基础回顾
 - 数据表示
 - 布尔代数
- CPU基本知识和组织结构
 - 寄存器、ALU、控制单元
 - 总线、时钟、I/O子系统

扩展阅读

- 并行计算
 - https://computing.llnl.gov/tutorials/parallel_comp/
- GPU
 - <https://courses.cs.washington.edu/courses/cse471/13sp/lectures/GPUsStudents.pdf>
 - <http://www.imooc.com/article/256049> (中文)
- DSP
 - <http://bwrcs.eecs.berkeley.edu/Classes/CS252/Notes/Lec09-DSP.pdf>
- NPU
 - <http://klamath.stanford.edu/~nickm/papers/RMT-SIGCOMM.pdf>
- APU
 - <https://ieeexplore.ieee.org/iel7/8126322/8192462/08192463.pdf>
- 集群计算
 - <http://www.netlib.org/utk/people/JackDongarra/PAPERS/sunway-report-2016.pdf>



Q & A