计算机组成
(2022秋)

计算机组成课程组
（刘旭东、高小鹏、肖利民、栾钟治、万寒）
北京航空航天大学计算机学院中德所
栾钟治

1

习题3——主存储器

- 已发布
  - Spoc平台
- 10月21日截止
  - 23:55
- 在sopc提交
  - 电子版，可手写

2

习题4——汇编语言

- 已发布
  - Spoc平台
- 10月28日截止
  - 23:55
- 在sopc提交
  - 电子版，可手写

3

回顾： CPU的功能与组成

- CPU所需的功能部件
  - 取指令：指令地址部件、指令寄存部件、译码部件
  - 执行指令：执行部件、控制信号逻辑部件
- 数据通路+控制逻辑
- 指令功能的形式化描述：RTL（Register Transfer Language，寄存器传送语言）

4

回顾： 处理器设计的一般方法

- 设计步骤
  1. 分析指令系统需求：包括指令格式、指令类型、每种指令的功能、寻址方式等；
  2. 数据通路构建
     - ① 根据指令需求选择数据通路部件，如PC、ALU、寄存器堆、指令/数据存储器、多路开关等等；
     - ② 根据指令执行流程构建每种类型指令的数据通路；
     - ③ 对所有类型指令执行数据通路综合形成综合数据通路。
  3. 控制器设计
     - ① 确定控制器时序控制方式（单周期、或多周期或其他）
     - ② 根据每种类型指令执行流程，确定该指令执行时各个数据通路部件所需要的控制信号与相应状态、条件；
     - ③ 对控制信号进行综合以得到每个控制信号的逻辑方程；
     - ④ 逻辑电路实现各个控制信号。

5

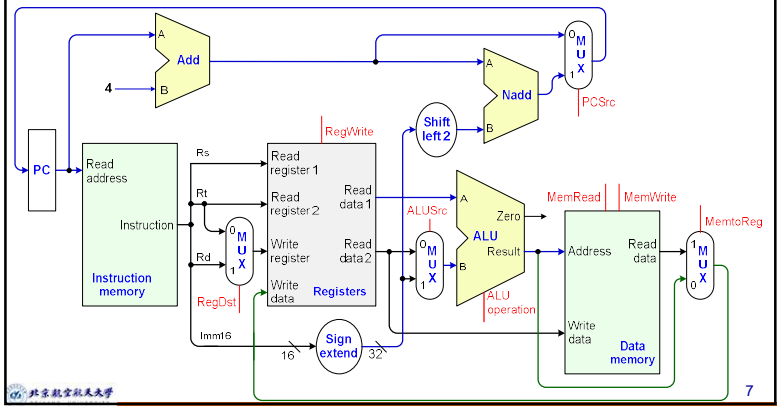
回顾：MIPS模型机指令集

模型机指令集（8条指令）

指令格式	指令	功能	说明
R类型	add rd, rs, rt	R[rd]← R[rs] + R[rt]	加运算：寄存器 rs 和寄存器 rt 相加，结果送寄存器 rd
	sub rd, rs, rt	R[rd]← R[rs] - R[rt]	减运算：寄存器 rs 和寄存器 rt 相减，结果送寄存器 rd
	and rd, rs, rt	R[rd]← R[rs] & R[rt]	与运算：寄存器 rs 和寄存器 rt 按位与，结果送寄存器 rd
	or rd, rs, rt	R[rd]← R[rs] | R[rt]	或运算：寄存器 rs 和寄存器 rt 按位或，结果送寄存器 rd
I类型	lw rt, rs, imm16	Add = R[rs] +Signext(imm16) R[rt] ← M[Add]	取字：寄存器 rs 和立即数 imm16（符号扩展至 32 位）相加得到内存地址，从内存该地址单元读取数据送 rt
	sw rt, rs, imm16	Add = R[rs] +Signext(imm16) M[Add] ← R[rt]	存字：寄存器 rs 和立即数 imm16（符号扩展至 32 位）相加得到内存地址，寄存器 rt 数据写入内存该地址单元
	beq rs, rt, imm16	If (R[rs] – R[rt] =0) then PC ← PC + Signext(imm16)<<2	分支：如果寄存器 rs 与 rt 相等，则转移（imm16 符号扩展至 32 位），否则顺序执行。（取指令后，PC+4）
J类型	j target	PC(31:2)←PC(31:28) || target(25:0)	跳转：当前 PC 的高 4 位与 target（26 位）拼接成 30 位目标地址送 PC（31:2）。（取指令后，PC+4）。

6

指令	Adder		PC	IM Add.	Registers				ALU		DM		Sign-ext	Nadd	
	A	B			Reg1	Reg2	Wreg	Wdata	A	B	Add.	Wdata		A	B
合并	PC	4	Adder | Nadd	PC	Rs	Rt	Rd | Rt	ALU | DM	Rdata1	Rdata2 | Sign-ext	ALU	Rdata2	imm16	Adder	Shift



7

3.2 单周期控制器设计

- 单周期通路所需控制信号
  - ALU控制（ALU Operation）：4位
  - 其他控制信号：7个

ALU 控制

输入		ALU operation	ALU运算
A	B	0000	A & B
A	B	0001	A | B
A	B	0010	A + B
A	B	0110	A – B
A	B	0111	If A=B then Result=1

