计算机组成与实现

多周期CPU

高小鹏
北京航空航天大学计算机学院

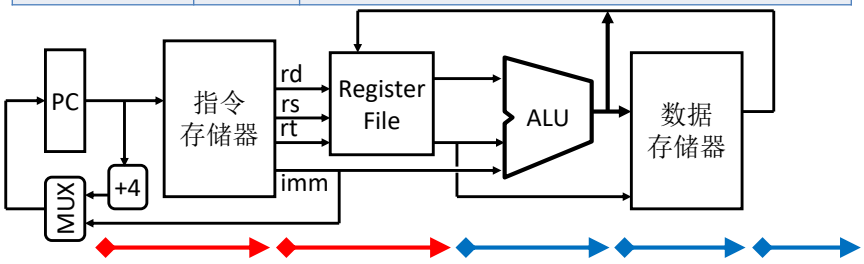
目录

- 单周期的不足
- 破解关键路径的一般性方法
- 多周期数据通路
- 建模多周期数据通路执行过程
- 建模多周期控制器
- 多周期性能分析

回顾：单周期数据通路

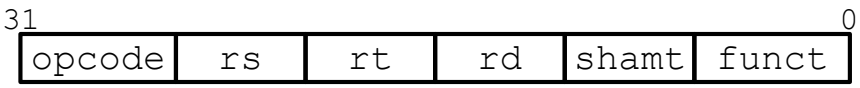
- 从逻辑上：数据通路总共为5个逻辑步骤
- 公共步骤：取指令、译码/读操作数
  - 所有指令均必须经历的2个步骤

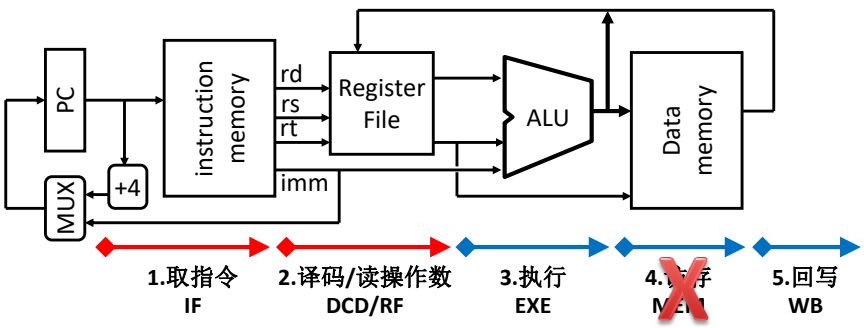
取指令	IF	PC驱动IM读取指令
译码/读操作数	DCD/RF	译码属于控制器范畴；可以与读操作数并行
执行	EXE	ALU完成算数/逻辑运算
访存	MEM	读DM或写DM
回写	WB	ALU计算结果或IM读出数据写入寄存器堆



运算类指令：理想执行过程

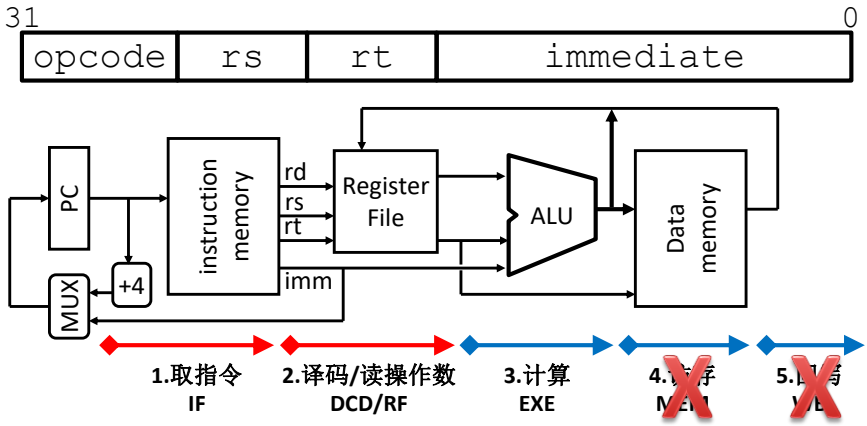
- 指令：ADD、SUB、OR。。。
- 需求：R[rd] ← R[rs] op R[rt]
- 过程：取指、译码/读寄存器、执行、~~访存~~、回写





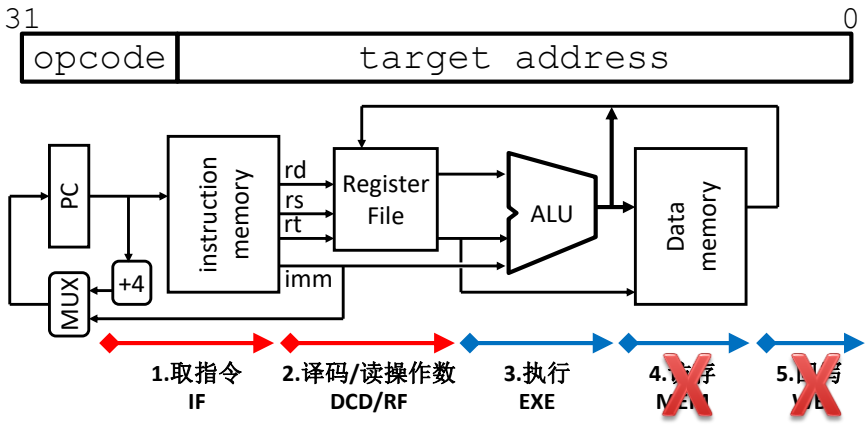
分支类指令：理想执行过程

- 指令：BEQ、。。。。
- 需求：PC ← 条件? PC + Ext(Imm) : PC + 4
- 过程：取指、译码/读寄存器、执行、~~访存~~、~~回写~~



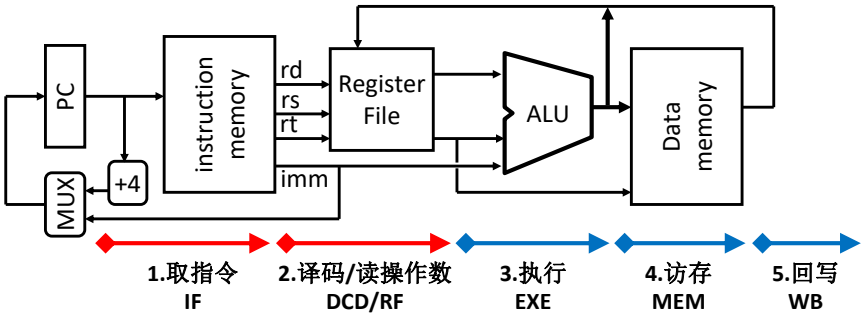
跳转指令：理想执行过程

- 指令：J
- 需求：PC ← PC[31:28] || target_address || 00
- 过程：取指、译码/读寄存器、执行、~~访存~~、~~回写~~



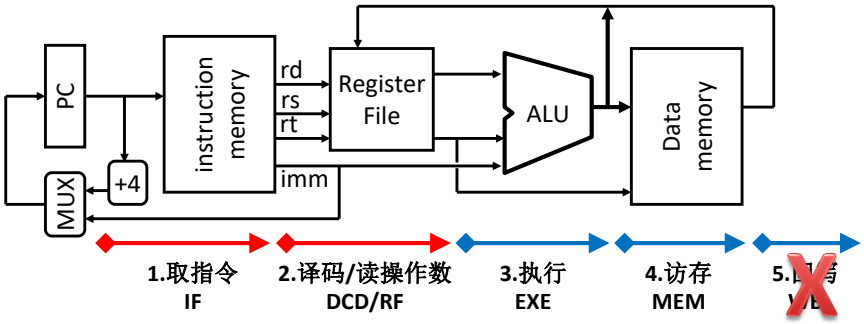
读存储指令：理想执行过程

- 指令：LW
- 需求：RF[rt] ← memory[RF[base] + offset]
- 过程：取指、译码/读寄存器、执行、访存、回写



写存储指令：理想执行过程

- 指令：SW
- 需求：memory[RF[base] + offset] ← RF[rt]
- 过程：取指、译码/读寄存器、执行、访存、~~回写~~



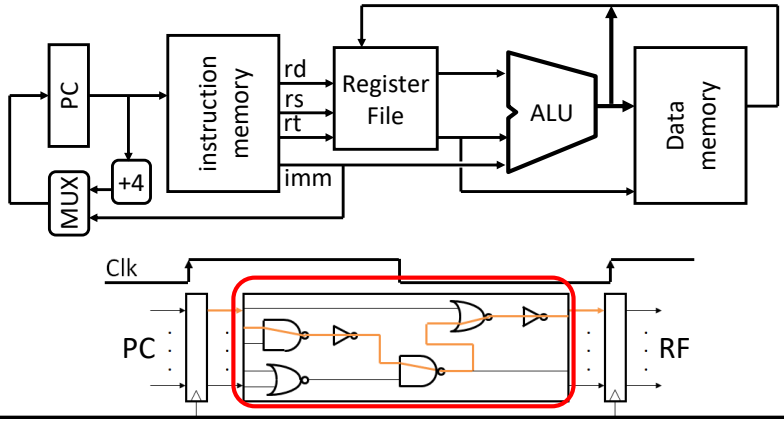
物理执行路径 vs. 理想执行过程

- 2个现象
  - 现象1：不同指令的理想执行过程不同
  - 现象2：所有指令都有前3个阶段
    - 前2阶段完全相同；第3阶段功能有差异
- 启示
  - 不同指令的理想执行过程不同，即理想执行时间不同
  - 不同指令能否具有不同物理执行路径，以对应理想执行过程？

	IF	DCD/RF	EXE	MEM	WB
计算	✓	✓	✓		✓
分支	✓	✓	✓		
跳转	✓	✓	✓		?
读存储	✓	✓	✓	✓	✓
写存储	✓	✓	✓	✓	

