

计算机体系结构



2018-8-18

版权所有，未经同意，不得修改

编辑历史

|  |  |  |
| --- | --- | --- |
| **时间** | **修订人** | **说明** |
| 2018-08-18 | 邹弢 | 开始编写 |
|  |  |  |
|  |  |  |

计算机体系结构

# 背景

体系结构:

* **designing, selecting, and interconnecting**硬件组件
* designing 计算机系统的软件硬件接口
* 满足：**functional, performance, energy consumption, cost** 和其它目标

体系结构从另一个角度来看：体系结构也是一种算法，只是该算法由硬件实现。

体系结构的用户：可以是Problem/Algorithm/Program/Language/Runtime System(VM/OS/MM)

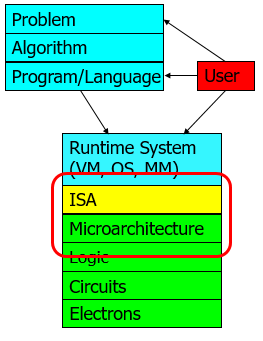


图 1 体系结构的用户

学习要点：

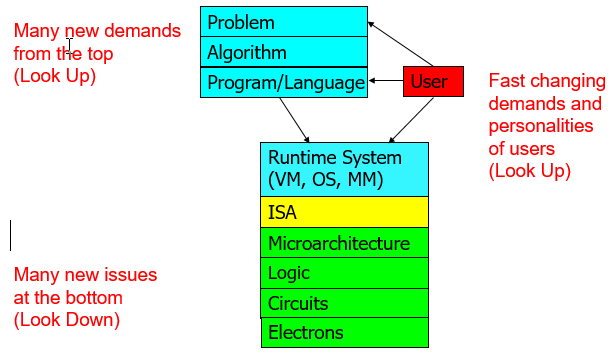
* 关注现代处理器的主要组件、软硬件接口的设计原则和权衡。
* 关注如何设计、实现和评估现代处理器

**目标：**

* 熟悉现代计算系统中操作原则和设计权衡：处理器、内存、平台构架。预测技术演进。
* 通过RTL/C实现，提供背景和经验，以设计、实现和评估现代处理器。

计算机体系结构的挑战：

* Power/energy constraints 🡪 multi-core?
* Complexity of design 🡪 multi-core?
* Difficulties in technology scaling 🡪 new technologies?
* Memory wall/gap
* Reliability wall/issues
* Programmability wall/problem
* Huge hunger for data and new data-intensive applications

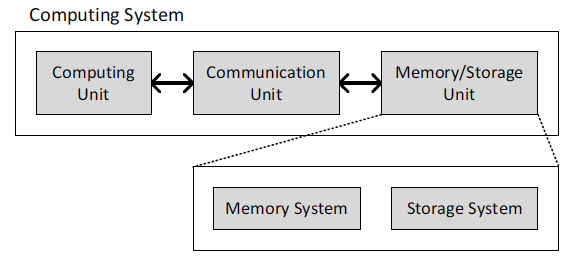


新的需要和约束条件，导致所有的组件及其接口，以及整个系统都需要重新审视。

## 计算机模型

计算机关键组件：

* Computation
* Communication
* Storage (memory)



### Von Neumann模型：也叫存储程序

Von Neumann模型：也叫存储程序

* 存储程序：Stored program
  + 指令存储在线性内存单元中。
  + 指令和数据统一存储：数据的解释依赖于控制信号
* 顺序执行：Sequential instruction processing
  + 一次执行一条指令 (fetched, executed, and completed)
  + Program counter (instruction pointer) 指示当前指令。
  + Program counter顺序递增，除非有分支跳转。

