第2章　计算机指令集结构

2.4 操作数的类型和大小

- 数据表示：计算机硬件能够直接识别、指令集可以直接调用的数据类型。
  - 所有数据类型中最常用、相对比较简单、用硬件实现比较容易的几种。
- 数据结构：由软件进行处理和实现的各种数据类型。

研究：这些数据类型的逻辑结构与物理结构之间的关系，并给出相应的算法。

系统结构设计者要解决的问题：如何确定数据表示？

（软硬件取舍折中的问题）

2.4 操作数的类型和大小

1. 表示操作数类型的方法有两种

- 由指令中的操作码指定操作数的类型。（多用）eg：ADDI
- 带标志符的数据表示。给数据加上标识，由数据本身给出操作数类型。（少用）eg：ADD 1, A, B 标识符.

  - 优点：简化指令集，可由硬件自动实现一致性检查和类型转换，缩小了机器语言与高级语言的语义差距，简化编译器等。
  - 缺点：由于需要在执行过程中动态检测标志符，动态开销比较大，所以采用这种方案的机器很少见。

2. 操作数的大小：操作数的位数或字节数。

主要的大小：字节（8位）、半字（16位）
字（32位）、双字（64位）

2.5 指令格式的设计

1. 指令由两部分组成：操作码、地址码 指令=操作码+地址码
2. 指令格式的设计

   确定指令字的编码方式，包括操作码字段和地址码字段的编码和表示方式。
3. 操作码的编码比较简单和直观
   - Huffman编码法

     减少操作码的平均位数，但所获得的编码是变长的，不规整，不利于硬件处理。
   - 固定长度的操作码（现在多用）

     保证操作码的译码速度。

4. 两种表示寻址方式的方法

- 将寻址方式编码于操作码中，由操作码描述相应操作的寻址方式。

  适合：处理机采用load-store结构，寻址方式只有很少几种。

- 设置专门的地址描述符，由地址描述符表示相应操作数的寻址方式。

  适合：处理机具有多种寻址方式，且指令有多个操作数。

5. 考虑因素

- 机器中寄存器的个数和寻址方式的数目对指令平均字长的影响以及它们对目标代码大小的影响。
- 所设计的指令格式便于硬件处理，特别是流水实现。
- 指令字长应该是字节（8位）的整数倍，而不能是随意的位数。

6. 指令集的3种编码格式

变长编码格式、定长编码格式、混合型编码格式

变长编码格式

- 当指令集的寻址方式和操作种类很多时，这种编码格式是最好的。
- 用最少的二进制位来表示目标代码。
- 可能会使各条指令的字长和执行时间相差很大。

CISC指令集多用

操作码	地址描述符 1	地址码 1	…	地址描述符 n	地址码 n

定长编码格式

- 将操作类型和寻址方式一起编码到操作码中。
- 当寻址方式和操作类型非常少时，这种编码格式非常好。
- 可以有效地降低译码的复杂度，提高译码的速度。
- 大部分RISC的指令集均采用这种编码格式。

操作码	地址码 1	地址码 2	地址码 3

混合型编码格式

- 提供若干种固定的指令字长。
- 以期达到既能够减少目标代码长度又能降低译码复杂度的目标。

2.6 MIPS指令集结构

一条指令长度为64位

介绍MIPS64的一个子集，简称为MIPS。

2.6.1 MIPS的寄存器

1. 32个64位通用寄存器（GPRs）
   - R0，R1，...，R31
   - 也被称为整数寄存器
   - R0的值永远是0
2. 32个64位浮点数寄存器（FPRs）
   - F0，F1，...，F31

- 用来存放32个单精度浮点数（32位），也可以用来存放32个双精度浮点数（64位）。用操作码指明
- 存储单精度浮点数（32位）时，只用到FPR的一半，其另一半没用。