单周期数据通路的缺陷

- 模型：PC → 组合逻辑 → 寄存器堆
  - 单一组合逻辑实现了全部5个阶段的逻辑功能
  - 必然存在关键路径，且关键路径导致每条指令延迟均相同
- 结论：无法利用不同指令具有不同执行需求的潜在特性



目录

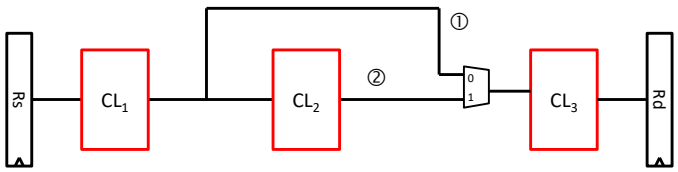
- 单周期的不足
- 破解关键路径的一般性方法
- 建模多周期数据通路
- 建模多周期数据通路执行过程
- 建模多周期控制器
- 多周期性能分析

在关键路径上均匀插入寄存器

- 造成单周期频率低的根本原因在于存在明显的关键路径
- 破解关键路径是时序电路优化的核心内容之一，基本思路是在关键路径上插入一系列寄存器
  - 插入的寄存器将关键路径分割为若干小的组合逻辑
  - 假设分割的比较均匀，则各组合逻辑延迟都比较小
  - 由此就可以大幅度提高系统的时钟频率

在关键路径上均匀插入寄存器

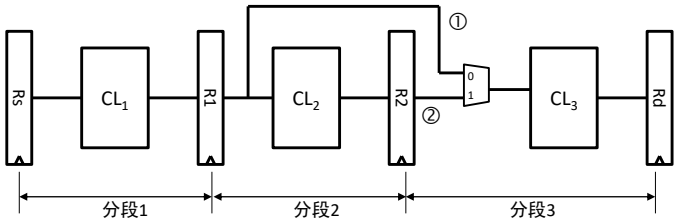
- 改造前：Rs与Rd之间的所有组合逻辑都在关键路径上



$$T = T_{CL1} + T_{CL2} + T_{CL3} = 3T$$

$$f = 1/3T$$

- 改造后：插入R1和R2后，任意两个寄存器之间的延迟都为T



$$f = 1/T$$

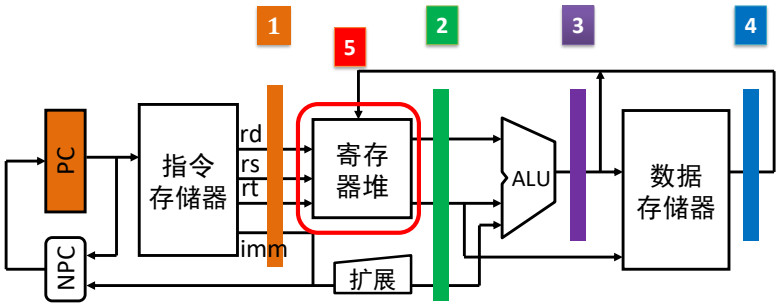
分段1	$\{R_s, R_1\}$
分段2	$\{R_1, R_2\}$
分段3	$\{R_2, R_d\}$ $\{R_1, R_d\}$

※假设：CL1、CL2、CL3延迟均为T；忽略MUX延迟

目录

多周期数据通路构思

- 单周期的单一路径被物理切分为多段路径
  - 在数据通路上插入多个寄存器
  - 单一组合逻辑被切分为多段组合逻辑
  - 单一关键路径的大延迟变为多个分段路径的小延迟



计算机组成与实现

需要的寄存器

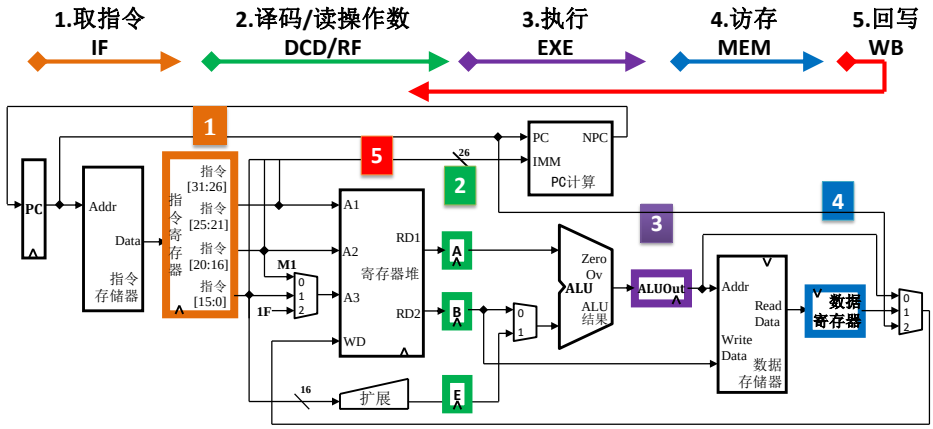
- 每个功能部件后面都插入寄存器
  - RF和DM：虽然是时序部件，但在读出操作时表现为组合逻辑

组合逻辑	插入的寄存器	用途
IM	IR	保存指令
RF（读）	A和B	保存2个寄存器值
扩展单元	E	保存32位扩展值
ALU	ALUOut	保存计算结果
DM（读）	DR	保存读出的数据

计算机组成与实现

基础多周期数据通路

- NPC（PC计算）：完成与PC相关的一切计算
  - PC+4；PC+imm16；PC+imm26



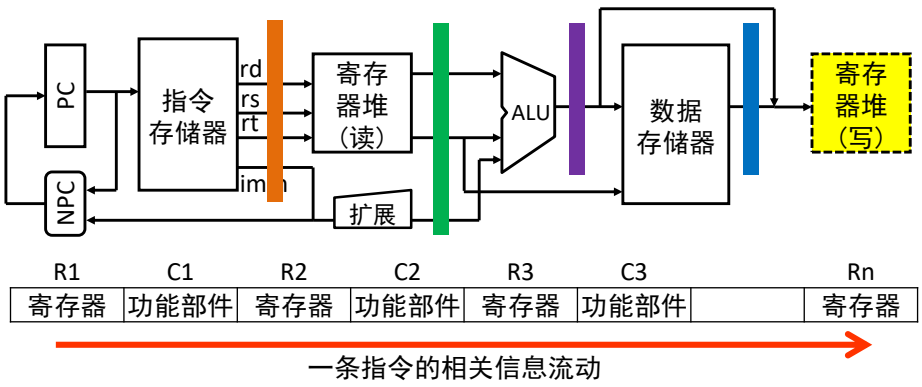
多周期数据通路特点

- 段内执行： 每段1个Cycle，段内模型R→C→R
  - 寄存器输出→组合逻辑→寄存器写入
  - 组合逻辑：如NPC、ALU等
    - RF/DM等：在读操作时，其行为为组合逻辑
- 段间执行：存在逻辑依赖关系
  - 前段没有执行，后段执行无意义
    - 不是不能执行，而是执行结果无意义
    - 例如：指令不读入IR，读操作数就无意义
- 根据指令需求组合分段，构成相应通路
  - 不同指令执行，占用不同的功能单元
  - 很好的映射不同指令的理想执行过程
  - 共有阶段：阶段1(读取指令)、阶段2(读操作数)

※
逻辑依赖是分析数据通路构造的关键

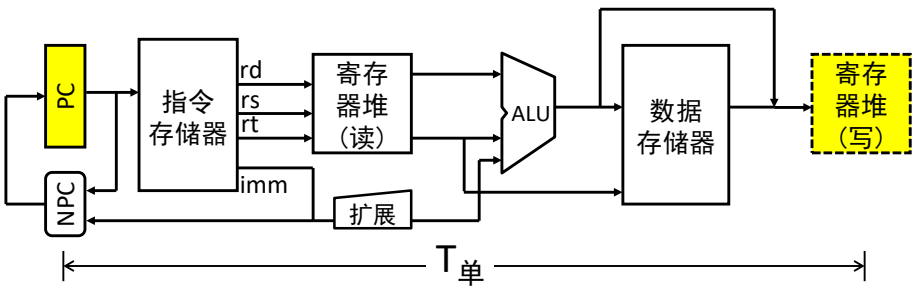
多周期数据通路特点

- 在推导某条指令的具体执行路径时，对于任意一级寄存器R_i，只要前面任意一级的寄存器/组合逻辑包含有需要的信息，均可以将相关信息直接输入到



多周期数据通路的优势

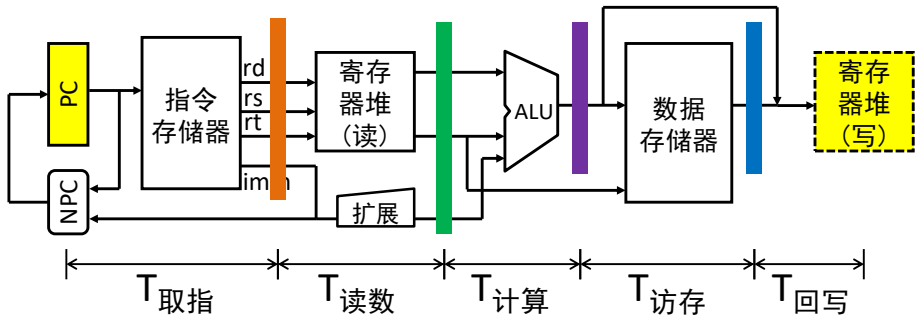
- $T_{单}$ = 关键路径=T_{lw}



- $f_{单}$ = $1/T_{单}$
- 虽然每条指令都只需1个cycle，但时钟频率低
  - CPI = 1

多周期数据通路的优势

- $T_{多}$ = MAX{$T_{取指}$， $T_{读数}$， $T_{计算}$， $T_{访存}$， $T_{回写}$}

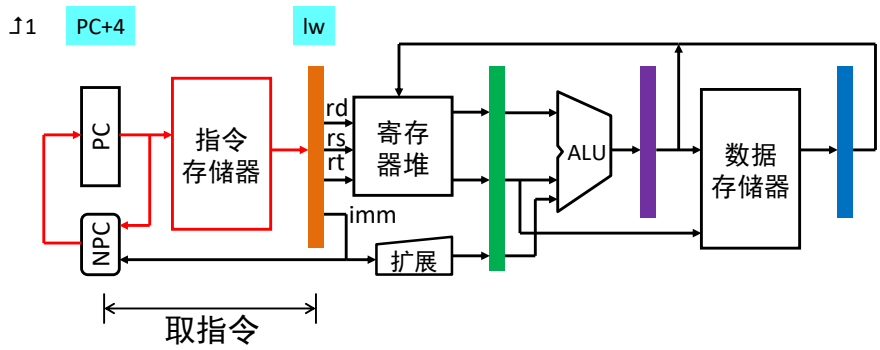


- $T_{多} \approx T_{单}/5$， $f_{多} = 1/T_{多} \approx 5f_{单}$；时钟频率高
- 每个指令需要多个cycle
  - CPI > 1
  - lw：最多，5个cycle，CPI=5