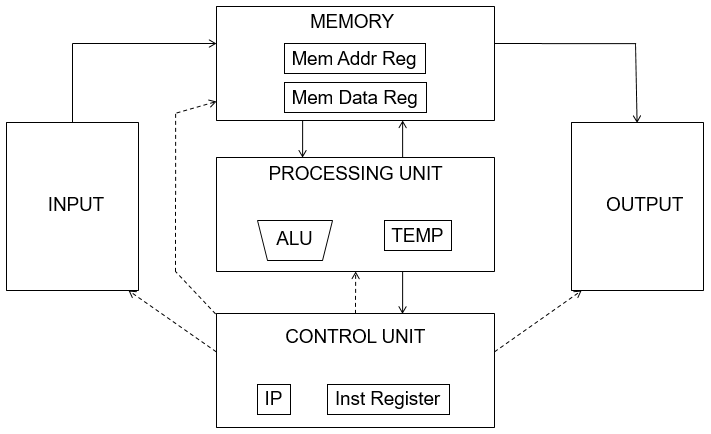


图 2 Von Neumann模型

Von Neumann模型与数据流模型：

|  |  |
| --- | --- |
| Von Neumann模型 | Dataflow模型 |
| 指令按控制流取指并执行：   * 指令只PC指令指示。 * 顺序执行，除非有分支跳转。 | Dataflow模型：指令按数据流的顺序取指和执行。   * 当指令操作数准备OK * 没有指令指针IP。 * 指令顺序由数据流依赖指定：   + 每条指令指定who应该接收结果。   + 当指令的所有操作数都收到后，该指令可以fire。 * 多条指令可能同时执行：并行性更强。 |
| v <= a + b;  w <= b \* 2;  x <= v - w  y <= v + w  z <= x \* y |  |