3. 一些特殊寄存器

- 它们可以与通用寄存器交换数据。
- 例如，浮点状态寄存器用来保存有关浮点操作结果的信息。

2.6.2 MIPS的数据表示

1. MIPS的数据表示

- 整数

  字节（8位）　　半字（16位）

  字（32位）　　双字（64位）

- 浮点数

  单精度浮点数（32位）　双精度浮点数（64位）

2. 字节、半字或者字在装入64位寄存器时，用零扩展或者用符号位扩展来填充该寄存器的剩余部分。装入以后，对它们将按照64位整数的方式进行运算。

2.6.3 MIPS的数据寻址方式

1. 立即数寻址与偏移量寻址

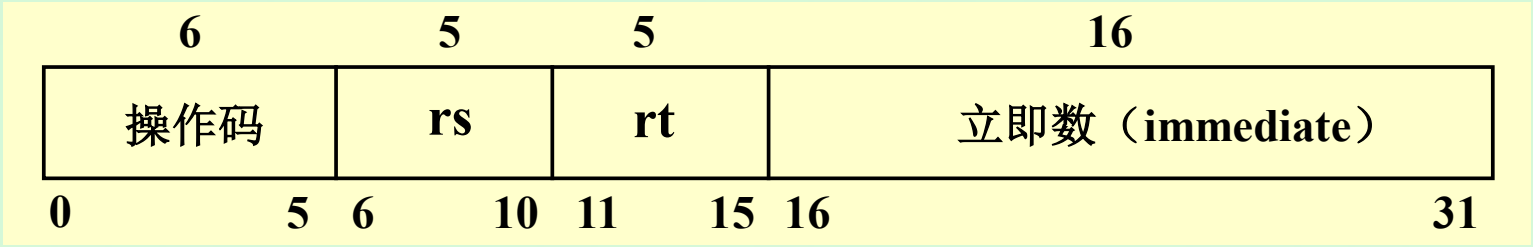
   立即数字段和偏移量字段都是16位的。

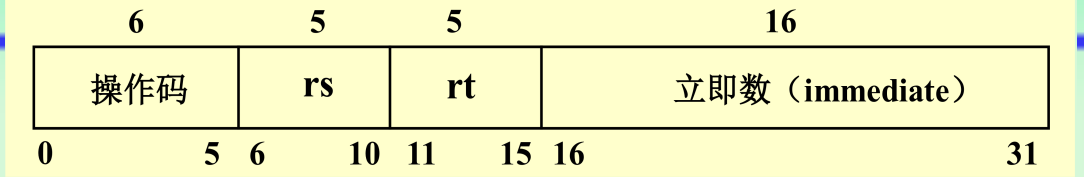
2. 寄存器间接寻址是通过把0作为偏移量来实现的
3. 16位绝对寻址是通过把R0（其值永远为0）作为基址寄存器来完成的
4. MIPS的存储器是按字节寻址的，地址为64位
5. 所有存储器访问都必须是边界对齐的

2.6.4 MIPS的指令格式

1. 寻址方式编码到操作码中
2. 所有的指令都是32位的
3. 操作码占6位
4. 3种指令格式

- I类指令 Immediate
  - 包括所有的load和store指令、立即数指令、，分支指令、寄存器跳转指令、寄存器链接跳转指令。
  - 立即数字段为16位，用于提供立即数或偏移量。





- load指令

  访存有效地址：Regs[rs]＋immediate

  从存储器取来的数据放入寄存器rt

- store指令

  访存有效地址：Regs[rs]＋immediate

  要存入存储器的数据放在寄存器rt中

- 立即数指令

  Regs[rt] ← Regs[rs] op immediate

- 分支指令

  转移目标地址：Regs[rs]＋immediate，rt无用

- 寄存器跳转、寄存器跳转并链接

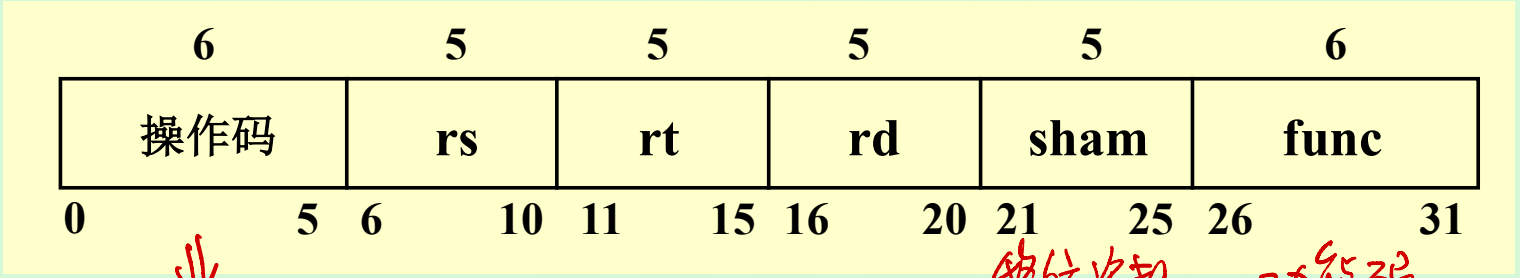
  转移目标地址为Regs[rs]

R类指令

- 包括ALU指令、专用寄存器读/写指令、move指令等。
- ALU指令

  Regs[rd]← Regs[rs] funct Regs[rt]

  func为具体的运算操作编码



指明3操作类型 ⇒ 每个寄存器只需5位.
(寄址or总线相)

移位次数

功能码

J类指令

- 包括跳转指令、跳转并链接指令、自陷指令、异常返回指令。
- 在这类指令中，指令字的低26位是偏移量，它与PC值相加形成跳转的地址。

2.6.5 MIPS的操作

1. MIPS指令可以分为四大类

- load和store
- ALU操作
- 分支与跳转
- 浮点操作

2. 符号的意义

- $x \leftarrow_n y$：从y传送n位到x
- x，y←z：把z传送到x和y

- 下标：表示字段中具体的位；
  - 对于指令和数据，按从最高位到最低位（即从左到右）的顺序依次进行编号，最高位为第0位，次高位为第1位，依此类推。
  - 下标可以是一个数字，也可以是一个范围。