指令执行案例分析：lw

- Cycle 1：取指令
  - PC驱动IM，读出lw，并写入IR
  - PC驱动NPC，计算PC+4，并写入PC

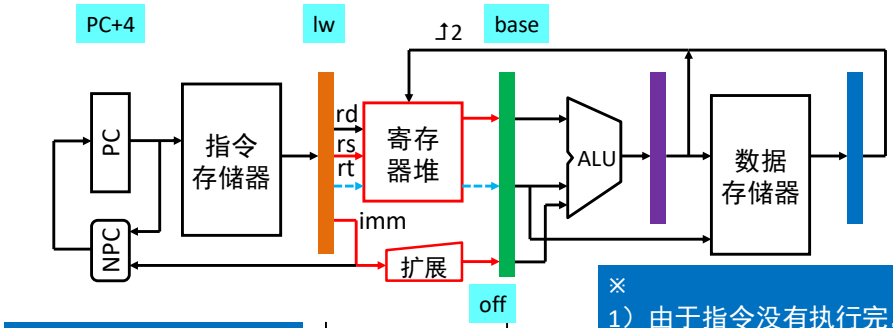
※
取指令：所有指令的公共环节
时序：时钟↑后，IR装入lw，PC指向lw下一条指令



R[rt]←MEM[R[rs]+sign_ext(imm16)]

指令执行案例分析：lw

- Cycle 2：读取寄存器&符号扩展
  - IR[25:21]驱动寄存器堆，读出rs并写入A
  - IR[15:00]驱动扩展单元，符号扩展结果写入E

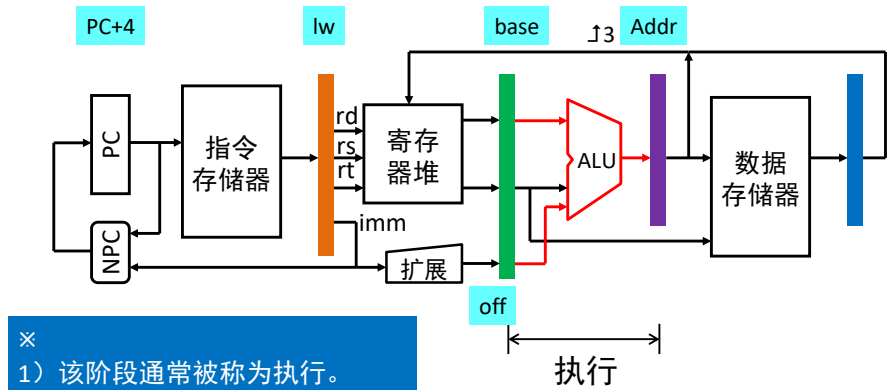


※
对于多数指令来说，这个阶段是准备操作数。同时指令产生有效译码结果，因此通常把该阶段称为译码/读操作数

※
1）由于指令没有执行完，PCWr和IRWr都必须为0，以防止误加载新指令。
2）寄存器堆仍然会读出rt寄存器的值，只是没有意义。

指令执行案例分析：lw

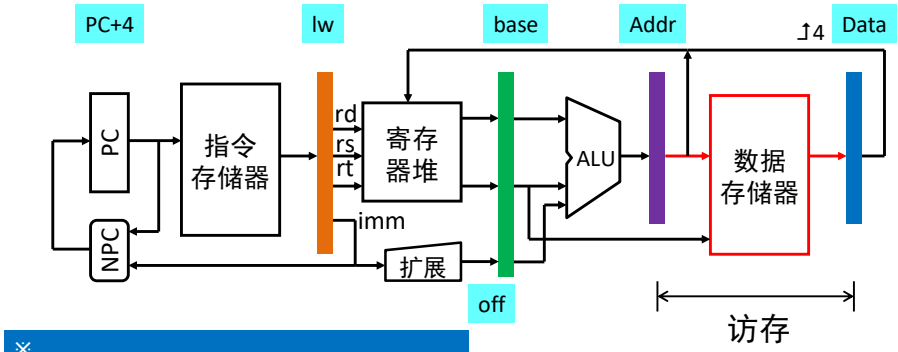
- Cycle 3：ALU计算地址
  - ALU执行加法，计算出的地址写入ALUOut



※
1）该阶段通常被称为执行。
2）与前阶段相同，由于指令没有执行完，PCWr和IRWr都必须为0。

指令执行案例分析：lw

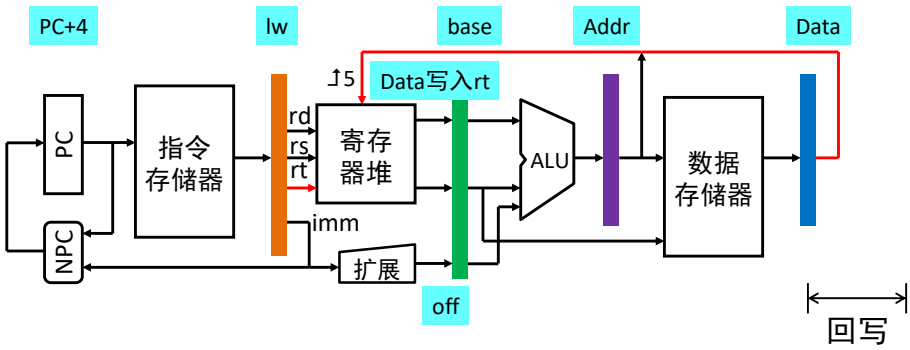
- Cycle 4：读存储器
  - ALUOut驱动DM读出数据，并写入DR



※
对于load类指令，该阶段为读存储器；
对于store类资料，该阶段为写存储器。
该阶段通常被称为访存

指令执行案例分析：lw

- Cycle 5：数据写入寄存器堆
  - DR和IR[21:16]驱动RF，数据写入rt

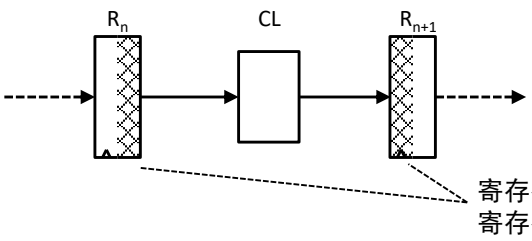


功能部件及其控制信号使用约束

- PC、IR、RF、DM：需要写使能
  - PCWr/IRWr/RFWr/DMWr：只能在特定时间有效！其他时间必须无效！
- A/B/E、ALUOut、DR：不需要写使能
  - 随时可写

多周期分段抽象模型

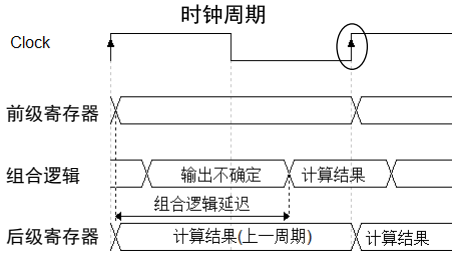
- 任何一个分段都可以抽象为：R_n-CL-R_{n+1}
  - R_n：前级寄存器，为组合逻辑提供输入
  - R_{n+1}：后级寄存器，保存组合逻辑的计算结果
  - CL：组合逻辑，完成计算



- 每个寄存器都扮演读出/写入的双重身份

多周期分段时序模型

- 前一个时钟上升沿
  - 前级寄存器开始输出有效值
  - 经过组合逻辑的计算延迟后，组合逻辑输出正确的计算结果
- 下一个时钟上升沿
  - 后级寄存器更新



※
理解时钟周期与时钟上升沿的关系
1. 周期是时间，上升沿是时刻
2. 组合逻辑在周期内完成计算；计算结果在上升沿时写入寄存器

目录

多周期建模的基本思路

- 多周期建模方法包含三大环节
  - 根据指令的RTL，逐条指令地建模指令的数据通路及控制信号取值
  - 将所有指令的指令级别数据通路综合成完整数据通路
  - 将所有控制信号取值综合成控制器里的表达式
- 由于多周期执行指令需要多个周期，因此单周期建模时采用的多个逻辑“环节”就被替换为物理的“时钟周期”

周期	步骤	语义	RTL	功能部件	控制信号

- 为便于推理，采用R-C-R的基本分段模型表示多周期数据通路

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)

多周期建模的基本思路

- 1、每条指令映射到3～5个cycle
- 2、通过寄存器实现信息传递的相关operation必须部署在不同的cycle
  - 这是由分段数据通路的依赖性决定的
- 3、可以提早执行的operation就“尽早执行”
  - 对于任一电路来说，无论其输出是否被其他电路使用，它都始终在工作
  - 与该电路连接的其他电路只是根据条件决定是否采用其输出值而已
  - 因此即使其他电路不需要该电路“尽早执行”所产生的结果，也无所谓