### Dataflow模型

Dataflow模型

**Data flow结点**

Data flow结点和ISA表示：

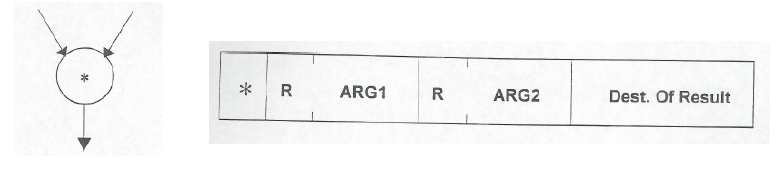


图 3 Data flow结点和ISA表示

**Data flow结点：**分支、关系、同步屏障

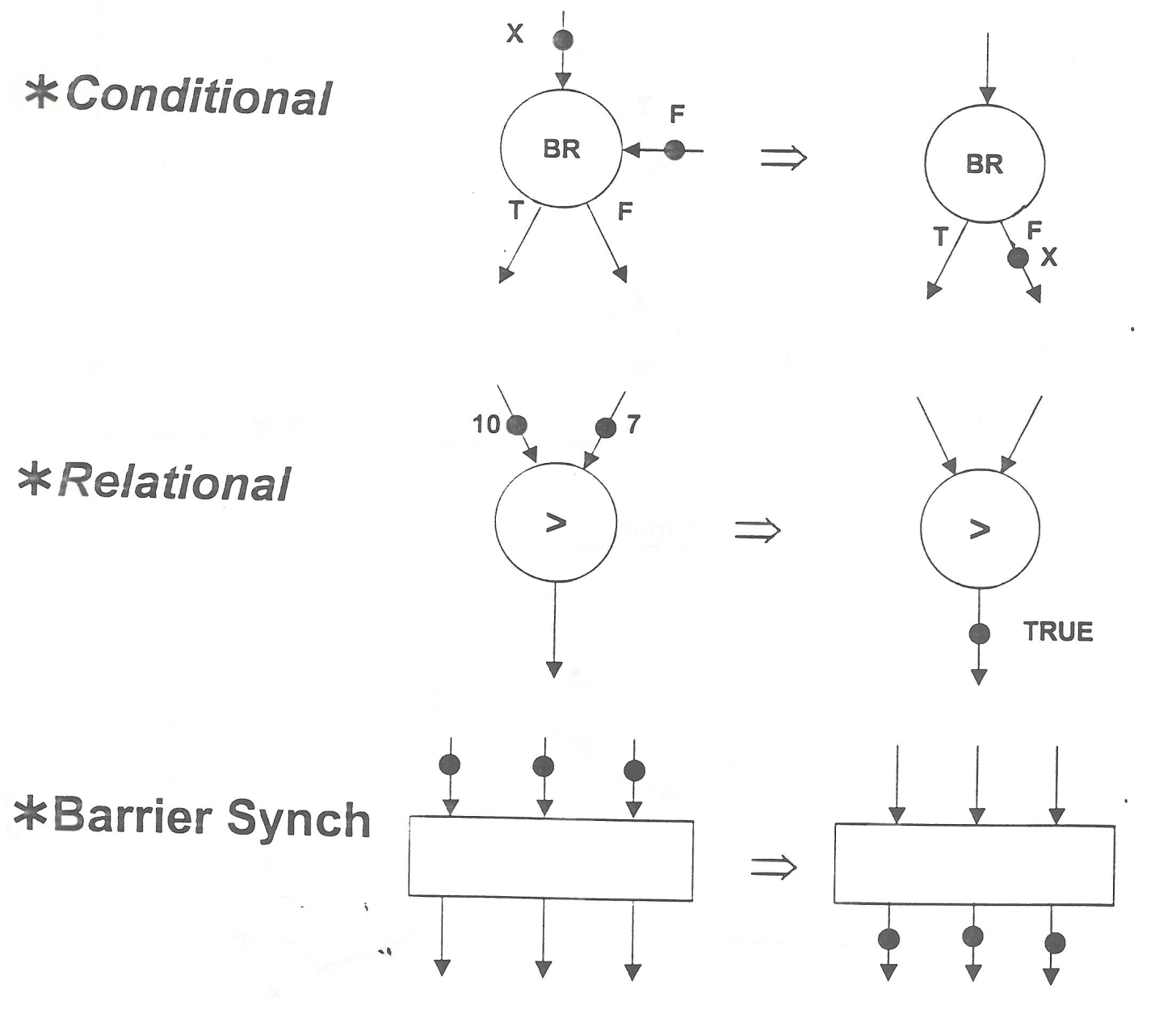


图 4 Data flow结点

## ISA & Microarchitecture

**ISA &** **uarch & uprocessor：**

* ISA
  + 软件<->硬件接口
    - SW/compiler assumes, HW promises
  + 编程需要知道的信息。
  + eg: x86, ARM, SPARC, MIPS, Alpha
* Microarchitecture
  + Specific **implementation of an ISA**
  + Not visible to the software
  + uarch的实现有多种，只要满足ISA即可。
  + uarch比ISA变更的更快。
* Microprocessor
  + **ISA, uarch**, circuits
  + “Architecture” = ISA + microarchitecture

ISA内容：

* 指令
  + Opcodes, Addressing Modes, Data Types
  + Instruction Types and Formats
  + Registers, Condition Codes
* Memory
  + Address space, Addressability, Alignment
  + Virtual memory management
* Call, Interrupt/Exception Handling
* Access Control, Priority/Privilege
* I/O: memory-mapped vs. instr.
* Task/thread Management
* Power and Thermal Management
* Multi-threading support, Multiprocessor support

Microarchitecture

* 基于特定的约束和目标实现ISA
* 其它由硬件实现，并且未暴露给软件的
  + Pipelining
  + In-order versus out-of-order instruction execution
  + Memory access scheduling policy
  + Speculative execution
  + Superscalar processing (multiple instruction issue?)
  + Clock gating
  + Caching? Levels, size, associativity, replacement policy
  + Prefetching?
  + Voltage/frequency scaling?
  + Error correction?

# 处理器微结构

## 乱序执行

**静态 vs 动态流水线调度：**

* **静态调度：**
  + 基于软件（编译器）的指令调度。
  + 难点：变长的操作时延，内存地址，分支方向未知。
* **动态调度：**
* 基于硬件的指令调度
* 动态信息可利用：Cache缺失、分支预测失败、Load/Store地址

**In-Order Pipeline：**

F

D

E

R

E

E

E

E

E

E

E

E

E

E

E

E

E

E

E

E

E

E

E

E

**. . .**

Integer add

Integer mul

FP mul

Cache miss

W

* 问题：数据依赖会推迟后续指令的Dispatch（发送一条指令到Functional Unit - FU）。
* 阻塞示例：ADD指令阻塞了整个流水线，后续的独立指令也无法执行。

|  |  |
| --- | --- |
| IMUL R3 🡨 R1, R2  **ADD R3 🡨 R3, R1**  ADD R1 🡨 R6, R7  IMUL R5 🡨 R6, R8  ADD R7 🡨 R9, R9 | LD R3 🡨 R1 (0) //延迟未知,编译器优化流水线困难  **ADD R3 🡨 R3, R1**  ADD R1 🡨 R6, R7  IMUL R5 🡨 R6, R8  ADD R7 🡨 R9, R9 |

**防止Dispatch Stall的方法：**

* 粒度合适的多线程。 -- 不能改善单线程的性能
* 值预测 – 不总能预测准确
* 编译时的指令调度和重排序 – 缺少动态信息，调度困难
* 乱序Dispatch/Scheduling/execution：跟踪数据流，当指令依赖的数据ready后，才发射指令。