    例如：$Regs[R4]_0$：寄存器R4的符号位

    $Regs[R4]_{56..63}$：R4的最低字节
- Mem：表示主存；
  - 按字节寻址，可以传输任意个字节。
- 上标：用于表示对字段进行复制的次数。

  例如：0^{32}：一个32位长的全0字段

- 符号##：用于两个字段的拼接，并且可以出现在数据传送的任何一边。

举例：R8、R10：64位的寄存器，则

$$Regs[R8]_{32..63} \leftarrow_{32} (Mem\ [Regs[R6]]_0)^{24}\ \#\#\ Mem\ [Regs[R6]]$$

表示的意义是：

以R6的内容作为地址访问内存，得到的字节按符号位扩展为32位后存入R8的低32位，R8的高32位（即$Regs[R8]_{0..31}$）不变。

3. load和store指令

指令举例	指令名称	含 义
LD R2，20(R3)	装入双字 Double	Regs[R2] $\leftarrow_{64}$ Mem[20+Regs[R3]]
LW R2，40(R3)	装入字 Word	Regs[R2] $\leftarrow_{64}$ (Mem[40+Regs[R3]]$_0$)^{32} ## Mem[40+Regs[R3]]
LB R2，30(R3)	装入字节 Byte	Regs[R2] $\leftarrow_{64}$ (Mem[30+Regs[R3]]$_0$)^{56} ## Mem[30+Regs[R3]]
LBU R2，40(R3)	装入无符号字节	Regs[R2] $\leftarrow_{64}$ 0^{56} ## Mem[40+Regs[R3]]
LH R2，30(R3)	装入半字	Regs[R2] $\leftarrow_{64}$ (Mem[30+Regs[R3]]$_0$)^{48} ## Mem[30+Regs[R3]]## Mem[31+Regs[R3]]
L.S F2，60(R4)	装入~~半字~~ 单精度浮点数	Regs[F2] $\leftarrow_{64}$ Mem[60+Regs[R4]] ## 0^{32}
L.D F2，40(R3)	装入双精度浮点数	Regs[F2] $\leftarrow_{64}$ Mem[40+Regs[R3]]
SD R4，300(R5)	保存双字	Mem[300+Regs[R5]] $\leftarrow_{64}$ Regs[R4]
SW R4，300(R5)	保存字	Mem[300+Regs[R5]] $\leftarrow_{32}$ Regs[R4]
S.S F2，40(R2)	保存单精度浮点数	Mem[40+Regs[R2]] $\leftarrow_{32}$ Regs[F2] $_{0..31}$
SH R5，502(R4)	保存半字	Mem[502+Regs[R4]] $\leftarrow_{16}$ Regs[R5] $_{48..63}$

1. LD = LOAD DOUBLE
2. LW = LOAD WORD
3. LB = LOAD BYTE
4. LBU = LOAD BYTE UNSIGNED
5. LH = LOAD HALF WORD
6. L.S = LOAD SINGLE FLOAT
7. L.D = LOAD DOUBLE FLOAT

1. SD = STORE DOUBLE
2. SW = STORE WORD
3. SB = STORE BYTE
4. SBU = STORE BYTE UNSI
5. SH = STORE HALF WORD
6. S.S = STORE SINGLE FL
7. S.D = STORE DOUBLE FL

4. ALU指令

寄存器-寄存器型（RR型）指令或立即数型

算术和逻辑操作：加、减、与、或、异或和移位等

指令举例	指令名称	含义
DADDU R1，R2，R3	无符号加	Regs[R1]← Regs[R2]+ Regs[R3]
DADDIU R4，R5，#6	加无符号立即数	Regs[R4]← Regs[R5]+6
LUI R1，#4	把立即数装入到一个字的高16位	Regs[R1]← 0^{32} ## 4 ## 0^{16}
DSLL R1，R2，#5	逻辑左移	Regs[R1]← Regs[R2]<<5
DSLT R1，R2，R3	置小于	If(Regs[R2]< Regs[R3]) Regs[R1]← 1 else Regs[R1]←0

1. DADDU = DATA ADD UNSIGNED
2. DADDIU = DATA ADD IMMEDIATE UNSIGNED
3. LUI = LOAD UNSIGNED IMMEDIATE
4. DSLL = DATA SHIFT LOGIC LEFT
5. DSLT = DATA SET IF LITTLE

R0的值永远是0，它可以用来合成一些常用的操作。

例如：

```
DADDIU R1，R0，#100
                    //给寄存器R1装入常数100
DADD R1，R0，R2
                    //把寄存器R2中的数据传送到寄存器R1
```