多周期建模的基本思路

- 4、Cycle1：该周期只能是取指令
  - PC驱动IM读出指令，并将指令写入IR
  - 根据“尽早执行”原则，应同时执行PC←PC+4
- 5、Cycle2：译码信号有效，并同时产生操作数
  - 1）IR存入指令后，控制器就可以产生有效译码结果
  - 2）IR有效，意味着RF必然输出rs值和rt值并写入A/B
    - 当然，A/B存入的可能不是有意义的值（例如对于jal）
  - 3）IR同时也驱动扩展单元，并将扩展立即数写入E
    - 同理，E存入的可能不是有意义的值（例如对于add）

目录

ADDU：RTL建模（倒推1）

- 最后cycle必是ALUOut回写寄存器
- Cycle -1：回写
  - Op1：ALUOut写入rd寄存器

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	E				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-1	回写	计算结果回写至rd寄存器	RF[rd]←ALUOut	RF	RFWr:1

R[rd]←R[rs]+R[rt]

ADDU：RTL建模（倒推2）

- 计算结果来自ALU，计算的数据来自A/B
- Cycle -2：计算
  - Op1：A/B驱动ALU计算，结果存入ALUOut

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	E				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-2	计算	执行加法，结果存入ALUOut	ALUOut←ALU(A,B)	ALU	ALUOp:ADD

R[rd]←R[rs]+R[rt]

ADDU：RTL建模（倒推3）

- A/B保存的是RF读出的rs和rt
- Cycle -3：读操作数
  - Op1：IR[25:21]驱动RF输出rs，存入A
  - Op2：IR[20:16]驱动RF输出rt，存入B

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	E				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-3	读操作数	2个操作数存入A/B	A←RF[rs]; B←RF[rt]	RF	无需控制

R[rd]←R[rs]+R[rt]

ADDU：RTL建模（倒推4）

- 写入IR的一定是指令
- Cycle -4：取指令
  - Op1：读IM，指令写入IR
  - Op2：NPC计算PC+4，并更新PC

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	E				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-4	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1

R[rd]←R[rs]+R[rt]

ADDU：RTL建模（完整）

- 指令时间：4个周期
- 执行路径：→IR→A/B→ALUOut→RF

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr:1 NPCOp:+4 PCWr:1
2	读操作数	操作数存入A 无符号扩展	A←RF[rs]; B←RF[rt]	RF	
3	执行	执行加法，结果存入ALUOut	ALUOut←ALU(A,B)	ALU	ALUOp:ADD
4	回写	计算结果回写至rt寄存器	RF[rt]←ALUOut	RF	RFWr:1

R[rd]←R[rs]+R[rt]

目录

- 单周期的不足
- 破解关键路径的一般性方法
- 多周期数据通路
- 建模多周期数据通路执行过程
  - Ori
- 建模多周期控制器
- 多周期性能分析

ORI：RTL描述表

- 指令时间：4个周期
- 执行路径：→IR→A/E→ALUOut→RF

Ori的执行与addu/sub非常类似，区别主要在于第2个操作数的来源

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr:1 NPCOp:+4 PCWr:1
2	读操作数	操作数存入A 无符号扩展	A←RF[rs]; E←EXT(IR[15:0])	EXT	EXTOp:UE
3	执行	执行加法，结果存入ALUOut	ALUOut←ALU(A,E)	ALU	ALUOp:OR
4	回写	计算结果回写至rt寄存器	RF[rt]←ALUOut	RF	RFWr:1

R[rt]←R[rs]+zero_ext(imm16)

目录

LW：RTL建模（倒推）

- Cycle -1：DM回写
  - DR→无功能部件→RF

周期	步骤	语义	RTL	功能部件	控制信号
-1	DM回写	DR写入rt寄存器	RF[rt]←DR	RF	RFWr:1

- Cycle -2：读DM
  - ALUOut→DM→DR

周期	步骤	语义	RTL	功能部件	控制信号
-2	读DM	读取DM，数据存储DR	DR←DM[ALUOut]	DM	

R[rt]←MEM[R[rs]+sign_ext(imm16)]

LW：RTL建模（倒推）

- Cycle -3：计算地址
  - A/E→ALU→ALUOut

周期	步骤	语义	RTL	功能部件	控制信号
-3	计算地址	执行加法，结果存入ALUOut	ALUOut←ALU(A,E)	ALU	ALUOp:ADD

- Cycle -4：读操作数
  - IR[25:21]→RF→A；IR[15:0]→RF→E

周期	步骤	语义	RTL	功能部件	控制信号
-4	DCD/RF 读操作数	基地址存入A；偏移符号扩展	A←RF[rs] E←EXT(IR[15:0])	EXT	EXTOp:SE

R[rt]←MEM[R[rs]+sign_ext(imm16)]

LW：RTL建模（倒推）

- Cycle -5：取指令
  - PC→IM→IR；PC→NPC→PC

周期	步骤	语义	RTL	功能部件	控制信号
-5	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1

R[rt]←MEM[R[rs]+sign_ext(imm16)]

LW：RTL描述表

- 指令时间：5个周期
- 执行路径：→IR→A/E→ALUOut→DR→RF

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1; NPCOp:+4; PCWr:1
2	读操作数	基地址存入A； 偏移符号扩展	A←RF[rs] E←EXT(IR[15:0])	EXT	EXTOp:SE
3	计算地址	执行加法，结果存入ALUOut	ALUOut←ALU(A,E)	ALU	ALUOp:ADD
4	读存储器	读取DM，数据存储DR	DR←DM[ALUOut]	DM	
5	回写	DR写入rt寄存器	RF[rt]←DR	RF	RFWr:1

R[rt]←MEM[R[rs]+sign_ext(imm16)]

SW：RTL描述表

- 指令时间：4个周期
- 执行路径：→IR→A/E→ALUOut→DM

周期	步骤	语义	RTL	功能部件	控制信号
1	Fetch 取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1
2	DCD/RF 读操作数	基地址存入A； 偏移符号扩展	A←RF[rs] E←EXT(IR[15:0])	EXT	EXTOp:SE
3	MA 计算地址	执行加法，结果存入ALUOut	ALUOut←ALU(A,E)	ALU	ALUOp:ADD
4	MW 写存储器	rt寄存器写入DM	DM[ALUOut]←RF[rt]	DM	DMWr:1

MEM[R[rs]+sign_ext(imm16)] ← R[rt]

目录

- 单周期的不足
- 破解关键路径的一般性方法
- 多周期数据通路
- 建模多周期数据通路执行过程
  - Beq
- 建模多周期控制器
- 多周期性能分析

BEQ：RTL建模（1）

- Cycle 1：取指令
  - Op1：读IM，指令写入IR
  - Op2：NPC计算PC+4，并更新PC

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	E				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1

```
PC ← (R[rs]==R[rt]) ?
   PC+4+(sign_ext(imm16)||00) :
   PC+4
```


BEQ：RTL建模（2）

- Cycle 2：读操作数
  - Op1：IR[25:21]驱动RF输出rs，存入A
  - Op2：IR[20:16]驱动RF输出rt，存入B

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	E				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
2	读操作数	2个操作数存入A/B	A←RF[rs]; B←RF[rt]	RF	无需控制

```
PC ← (R[rs]==R[rt]) ?
   PC+4+(sign_ext(imm16)||00) :
   PC+4
```

BEQ：RTL建模（3）

- 问题：cycle3可否条件写PC?
  - 取决于：1）分支地址是否就绪；2）Zero是否就绪？
- **分支地址**： NPC于cycle2就完成计算
  - cycle1：beq存入IR；PC+4存入PC
  - cycle2：因为NPCOp/imm已有效，所以可计算
- **Zero**：ALU于cycle3产生有效状态

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	**ALU**	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	E				DM(写)		

- 结论：PC写入的前提条件在cycle3均已产生

```
PC ← (R[rs]==R[rt]) ?
   PC+4+(sign_ext(imm16)||00) :
   PC+4
```


BEQ：RTL建模（3）

- Cycle 3：条件写PC
  - Op1：ALU比较A和B，根据Zero决定是否写PC

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	**ALU**	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	E				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
3	条件写分支地址	执行减法，并根据Zero写PC	Zero←ALU(A,B) PC←Zero? NPC(PC,IR[15:0]) PC	ALU NPC PC	ALUOp: SUB NPC: BNPC **PCWr: Zero**

```
PC ← (R[rs]==R[rt]) ?
   PC+4+(sign_ext(imm16)||00) :
   PC+4
```

BEQ：RTL描述表

- 指令时间： 3个cycle
- 执行路径：→IR→A/B→PC
  - ①PC+4：cycle1；②PC+4+偏移：cycle3

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr:1 NPCOp:+4 PCWr:1
2	读操作数	RS操作数存入A; 32位无符号扩展	A←RF[rs]; B←RF[rt]	RF	
3	计算并写分支地址	执行减法，判断Zero	Zero←ALU(A,B) PC←Zero? NPC(PC,IR[15:0]): PC	ALU NPC PC	ALUOp:SUB NPC:BNPC PCWr:Zero

目录

- 单周期的不足
- 破解关键路径的一般性方法
- 多周期数据通路
- 建模多周期数据通路执行过程
  - Jal
- 建模多周期控制器
- 多周期性能分析

JAL：RTL建模分析

- 更新PC的前提：
  - PCWr： IR存入jal后就可以产生
  - imm26：IR存入jal后就可以产生
- 回写寄存器的前提：
  - RFWr：IR存入jal后就可以产生
  - PC+4：第1个cycle后，PC+4就计算完毕
- 以上分析表明：写寄存器&更新PC，仅仅依赖于IR是否存储了jal