**乱序执行的思路：**

* **分离指令：**将有依赖的指令与无依赖的指令分开。无依赖指令可执行，有依赖指令暂存在 Reservation Station（保留站）中。
* **监控输入源：**监控保留站中每条指令所有输入数据的ready情况。
* **发射指令：**当一条指令的所有数据准备ok，Dispatch该指令。即：指令Dispatch按数据流顺序，而不是按控制流顺序。
* **优点：**当出现长延迟操作时，允许无依赖指令执行。

**顺序执行 vs 乱序执行：**

|  |  |
| --- | --- |
| IMUL R3 🡨 R1, R2  ADD R3 🡨 R3, R1  ADD R1 🡨 R6, R7  IMUL R5 🡨 R6, R8  ADD R7 🡨 R3, R5 | |
| 图 5 顺序执行 + 准确异常 – 16cycle | 图 6 乱序执行 + 准确异常 – 12cycle |

**实现乱序执行的思路：**

1. **关联Value：**通过寄存器重命名，将【tag】与每个数据相联，建立consumer <-> producer的连接
2. **缓存指令：**将重命名后的指令插入【保留站】
3. **跟踪指令Source Value：**
   1. 值产生后，广播【tag】
   2. 指令比较【source tag】与【broadcast tag】，相等则ready
4. 当一条指令的所有Source Value准备ok，**Dispatch该指令**到Functional Unit(FU)
   1. 所有Source ok -> 指令wake up
   2. 如果多个指令ok，每个FU选择一条指令。

### Tomasulo算法

#### 背景

**背景：**寄存器重命令的乱序执行算法，由Robert Tomasulo发明，应用于IBM 360/91 浮点运算单元（FPU – Floating Point Unit）

**相比于当今算法的主要不同：**

* 支持精确异常：IBM 360/91不支持

**现代Pipeline结构：**

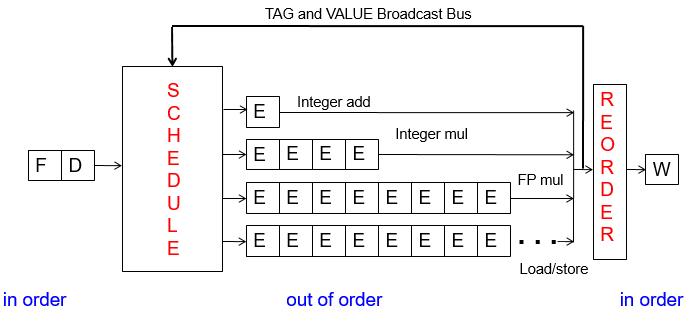


图 7 现代Pipeline结构

* Schedule模块：保留站（scheduling window）
* Reorder模块：Reordering（reorder buffer, instruction window），保证Load/Store的顺序。

乱序执行处理器结构：

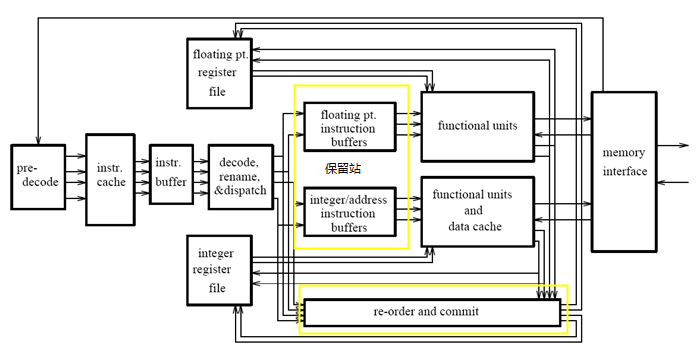


图 8 乱序执行处理器结构

**Tomasulo机器：**IBM 360/91结构

取指->译码->分发到保留站等待操作数->操作数ready,Dispatch到执行单元(FP FU)->执行结果经CDB总线广播: tag + value-> 匹配tag， 更新FP Register和保留站指令等待状态。