2.6.6 MIPS的控制指令

1. 由一组跳转和一组分支指令来实现控制流的改变
2. 典型的MIPS控制指令

指令举例	指令名称	含义
J name	跳转	PC $_{36..63}$ ← name<<2
JAL name	跳转并链接	Regs[R31] ←PC+4; PC $_{36..63}$ ←name<<2; ((PC+4) −2^{27}) ≤name < ((PC+4) +2^{27})
JALR R3	寄存器跳转并链接	Regs[R31] ←PC+4; PC← Regs[R3]
JR R5	寄存器跳转	PC← Regs[R5]
BEQZ R4, name	等于零时分支	if(Regs[R4]== 0) PC←name ; ((PC+4) −2^{17}) ≤name < ((PC+4) +2^{17})
BNE R3, R4, name	不相等时分支	if(Regs[R3]!= Regs[R4]) PC←name ((PC+4) −2^{17}) ≤name < ((PC+4) +2^{17})
MOVZ R1, R2, R3	等于零时移动	if(Regs[R3]==0) Regs[R1] ← Regs[R2]

3. 跳转指令

- 根据跳转指令确定目标地址的方式不同以及跳转时是否链接，可以把跳转指令分成4种。
- 确定目标地址的方式

  - 把指令中的26位偏移量左移2位（因为指令字长都是4个字节）后，替换程序计数器的低28位。
  - 间接跳转：由指令中指定的一个寄存器来给出转移目标地址。
- 跳转的两种类型

  - 简单跳转：把目标地址送入程序计数器。
  - 跳转并链接：把目标地址送入程序计数器，把返回地址（即顺序下一条指令的地址）放入寄存器R31。

4. 分支指令（条件转移）

- 分支条件由指令确定。

  例如：测试某个寄存器的值是否为零

- 提供一组比较指令，用于比较两个寄存器的值。

  例如：“置小于”指令

- 有的分支指令可以直接判断寄存器内容是否为负，或者比较两个寄存器是否相等。

- 分支的目标地址。

  由16位带符号偏移量左移两位后和PC相加的结果来决定

- 一条浮点条件分支指令：通过测试浮点状态寄存器来决定是否进行分支。

2.6.7 MIPS的浮点操作

1. 由操作码指出操作数是单精度（SP）或双精度（DP）
   - 后缀S：表示操作数是单精度浮点数
   - 后缀D：表示是双精度浮点数
2. 浮点操作

   包括加、减、乘、除，分别有单精度和双精度指令。
3. 浮点数比较指令
   - 根据比较结果设置浮点状态寄存器中的某一位，以便于后面的分支指令BC1T（若真则分支）或BC1F（若假则分支）测试该位，以决定是否进行分支。

第3章 流水线技术

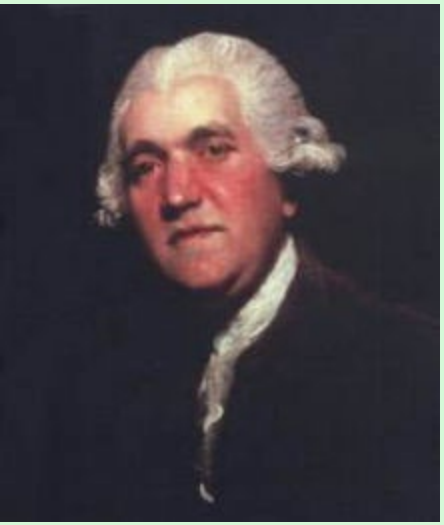
1、下次交作业时间：第五周的周五

第三章 作业 P104

第 2，4, 6, 7, 8, 9, 10, 11, 12, 13,15,16题。

流水线技术的由来

1769年，英国人乔赛亚·韦奇伍德开办埃特鲁利亚陶瓷工厂，在场内实行精细的劳动分工，他把原来由一个人从头到尾完成的制陶流程分成几十道专门工序，分别由专人完成。这样一来，原来意义上的“制陶工”就不复存在了，存在的只是挖泥工、运泥工、扮土工、制坯工等等制陶工匠变成了制陶工场的工人，他们必须按固定的工作节奏劳动，服从统一的劳动管理。







1、一个人从头到尾完成的制陶流程分成几十道专门工序:

2、每个人只负责制陶流程中的一道专门工序:

汽车组装流水线

3.1 流水线的基本概念

3.1.1 什么是流水线

1. 工业生产流水线

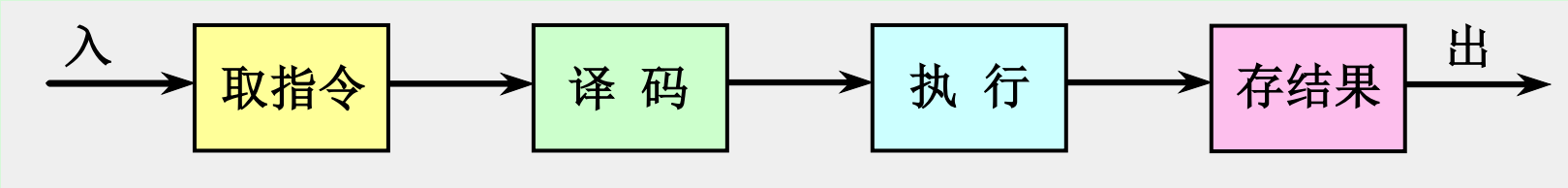
2. 流水线技术
   - 把一个重复的过程分解为若干个子过程，每个子过程由专门的功能部件来实现。
   - 把多个处理过程在时间上错开，依次通过各功能段，这样，每个子过程就可以与其他的子过程并行进行。
3. 流水线中的每个子过程及其功能部件称为流水线的级或段，段与段相互连接形成流水线。流水线的段数称为流水线的深度。