PC ← PC[31:28] || imm26 || 00
R[31] ← PC+4

JAL：RTL描述表

- 指令时间：2个cycle
- 执行路径：→IR→PC/RF
  - jal存入IR后，就可以启动写寄存器和更新PC

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr:1 NPCOp:+4 PCWr:1
2	跳转	计算并保存转移PC; 保存PC	RF[31]←PC PC←NPC(PC,IR[25:0])	RF NPC PC	RFWr:1 NPCOp:JNPC PCWr:1

PC ← PC[31:28] || imm26 || 00
R[31] ← PC+4

目录

LUI：RTL建模（倒推1）

- Cycle -1：回写寄存器
  - Op1：E写入rt寄存器

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	E				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-1	回写	扩展数据回写至rt寄存器	RF[rt]←E	RF	RFWr:1

$R[rt] \leftarrow imm16 \,||\, 0^{16}$

LUI：RTL建模（倒推2）

- Cycle -2：高位扩展
  - Op1：IR[15:0]驱动EXT高位扩展，结果写入E

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	E				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-1	高位扩展	高位扩展，结果写E	E←EXT(IR[15:0])	EXT	EXTOp:HE

R[rt] ← imm16 || 0^{16}

LUI：RTL建模（倒推3）

- Cycle -3：读取指令
  - Op1：读IM，指令写入IR
  - Op2：NPC计算PC+4，并更新PC

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	E				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-3	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1

R[rt] ← imm16 || 0^{16}

LUI ： RTL描述表

- 指令时间：3个cycle
- 执行路径：→IR→E→RF
  - lui存入IR后，就可以启动写寄存器和更新PC

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1
2	高位扩展	高位扩展，结果写E	E←EXT(IR[15:0])	EXT	EXTOp:HE
3	回写	扩展数据回写至rt寄存器	RF[rt]←E	RF	RFWr:1

$R[rt] \leftarrow imm16 \,||\, 0^{16}$

LUI：RTL描述表（4cycle）

- 指令时间：4个cycle
- 执行路径：→IR→A/E→ALUOut→RF
- 借用R型计算
  - 注意$0的用途：可以体会MIPS指令格式定义的巧妙

Lui的第2种思路是借用ALU的运算功能，于是与addu/sub/ori等类似，需要4个时钟周期

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr:1 NPCOp:+4 PCWr:1
2	读操作数	$0存入A 无符号扩展	A←RF[0] E←EXT(IR[15:0])	EXT	EXTOp:HE
3	执行	执行**OR**，结果存入ALUOut	ALUOut←ALU(A,E)	ALU	ALUOp:OR
4	回写	计算结果回写至rt寄存器	RF[rt]←ALUOut	RF	RFWr:1

目录

- 单周期的不足
- 破解关键路径的一般性方法
- 多周期数据通路
- 建模多周期数据通路执行过程
  - 多周期执行的时序
- 综合多周期数据通路
- 综合多周期控制器
- 多周期性能分析

时序分析的目的

- 1、时序是数字电路的理解与设计难点之一
  - 理解不正确，设计/调试就易于出错
- 2、有利于后续状态机的设计与理解
  - 更好的理解寄存器的数据准备与数据写入的关系
- 3、多周期时序复杂度适中，适于讲解时序
  - 变化：PC、IR、A/B、DR、RF、DM会发生变化
  - 稳定：在1条指令内只变化1次(PC除外)

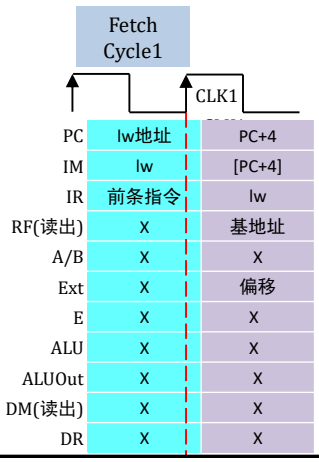
时序分析要点：RTL制导

- RTL制导分析需要哪些必要环节
  - 并非所有环节都需要(多周期的基本特点)
  - 只需关注前序寄存器与后继寄存器间关系
    - 前序寄存器经过组合计算后的值，在时钟沿到来后写入后继寄存器
- ☞注意时钟周期的概念
  - 2个时钟边沿之间的时间

LW时序分析：Cycle1

- Cycle1：取指令(公共周期)
  - 读取IM至IR；同时PC自增4
- X：代表其值无意义
- 边沿后的颜色：代表延迟

周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] E←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,E)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR

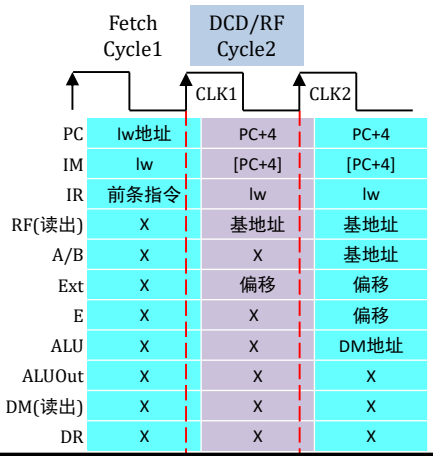


T
Cycle1之后，IR存入了lw。于是在clk1上升沿之后（即cycle2），在IR驱动控下：
1）控制器开始译码产生各种控制信号
2）RF读出2个寄存器值
3）EXT产生32位扩展数

LW时序分析：Cycle2

- Cycle2：读操作数(公共周期)
  - 读操作数，同时开始译码
  - 扩展单元产生32位扩展数

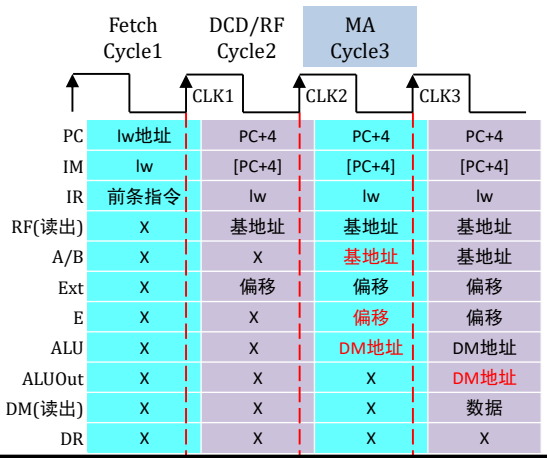
周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] E←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,E)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



LW时序分析：Cycle3

- Cycle3：计算地址
  - 计算地址并存入ALUOut

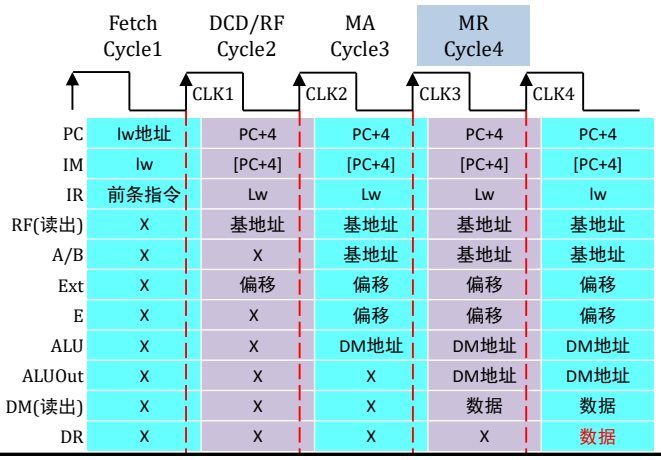
周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] E←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,E)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



LW时序分析：Cycle4

- Cycle4：读存储器
  - ALUOut驱动DM，数据写入DR

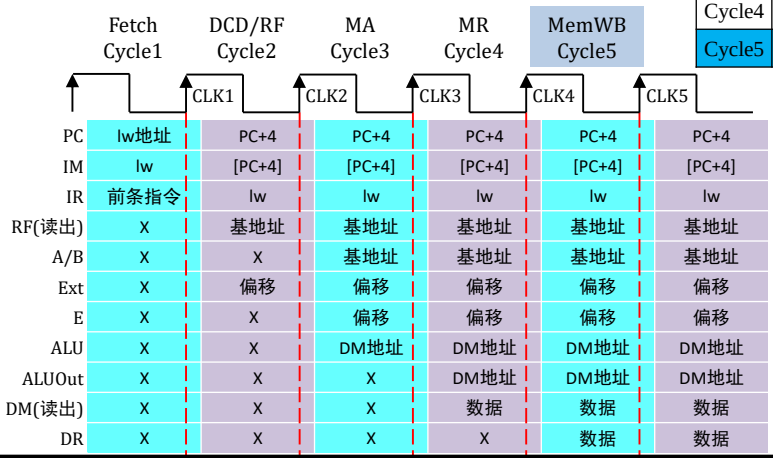
周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] E←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,E)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



LW时序分析：Cycle5

- Cycle5：写寄存器
  - DM读出数据写入RF

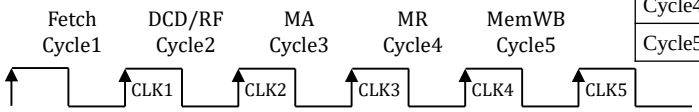
周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] E←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,E)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



LW时序分析：简明版

- 可以不考虑延迟
- 只关注前后依赖关系

周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] E←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,E)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