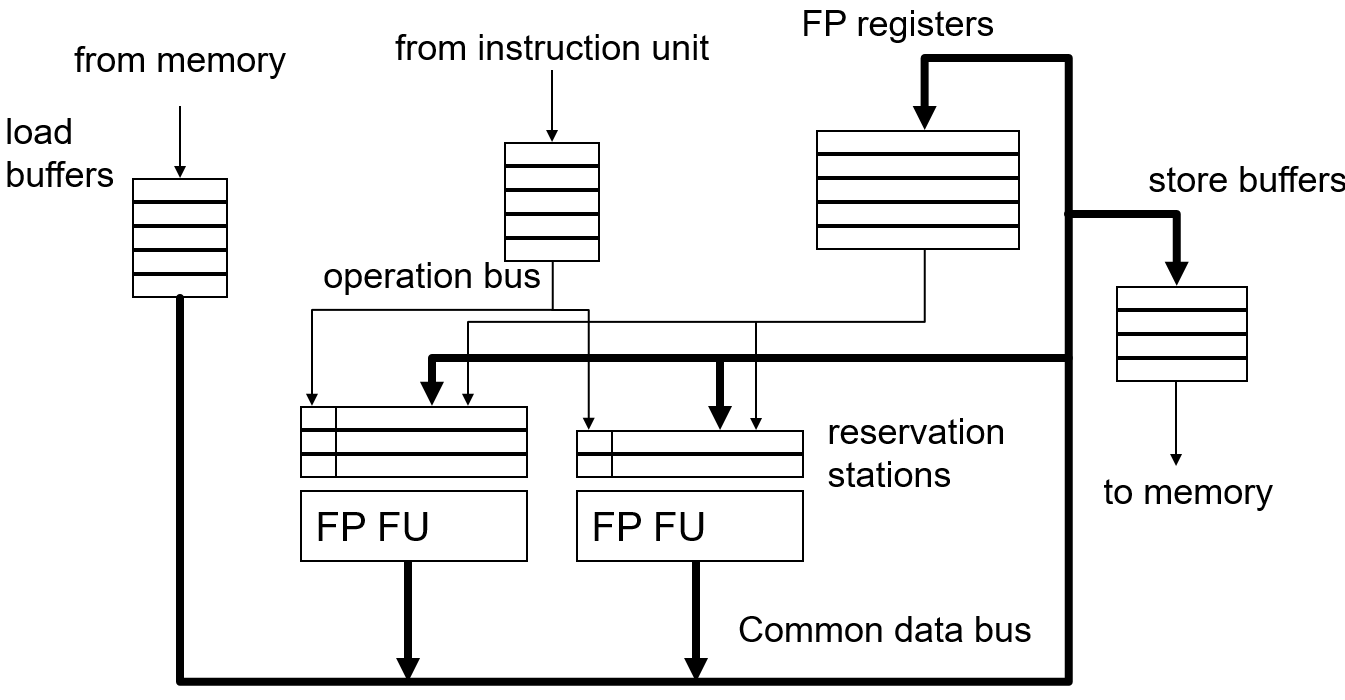


图 9 IBM 360/91流水线结构

#### 算法流程

**寄存器重命名：**

* Tomasulo算法的寄存器重命名规则：将Register ID重命名为保留站中持有该寄存器值的RS Entry ID，此ID即tag。
* 作用：可以消除输出依赖和自依赖。这些依赖都是应该寄存器不够导致的。

**Tomasulo算法流程：**



图 10 Tomasulo调度算法流程

#### 调度示例

调度示例：

|  |  |
| --- | --- |
| MUL R3 <- R1, R2  ADD R5 <- R3, R4  ADD R7 <- R2, R6  ADD R10 <- R8, R9  MUL R11 <- R7, R10  ADD R5 <- R5, R11 | **执行时间（EX）：** ADD - 1 cycle；MUL – 3 cycle  **执行器件：**加法器一个，乘法器一个 |

**In-order流水线周期：**

* 采用计分板方法调度
* 非精确异常
* 指令依赖的寄存器在译码阶段之后被更新，需要重新译码（获取寄存器值）

**no forwarding版本周期：** W后延才能获取写入寄存器的值。



图 11 In-Order流水线周期示例 - no forwarding

**full forwarding版本周期：**



图 12 In-Order流水线周期示例 – full forwarding

**Out-of-Order流水线周期：**非精确异常，Tomasulo调度算法



Tomasulo调度 – 各周期运行情况：



附录

# 术语

|  |  |  |
| --- | --- | --- |
| 术语 | 全称 | 描述 |
| FU | Functional Unit | 功能单元，如ALU，FPU |
| FPU | Floating Point Unit | 浮点运算单元 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 引用

# 标准样式

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **项目** | **描述** |
|  |  |
|  |  |
|  |  |