4. 指令流水线

- 把指令的解释过程分解为分析和执行两个子过程，并让这两个子过程分别用独立的分析部件和执行部件来实现。

  理想情况：速度提高一倍

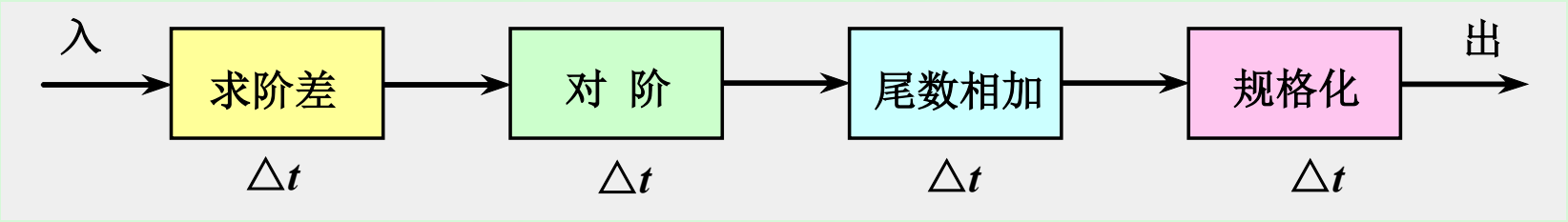
- 4段指令流水线



5. 浮点加法流水线

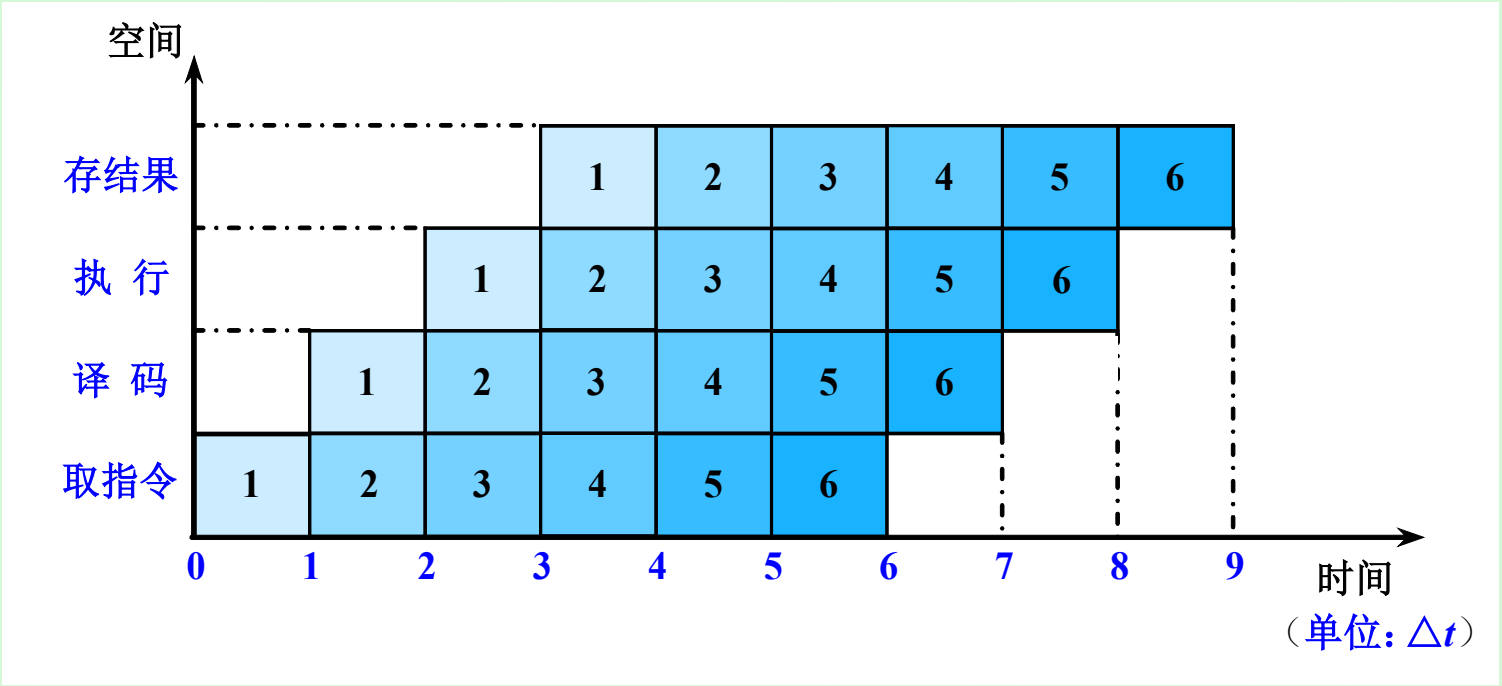
- 把流水线技术应用于运算的执行过程，就形成了运算操作流水线，也称为部件级流水线。
- 把浮点加法的全过程分解为求阶差、对阶、尾数相加、规格化4个子过程。