	Fetch Cycle1	DCD/RF Cycle2	MA Cycle3	MR Cycle4	MemWB Cycle5
PC	lw地址	PC+4	PC+4	PC+4	PC+4	PC+4
IM	lw	[PC+4]	[PC+4]	[PC+4]	[PC+4]	[PC+4]
IR	前条指令	lw	lw	lw	lw	lw
RF(读出)	X	基地址	基地址	基地址	基地址	基地址
A/B	X	X	基地址	基地址	基地址	基地址
Ext	X	偏移	偏移	偏移	偏移	偏移
E	X	X	偏移	偏移	偏移	偏移
ALU	X	X	DM地址	DM地址	DM地址	DM地址
ALUOut	X	X	X	DM地址	DM地址	DM地址
DM(读出)	X	X	X	数据	数据	数据
DR	X	X	X	X	数据	数据

目录

多周期数据通路描述表

- 多周期建模方法与单周期思路完全一致
  - 数据通路表中多了IR、A/B、E、ALUOut、DR
- 对于所有指令，这些寄存器输入源都是固定的
  - IR—IM
  - A/B—RF.RD1/RF.RD2
  - E—EXT；
  - ALUOut—ALU
  - DR—DM

部件	输入	任意指令
NPC	PC	
	Imm	
PC	NPC	
IM	A	
IR		IM.DO
RF	WD	
	A3	
A		RF.RD1
B		RF.RD2
EXT	Imm	
E		EXT
ALU	A	
	B	
ALUOut		ALU.C
DM	A	
	DI	
DR		DM.DO

lw：从RTL到数据通路

- 部分部件的输入源改为寄存器（注意红色信号）

部件	输入	lw
NPC	PC	PC.PC
	Imm	
PC	DI	NPC.NPC
IM	A	PC.DO
IR		IM.DO
RF	WD	DR
	A3	IR[20:16]
A		RF.RD1
B		RF.RD2
EXT	Imm	IR[15:0]
E		EXT
ALU	A	A
	B	EXT
ALUOut		ALU
DM	A	ALUOut
	WD	
DR		DM.RD

多周期

lw
DM.DO
IM[20:16]
IM[15:0]
EXT.Ext
RF.RD1
EXT.Ext
ALU.C

单周期

周期	RTL	控制信号
1	IR←IR[PC]； PC←NPC(PC)	IRWr:1； NPCOp:+4； PCWr:1
2	A←RF[rs] E←EXT(IR[15:0])	EXTOp:SE
3	ALUOut←ALU(A,E)	ALUOp:ADD
4	DR←DM[ALUOut]	
5	RF[rt]←DR	RFWr:1

jal：从RTL到数据通路

部件	输入	jal
NPC	PC	PC.PC
	Imm	IR[25:0]
PC	DI	NPC.NPC
IM	A	PC.DO
IR		IM
RF	WD	PC.DO
	A3	0x1F
A		
B		
EXT	Imm	
E		
ALU	A	
	B	
ALUOut		
DM	A	
	WD	
DR		

在单周期中，RF的回写数据来源于NPC.PC4。
在多周期中，由于PC已经在cycle 1完成了PC+4计算，因此我们不再需要NPC.PC4了。

周期	RTL	控制信号
1	IR←IM[PC]; PC←NPC(PC)	IRWr:1 NPCOp:+4 PCWr:1
2	RF[31]←PC PC←NPC(PC,IR[25:0])	RFWr:1 NPCOp:JNPC PCWr:1

用同样方法建模
其他指令的多周期数据通路

综合形成完整的数据通路

- 单周期数据

部件	输入	addu	ori	beq	lw	sw	jal	综合
NPC	PC	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO
	Imm			IR[15:0]			IR[25:0]	IR[25:0]
PC	DI	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC
IM	A	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO
IR		IM	IM	IM	IM	IM	IM	IM
RF	WD	ALUOut	ALUOut		DR		PC	ALUOut DR PC
	A3	IR[15:11]	IR[20:16]		IR[20:16]		0x1F	IR[15:11] IR[20:16] 0x1F
A		RF.RD1	RF.RD1	RF.RD1	RF.RD1	RF.RD1		RF.RD1
B		RF.RD2		RF.RD2				RF.RD2
EXT	Imm		IR[15:0]		IR[15:0]	IR[15:0]		IR[15:0]
E			EXT.Ext		EXT.Ext	EXT.Ext		EXT.Ext
ALU	A	A	A	A	A	A		A
	B	B	E	B	E	E		B E
ALUOut		ALU.C	ALU.C		ALU.C	ALU.C		ALU.C
DM	A				ALUOut	ALUOut		ALUOut
	WD					B		B
DR					DM.RD			DM.RD

计算机组成与实现

构造MUX

部件	输入	addu	ori	beq	lw	sw	jal	综合	
NPC	PC	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	
	Imm			IR[15:0]			IR[25:0]	IR[25:0]	
PC	DI	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	
IM	A	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	
IR		IM	IM	IM	IM	IM	IM	IM	
RF	WD	ALUOut	ALUOut		DR		PC	0:ALUOut 1:DR 2:PC	MRFWD
	A3	IR[15:11]	IR[20:16]		IR[20:16]		0x1F	0:IR[15:11] 1:IR[20:16] 2:0x1F	MRFA3
A		RF.RD1	RF.RD1	RF.RD1	RF.RD1	RF.RD1		RF.RD1	
B		RF.RD2		RF.RD2				RF.RD2	
EXT	Imm		IR[15:0]		IR[15:0]	IR[15:0]		IR[15:0]	
E			EXT.Ext		EXT.Ext	EXT.Ext		EXT.Ext	
ALU	A	A	A	A	A	A		A	
	B	B	E	B	E	E		0:B 1:E	MALUB
ALUOut		ALU.C	ALU.C		ALU.C	ALU.C		ALU.C	
DM	A				ALUOut	ALUOut		ALUOut	
	WD					B		B	
DR					DM.RD			DM.RD	

计算机组成与实现

部件	输入	addu	ori	beq	lw	sw	jal	综合	
NPC	PC	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	
	Imm			IR[15:0]			IR[25:0]	IR[25:0]	
PC	DI	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	
IM	A	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	
IR		IM	IM	IM	IM	IM	IM	IM	
RF	WD	ALUOut	ALUOut		DR		PC	0:ALUOut 1:DR 2:PC	MRFWD
	A3	IR[15:11]	IR[20:16]		IR[20:16]		0x1F	0:IR[15:11] 1:IR[20:16]	MRFA3
ALUOut		ALU	ALU		ALU	ALU		ALU	
DM	Ad				ALUOut	ALUOut		ALUOut	
	Din					B		B	
DR					DM.DO			DM.DO	
		00	00		01		10	MRFWD	
		00	01		01		10	MRFA3	
		0	1	0	1	1		MALUB	

建立指令与MUX控制信号的取值对应关系

以addu为例，回写数据来自ALUOut，而ALUOut连接至MRFWD的0端口，所以MRFWD的控制信号取值应为00

部件	输入	addu	ori	beq	lw	sw	jal	综合	
NPC	PC	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	
	Imm			IR[15:0]			IR[25:0]	IR[25:0]	
PC	DI	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	NPC.NPC	
IM	A	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	PC.DO	
IR		IM	IM	IM	IM	IM	IM	IM	
RF	WD	ALUOut	ALUOut		DR		PC	0:ALUOut 1:DR 2:PC	MRFWD
	A3	IR[15:11]	IR[20:16]		IR[20:16]		0x1F	0:IR[15:11] 1:IR[20:16]	MRFA3
ALUOut		ALU	ALU		ALU	ALU		ALU	
DM	Ad				ALUOut	ALUOut		ALUOut	
	Din					B		B	
DR					DM.DO			DM.DO	
		`ALU	`ALU		`MEM		`PC	MRFWD	
		`RD	`RT		`RT		`R31	MRFA3	
		`B	`EXT	`B	`EXT	`EXT		MALUB	

建立指令与MUX控制信号的取值对应关系

建议在Verilog中使用宏，而不是直接用00,01,10

例如：`ALU~2'b00，`MEM~2'b01，`PC~2'b10

目录

- 单周期的不足
- 破解关键路径的一般性方法
- 多周期数据通路
- 建模多周期数据通路执行过程
- 综合多周期数据通路
- 综合多周期控制器
- 多周期性能分析

回顾单周期控制器设计方法

- 在单周期中，控制信号仅与指令相关，因此控制信号取值如下：

指令	RFWr	DMWr
addu	1	0
subu	1	0
lw	1	0
sw	0	1

※
下面以{addu、subu、lw、sw}为指令集，以RFWr和DMWr这两个控制信号为例讨论如何构造多周期控制器

- 然后，将上述真正表转化为表达式

$$RFWr = addu \cdot 1 + subu \cdot 1 + lw \cdot 1 + sw \cdot 0$$
$$= addu + subu + lw$$

$$DMWr = addu \cdot 0 + subu \cdot 0 + lw \cdot 0 + sw \cdot 1$$
$$= sw$$

多周期控制信号的时间特性

- 与单周期只有值不同，多周期控制信号不仅有值的因素，还有时间因素，即：在正确的周期产生正确的值
- 示例：addu和lw中的RFWr
  - 对于addu，RFWr应在第4周期值1；对于lw，则应在第5周期值为1

Addu

周期	周期级RTL	控制信号
1	IR←IM[PC]; PC←NPC(PC)	IRWr:1 NPCOp:+4 PCWr:1
2	A←RF[rs]; B←RF[rt]	
3	ALUOut←ALU(A,B)	ALUOp:ADD
4	RF[rt]←ALUOut	RFWr:1

Lw

周期	周期级RTL	控制信号
1	IR←IM[PC]; PC←NPC(PC)	IRWr:1 NPCOp:+4 PCWr:1
2	A←RF[rs] E←EXT(IR[15:0])	EXTOp:SE
3	ALUOut←ALU(A,E)	ALUOp:ADD
4	DR←DM[ALUOut]	
5	RF[rt]←DR	RFWr:1

多周期控制信号的时间特性

- 用 {时间:值}的形式来表示多周期控制信号取值
  - 时间：用周期代表，如T4/T5分别代表第4/第5个周期
  - 值：表达方式与含义均与单周期相同

指令	RFWr
addu	T4:1
lw	T5:1

Addu

周期	周期级RTL	控制信号
1	IR←IM[PC]; PC←NPC(PC)	IRWr:1 NPCOp:+4 PCWr:1
2	A←RF[rs]; B←RF[rt]	
3	ALUOut←ALU(A,B)	ALUOp:ADD
4	RF[rt]←ALUOut	RFWr:1

Lw

周期	周期级RTL	控制信号
1	IR←IM[PC]; PC←NPC(PC)	IRWr:1 NPCOp:+4 PCWr:1
2	A←RF[rs] E←EXT(IR[15:0])	EXTOp:SE
3	ALUOut←ALU(A,E)	ALUOp:ADD
4	DR←DM[ALUOut]	
5	RF[rt]←DR	RFWr:1

RFWr有效的时钟周期

多周期控制信号表达式构造的基本思路

- 显然，时间与值是同步关系，即两者同时发生
- 因此，时间与值在表达式上就表现为“与”
- 示例1：RFWr表达式构造方法

指令	RFWr
addu	T4:1
lw	T5:1

$$RFWr = addu \cdot \mathrm{T4} \cdot 1 + lw \cdot \mathrm{T5} \cdot 1$$
$$= addu \cdot \mathrm{T4} \quad + lw \cdot \mathrm{T5}$$

- 后面再讨论如何产生和表示T4、T5这样的时钟周期

多周期控制信号表达式构造的基本思路

- 示例2：lw的IRWr和EXTOp
  - IRWr：{T1:1}
    - IRWr在第1个周期内有效
  - EXTOp：{T2:SE}
    - EXT在第2个周期内执行符号扩展
    - SE代表符号扩展功能的编码
- 同理建立其他控制信号取值

周期	周期级RTL	控制信号
1	IR←IM[PC]; PC←NPC(PC)	IRWr:1 NPCOp:+4 PCWr:1
2	A←RF[rs] E←EXT(IR[15:0])	EXTOp:SE
3	ALUOut←ALU(A,E)	ALUOp:ADD
4	DR←DM[ALUOut]	
5	RF[rt]←DR	RFWr:1

lw多周期建模（简版）

指令	NPCOp	PCWr	IRWr	RFWr	EXTOp	ALUOp	DMWr	MRFA3Sel	MRFWDSel	MALUBSel
lw	T1:+4	T1:1	T1:1	T5:1	T2:SE	T3:ADD		?	?	?

多周期控制信号表达式构造的基本思路

- 示例：lw的MRFWD控制信号
  - 假设MRFWD控制信号为MRFWDSel
  - MRFWD的功能是选择回写数据来源
  - 由于回写数据只在回写周期才被使用，因此在T5周期才需要考虑其取值
  - 回写数据来自DR，而DR被连接到MUX的01端口
  - 综上，控制信号取值应为{T5:01}
- 类似的，可以建立MRFA3和MALUB的控制信号取值

部件	输入	lw	综合
⋮	⋮	⋮	⋮
RF	WD	DR	0:ALUOut 1:DR 2:PC
	A3	IR[20:16]	0:IR[15:11] 1:IR[20:16] 2:0x1F
⋮	⋮	⋮	⋮
ALU	B	E	0:B 1:E
⋮	⋮	⋮	⋮

指令	NPCOp	PCWr	IRWr	RFWr	EXTOp	ALUOp	DMWr	MRFA3Sel	MRFWDSel	MALUBSel
lw	T1:+4	T1:1	T1:1	T5:1	T2:SE	T3:ADD		T5:01	T5:01	T3:1

合成

- 将所有指令的控制信号建模结果汇聚在一起

指令	NPCOp	PCWr	IRWr	RFWr	EXTOp	ALUOp	DMWr	MRFA3Sel	MRFWDSel	MALUBSel
lw	T1:+4	T1:1	T1:1	T5:1	T2:SE	T3:ADD		T5:RT	T5:DR	T3:E32
sw	T1:+4	T1:1	T1:1		T2:SE	T3:ADD	T4:1			T3:E32
addu	T1:+4	T1:1	T1:1	T4:1		T3:ADD		T4:RD	T4:AR	T3:B
subu	T1:+4	T1:1	T1:1	T4:1		T3:SUB		T4:RD	T4:AR	T3:B
ori	T1:+4	T1:1	T1:1	T4:1	T2:UE	T3:OR		T4:RD	T4:AR	T3:E32
lui	T1:+4	T1:1	T1:1	T4:1	T2:HE	T3:ADD		T4:RD	T4:AR	T3:E32
beq	T1:+4 T3:BNPC	T1:1 T3:Zero	T1:1			T3:SUB				T3:B
jal	T1:+4 T2:JNPC	T1:1 T2:1	T1:1	T2:1				T2:+31	T2:PC4	

※表中{时间:值}的值不用编码值而是采用宏，在实际开发中有助于理解和开发

生成表达式

- 采用前述方法生成各个控制信号的表达式
- 示例：RFWr

指令	NPCOp	PCWr	IRWr	RFWr	EXTOp	ALUOp	DMWr	MRFA3Sel	MRFWDSel	MALUBSel
lw	T1:+4	T1:1	T1:1	T5:1	T2:SE	T3:ADD		T5:RT	T5:DR	T3:E32
sw	T1:+4	T1:1	T1:1		T2:SE	T3:ADD	T4:1			T3:E32
addu	T1:+4	T1:1	T1:1	T4:1		T3:ADD		T4:RD	T4:AR	T3:B
subu	T1:+4	T1:1	T1:1	T4:1		T3:SUB		T4:RD	T4:AR	T3:B
ori	T1:+4	T1:1	T1:1	T4:1	T2:UE	T3:OR		T4:RD	T4:AR	T3:E32
lui	T1:+4	T1:1	T1:1	T4:1	T2:HE	T3:ADD		T4:RD	T4:AR	T3:E32
beq	T1:+4 T3:BNPC	T1:1 T3:Zero	T1:1			T3:SUB				T3:B
jal	T1:+4 T2:JNPC	T1:1 T2:1	T1:1	T2:1				T2:+31	T2:PC4	

$$RFWr = lw \cdot T5 \cdot 1 + addu \cdot T4 \cdot 1 + subu \cdot T4 \cdot 1 + ori \cdot T4 \cdot 1 + lui \cdot T4 \cdot 1 + jal \cdot T2 \cdot 1$$
$$= lw \cdot T5 + addu \cdot T4 + subu \cdot T4 + ori \cdot T4 + lui \cdot T4 + jal \cdot T2$$

生成表达式

- 示例：PCWr
  - 如果一个表格中有多个项，它们也是“OR”的关系

指令	NPCOp	PCWr	IRWr	RFWr	EXTOp	ALUOp	DMWr	MRFA3Sel	MRFWDSel	MALUBSel
lw	T1:+4	T1:1	T1:1	T5:1	T2:SE	T3:ADD		T5:RT	T5:DR	T3:E32
sw	T1:+4	T1:1	T1:1		T2:SE	T3:ADD	T4:1			T3:E32
addu	T1:+4	T1:1	T1:1	T4:1		T3:ADD		T4:RD	T4:AR	T3:B
subu	T1:+4	T1:1	T1:1	T4:1		T3:SUB		T4:RD	T4:AR	T3:B
ori	T1:+4	T1:1	T1:1	T4:1	T2:UE	T3:OR		T4:RD	T4:AR	T3:E32
lui	T1:+4	T1:1	T1:1	T4:1	T2:HE	T3:ADD		T4:RD	T4:AR	T3:E32
beq	T1:+4 T3:BNPC	T1:1 T3:Zero	T1:1			T3:SUB				T3:B
jal	T1:+4 T2:JNPC	T1:1 T2:1	T1:1	T2:1				T2:+31	T2:PC4	

$$PCWr = lw \cdot T1 \cdot 1 + sw \cdot T1 \cdot 1 + addu \cdot T1 \cdot 1 + subu \cdot T1 \cdot 1 + ori \cdot T1 \cdot 1 + lui \cdot T1 \cdot 1 + beq \cdot T1 \cdot 1 + beq \cdot T3 \cdot Zero + jal \cdot T1 \cdot 1 + jal \cdot T2 \cdot 1$$
$$= (lw+sw+addu+subu+ori+lui+beq+jal) \cdot T1 + beq \cdot T3 \cdot Zero + jal \cdot T2$$

生成表达式

- 示例：PCWr
  - 由于所有指令的T1周期都有效，因此T1项的指令变量可以被优化掉

指令	NPCOp	PCWr	IRWr	RFWr	EXTOp	ALUOp	DMWr	MRFA3Sel	MRFWDSel	MALUBSel
lw	T1:+4	T1:1	T1:1	T5:1	T2:SE	T3:ADD		T5:RT	T5:DR	T3:E32
sw	T1:+4	T1:1	T1:1		T2:SE	T3:ADD	T4:1			T3:E32
addu	T1:+4	T1:1	T1:1	T4:1		T3:ADD		T4:RD	T4:AR	T3:B
subu	T1:+4	T1:1	T1:1	T4:1		T3:SUB		T4:RD	T4:AR	T3:B
ori	T1:+4	T1:1	T1:1	T4:1	T2:UE	T3:OR		T4:RD	T4:AR	T3:E32
lui	T1:+4	T1:1	T1:1	T4:1	T2:HE	T3:ADD		T4:RD	T4:AR	T3:E32
beq	T1:+4 T3:BNPC	T1:1 T3:Zero	T1:1			T3:SUB				T3:B
jal	T1:+4 T2:JNPC	T1:1 T2:1	T1:1	T2:1				T2:+31	T2:PC4	

```
PCWr = (lw+sw+addu+subu+ori+lui+beq+jal)·T1+beq·T3·Zero+jal·T2
     = T1 + beq·T3·Zero + jal·T2
```