理想情况：速度提高3倍

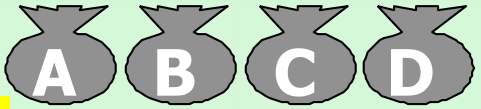


6. 时空图

- 时空图从时间和空间两个方面描述了流水线的工作过程。时空图中，横坐标代表时间，纵坐标代表流水线的各个段。
- 4段指令流水线的时空图



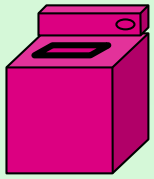
洗衣店的例子



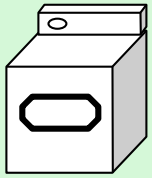
A, B, C, D共四个客户（家庭）的衣服要处理。

需要清洗，脱水，烘干共3台设备的处理

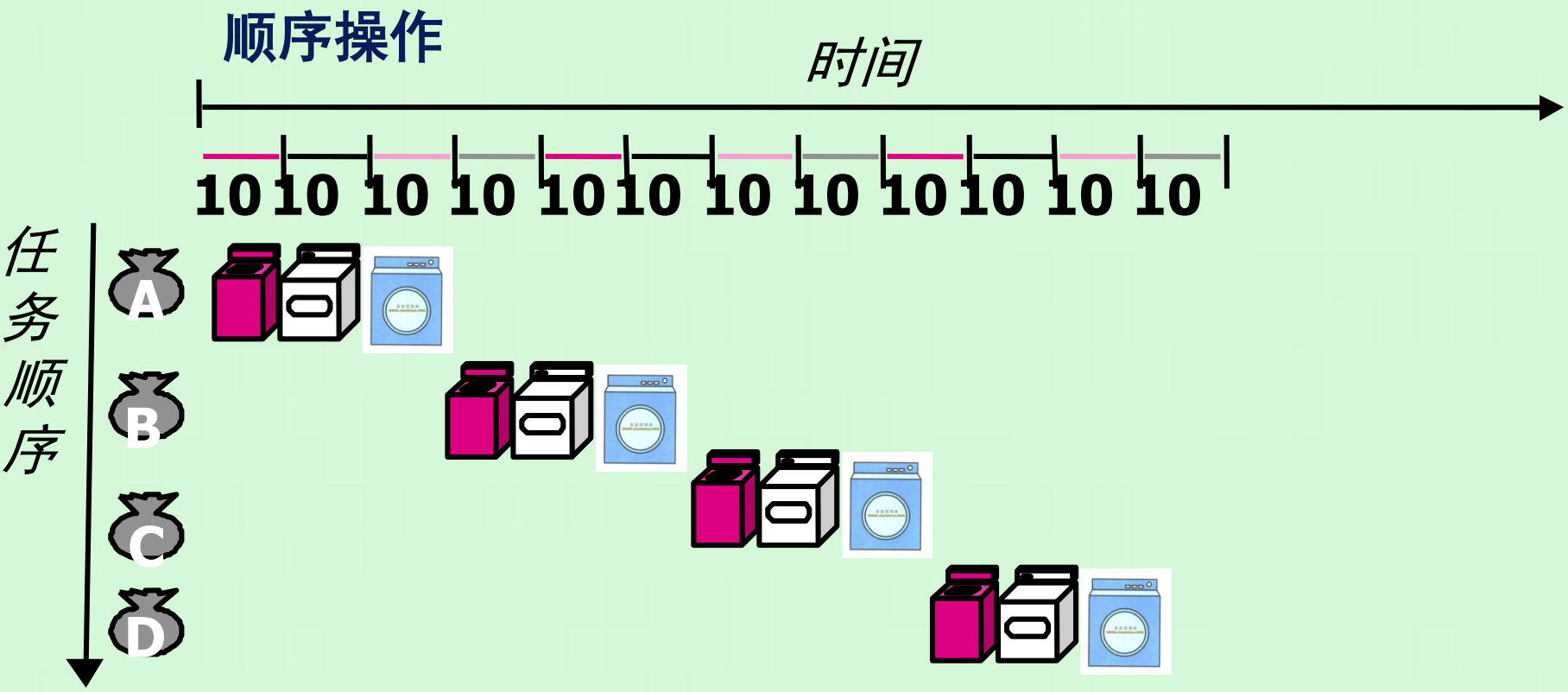
- 清洗要花 10 分钟



- 脱水要用 10 分钟



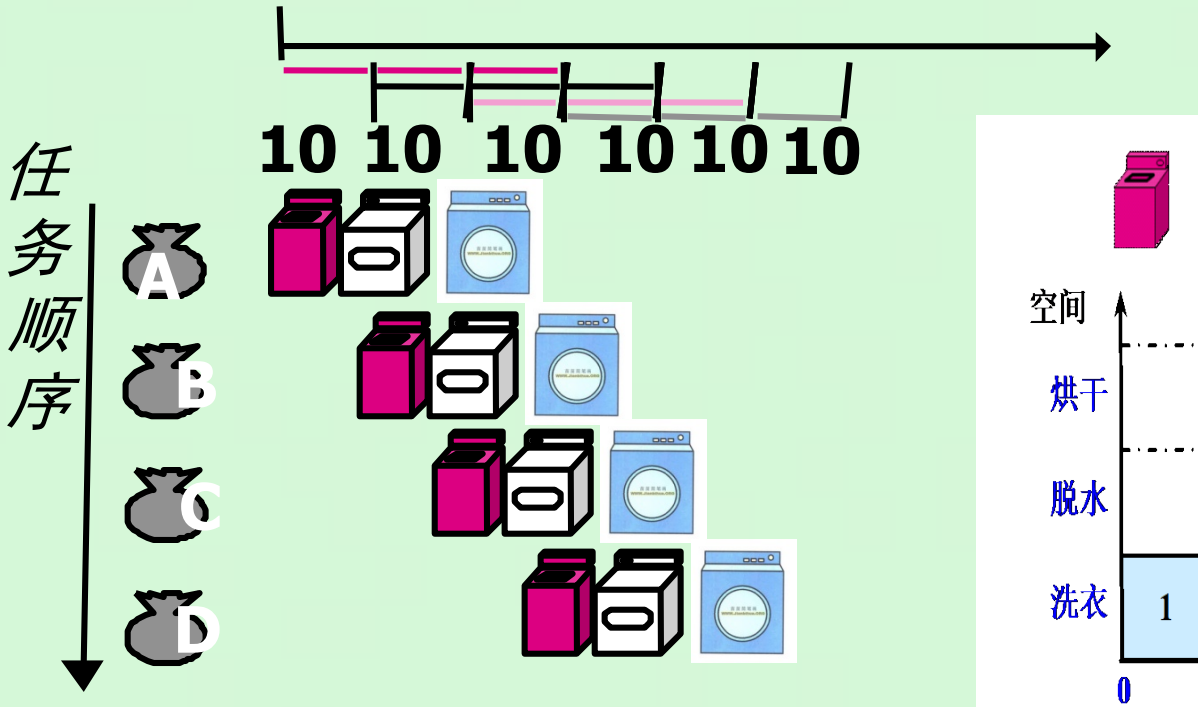
- 烘干也需要 10 分钟

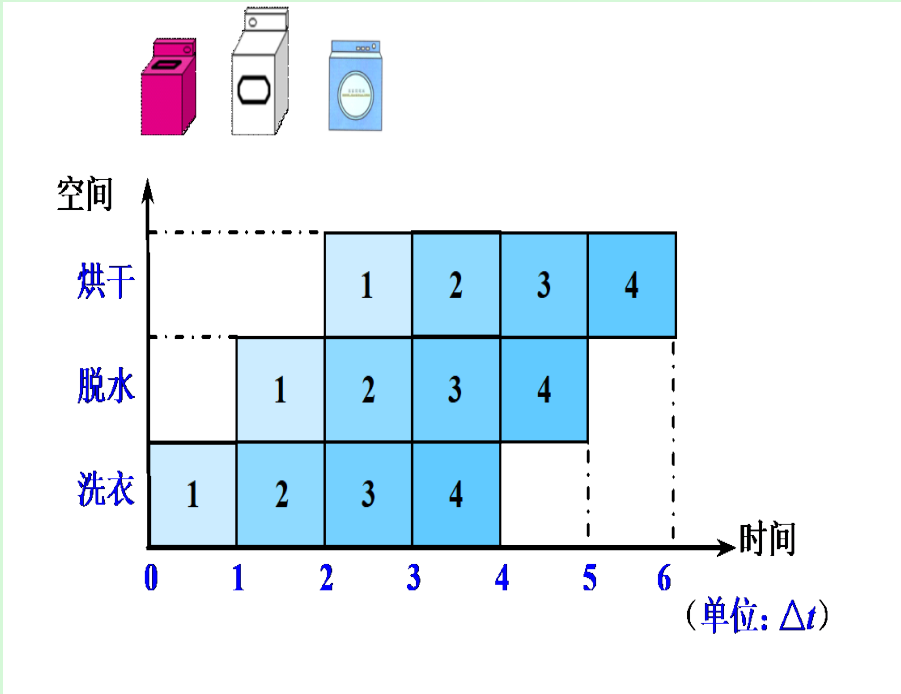


洗4 个人的衣物，顺序操作需要 120 分钟

- 如果使用流水线作业, 将需要多少时间呢?