生成表达式

- 示例：ALUOp
  - 对于多位控制信号，可以逐位构造表达式，也可以利用Verilog一次性的构造出所有位的表达式

指令	NPCOp	PCWr	IRWr	RFWr	EXTOp	ALUOp	DMWr	MRFA3Sel	MRFWDSel	MALUBSel
lw	T1:+4	T1:1	T1:1	T5:1	T2:SE	T3:ADD		T5:RT	T5:DR	T3:E32
sw	T1:+4	T1:1	T1:1		T2:SE	T3:ADD	T4:1			T3:E32
addu	T1:+4	T1:1	T1:1	T4:1		T3:ADD		T4:RD	T4:AR	T3:B
subu	T1:+4	T1:1	T1:1	T4:1		T3:SUB		T4:RD	T4:AR	T3:B
ori	T1:+4	T1:1	T1:1	T4:1	T2:UE	T3:OR		T4:RD	T4:AR	T3:E32
lui	T1:+4	T1:1	T1:1	T4:1	T2:HE	T3:ADD		T4:RD	T4:AR	T3:E32
beq	T1:+4 T3:BNPC	T1:1 T3:Zero	T1:1			T3:SUB				T3:B
jal	T1:+4 T2:JNPC	T1:1 T2:1	T1:1	T2:1				T2:+31	T2:PC4	

```
assign ALUOp[1:0] = (lw+sw+addu+lui)·T3·`ADD +
                    (subu+beq)·T3·`SUB +
                    ori·T3·`OR
```

状态机概述

- 现在需要解决如何表示T1、T2等时钟周期
- 我们采用状态机来刻画指令执行需要多个时钟周期这一行为
  - 有限状态机是建模时序电路行为的常用方法
- 为了构造状态机，仅需要知道每条指令执行多少个周期即可
- 状态机构造过程也是逐条构造每条指令对应的状态机，然后再进行整体的综合

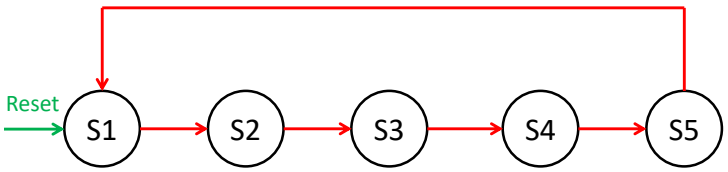
lw：状态机及控制信号

- 状态数：RTL表的周期数决定指令的执行状态数
  - 由于lw执行5个周期，因此需要S1至S5共5个状态
- 状态编码：5个状态需要3位二进制编码
  - S1：000
  - S2：010
  - S3：011
  - S4：100
  - S5：101
- 初始状态：状态机都需要设置一个初始状态，从而确保其执行是确定的
  - 显然，S1被指定为初始状态既简单又合理

周期	控制信号
1	IRWr:1; NPCOp:+4; PCWr:1
2	EXTOp:SE
3	ALUOp:ADD
4	
5	RFWr:1

lw：状态机及控制信号

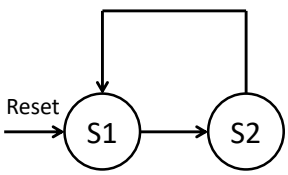
- 无限循环：由于是逐条指令构造状态机，这意味着相当于假设CPU只执行同一条指令，因此状态机是无限循环的，且状态间转移是无条件的
- 初始化：对于状态机来说，一般需要一个复位信号，从而确保状态机能被复位至初始状态



周期	控制信号
1	IRWr:1; NPCOp:+4; PCWr:1
2	EXTOp:SE
3	ALUOp:ADD
4	
5	RFWr:1

jal：状态机及控制信号

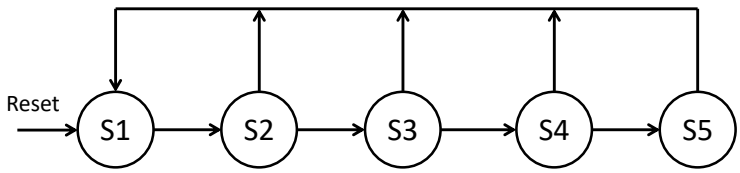
- 由于jal只有2个cycle，因此从S2返回



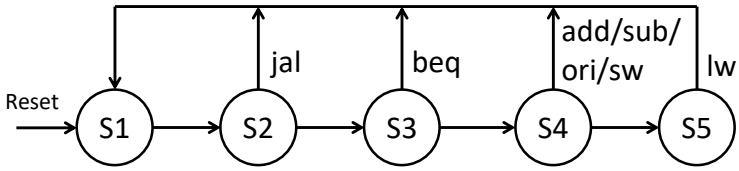
周期	控制信号
1	IRWr:1 NPCOp:+4 PCWr:1
2	RFWr:1 NPCOp:JNPC PCWr:1

合成完整状态机

- 用同样方法建模其他指令的状态机及控制信号，并将它们合成一个完整状态机



- 不同指令的周期数不同，因此在指令执行结束后增加返回初态的判断条件（即指令变量）



生成时钟周期变量

- 时钟周期编号与状态机的状态编号是一一对应的
- 时钟周期变量表达式就是判断当前状态是否是相应的状态编号
- 示例：T1
  - T1就是判断当前状态机变量S[2:0]是否是S1状态
  - 即判断S[2:0]的编码值是否是001

```
状态    parameter S1=3'b001,
定义              S2=3'b010,
                  …

风格1  assign T1 = !S[2] & !S[1] & S[0] ;     差

风格2  assign T1 = (S==3'b001) ;              中

风格3  assign T1 = (S==S1) ;                  好
```

目录

- 单周期的不足
- 破解关键路径的一般性方法
- 多周期数据通路
- 建模多周期数据通路执行过程
- 综合多周期数据通路
- 综合多周期控制器
- 多周期性能分析

平均CPI

- 单周期CPU执行任何指令都仅需要1个时钟周期，因此$CPI_{单周期}=1$
- 在多周期CPU中，不同类型指令具有不同的执行周期数
  - 例如：lw需要5个时钟周期，即$CPI_{lw}=5$
  - 例如：R型计算类指令需要4个时钟周期，即$CPI_{R型}=4$
- 因为各类指令的CPI不同，所以在分析多周期CPU的性能时，必须知道各类指令的执行频度
  - 执行频度是该指令执行次数与总的指令执行数的比值
- 多周期CPU只能按照加权平均方式计算其平均的$CPI_{多周期}$
  - 执行频度就是指令在程序执行过程中的权重

$$CPI_{平均}=\sum CPI_{指令}\times 频度_{指令}$$

CPI~Clock cycle Per Instruction，指令的平均时钟周期数

平均CPI

- 示例：某程序在多周期CPU上共执行了1千亿条指令，其中52%为R型计算类指令，25%为lw，10%为sw，11%为B类指令，2%为J类指令。请计算多周期CPU的CPI平均。
  - 假设：lw需要5个周期，R型计算类指令与sw需要4个周期，B类指令需要3个周期，J类指令需要3个周期
- 解：将各型指令的CPI及其执行频度带入$CPI_{平均}$计算公式

$$
\begin{aligned}
CPI_{平均} = {} & 0.52 \times 4 + && \text{R型} \\
& 0.25 \times 5 + && \text{lw} \\
& 0.1 \times 4 + && \text{sw} \\
& 0.11 \times 3 + && \text{B类} \\
& 0.02 \times 3 && \text{J类} \\
= {} & 4.12
\end{aligned}
$$

- 单周期与多周期执行上述程序谁的性能更好？

性能对比

- 单周期性能：80秒
  - 每条指令执行时间都是800ps，1000亿条指令共需要80秒
- 多周期性能：82.4秒
  - 最长分段是200ps，因此时钟周期长度是200ps

$$T_{多周期} = 1000亿指令 \times {}^{4.12周期}/_{指令} \times {}^{200ps}/_{周期} = 82.4秒$$

指令	读取指令	读寄存器	ALU	数据存取	写寄存器	理想执行时间	实际执行时间
addu	200	100	200		100	600	800
subu	200	100	200		100	600	800
ori	200	100	200		100	600	800
lw	200	100	200	200	100	800	800
sw	200	100	200	200		700	800
beq	200	100	200			500	800
jal	200				100	300	800
jr	200	100				300	800

为什么多周期会败给了单周期？

- 首先，多周期时钟周期由最慢环节决定，因此各分段延迟的均匀程度对性能影响极大。在本例中：
  - 1）R型性能无改善（0ps）
    - 单周期的执行时间为800ps，而多周期仍为800ps（200ps×4）
  - 2）lw性能有损失（+200ps）
    - 单周期的执行时间为800ps，而多周期则上升为1000ps（200ps×5）
  - 3）B类性能有改善（-200ps）
    - 单周期执行时间为800ps，多周期则缩短至600ps（200ps×3）
  - 4）J类性能有改善（-400ps）
    - 单周期执行时间为800ps，多周期则缩短至400ps（200ps×2）
- 在本例中lw占比过大！

为什么多周期会败给了单周期？

- 其次，案例不同则结果可能不同
  - 假设R型占比增加到67%，lw降低至10%，则多周期执行时间为79.4秒

$$
\begin{aligned}
CPI_{平均} = {} & 0.67\times 4 + && \text{R型} \\
& 0.1\times 5 + && \text{lw} \\
& 0.1\times 4 + && \text{sw} \\
& 0.11\times 3 + && \text{B类} \\
& 0.02\times 3 && \text{J类} \\
= {} & 3.97
\end{aligned}
$$

$$
T_{多周期} = 1000亿指令 \times {}^{3.97周期}/_{指令} \times {}^{200ps}/_{周期} = 79.4秒
$$