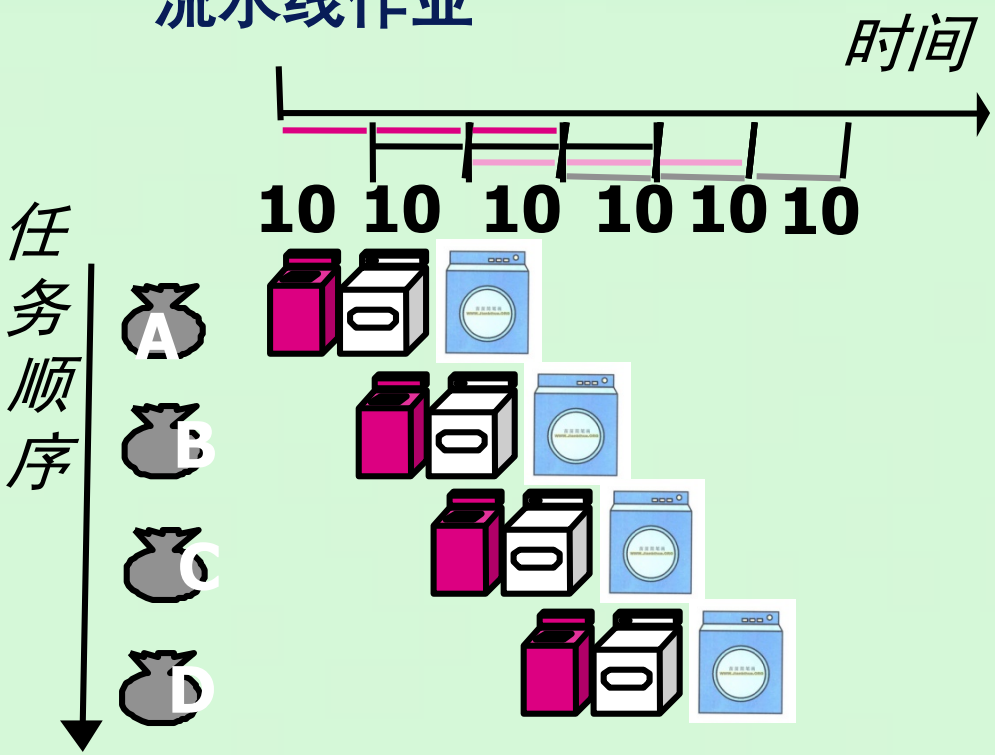
流水线作业





流水线作业洗4个人的衣物只需要 60分钟!

流水线作业



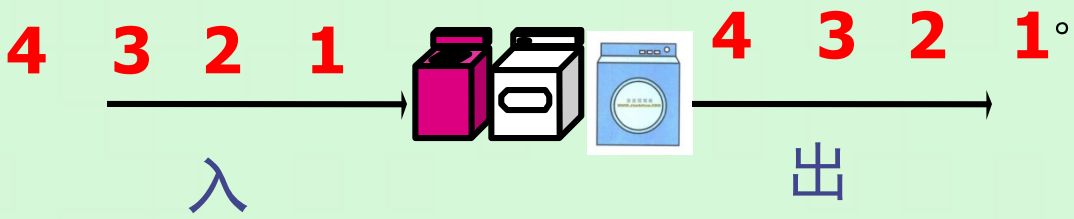
- 流水线无法帮助解决单个任务的延迟，有利于减少整个工作全部时间
- 多个任务同时操作需要不同的资源
- 可能的加速比 = 流水线的段数
- 流水线的速率受速度最慢的流水段的限制
- 流水线各段长度不均会降低加速比

充满流水线所需的时间和排空流水线所需的时间影响加速比

会由于依赖而造成阻塞

7. 流水技术的特点

- 流水线把一个处理过程分解为若干个子过程（段），每个子过程由一个专门的功能部件来实现。



7. 流水技术的特点

- 流水线把一个处理过程分解为若干个子过程（段），每个子过程由一个专门的功能部件来实现。
- 流水线中各段的时间应尽可能相等，否则将引起流水线堵塞、断流。

时间长的段将成为流水线的瓶颈。

时间长⇒瓶颈

7. 流水技术的特点

- 流水线把一个处理过程分解为若干个子过程（段），每个子过程由一个专门的功能部件来实现。
- 流水线中各段的时间应尽可能相等，否则将引起流水线堵塞、断流。

  时间长的段将成为流水线的瓶颈。
- 流水线每一个功能部件的后面都要有一个缓冲寄存器（锁存器），称为流水寄存器。
  - 作用：在相邻的两段之间传送数据，以保证提供后面要用到的数据，并把各段的处理工作相互隔离。

- 流水技术适合于大量重复的时序过程，只有在输入端不断地提供任务，才能充分发挥流水线的效率。

- 流水技术适合于大量重复的时序过程，只有在输入端不断地提供任务，才能充分发挥流水线的效率。
- 流水线需要有通过时间和排空时间。
  - 通过时间：第一个任务从进入流水线到流出结果所需的时间。
  - 排空时间：最后一个任务从进入流水线到流出结果所需的时间。