7个控制信号

控制信号	无效时作用	有效时作用
RegDst	寄存器堆写入端地址选择Rt字段	寄存器堆写入端地址选择 Rd字段
RegWrite	无	把数据写入寄存器堆中对应寄存器
ALUSrc	ALU输入端B选择寄存器堆输出R[rt]	ALU输入端B选择Signext输出
PCSrc	PC输入源选择 PC+4	PC输入选择beq指令的目的地址
MemRead	无	数据存储器DM读数据（输出）
MemWrite	无	数据存储器DM写数据（输入）
MemtoReg	寄存器堆写入端数据来自ALU输出	寄存器堆写入端数据来自DM输出

8

3.2 单周期控制器设计

❖控制器分成两部分：主控单元和ALU控制单元

➢主控单元

- 输入：指令操作码字段 Op（指令31:26位）
- 输出：
  - 7个控制信号
  - ALU控制单元所需的2位输入ALUop

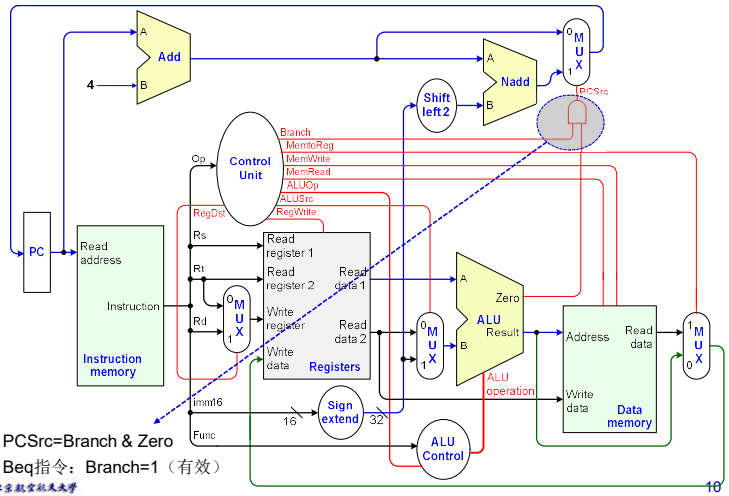
➢ALU控制单元

- 输入：
  - 主控单元生成的ALUop
  - 功能码字段Func（指令5:0位）
- 输出：ALU运算控制信号 ALU operation（4位）

ALUOp指明ALU的运算类型
- 00：访存指令所需加法
- 01：beq指令所需减法
- 10：R型指令功能码决定

Op (31-26)	Rs (25-21)	Rt (20-16)	Rd (15-11)	Shamt (10-6)	Func (5-0)

3.2 单周期控制器设计——模型机数据通路（带控制单元）



PCSrc=Branch & Zero

Beq指令：Branch=1（有效）

3.2 单周期控制器设计

❖主控单元控制信号分析

➢RegDst
- R型指令：RegDst=1，选择Rd
- Lw指令： RegDst=0，选择Rt
- 其他指令：不关心

➢ALUScr
- R型指令：ALUSrc=0，选择寄存器堆的 Read data2 输出
- Lw指令： ALUSrc=1，选择Signext的输出
- Sw指令：ALUSrc=1，选择Signext的输出
- Beq指令（减法运算）： ALUSrc=0，选择 Read data2 输出

➢MemtoReg
- R型指令：MemtoReg=0，选择 ALU 输出
- Lw指令： MemtoReg=1，选择数据存储器DM输出
- 其他指令： 不关心

➢Branch
- Beq指令：Branch=1，此时若Zero=1，PC输入选择加法器Nadd输出（分支指令目的地址），否则选择加法器Add输出（PC+4）
- 其他指令：Branch=0，PC输入选择加法器Add输出（PC+4）

3.2 单周期控制器设计

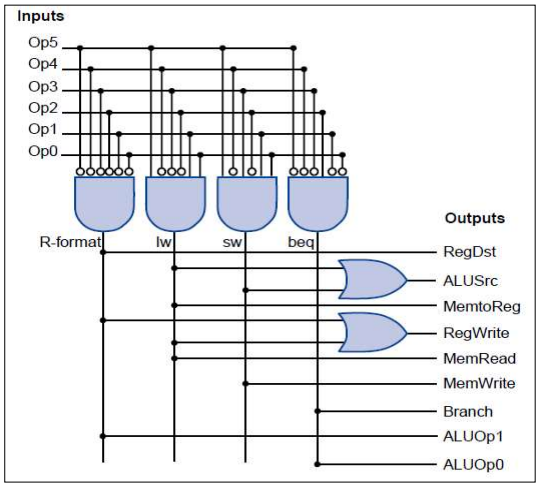
主控单元真值表

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

提供给ALU控制单元

3.2 单周期控制器设计

主控单元
逻辑实现

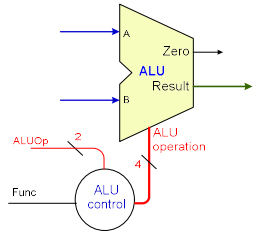


13

3.2 单周期控制器设计

❖ALU控制单元

- 输入：
  - 指令的Func字段（指令5:0位）
  - 由主控单元生成的 ALUOp
- ALUOp指明ALU的运算类型
  - 00：访存指令所需的加法
  - 01：beq指令所需的减法
  - 10：R型指令功能码字段决定



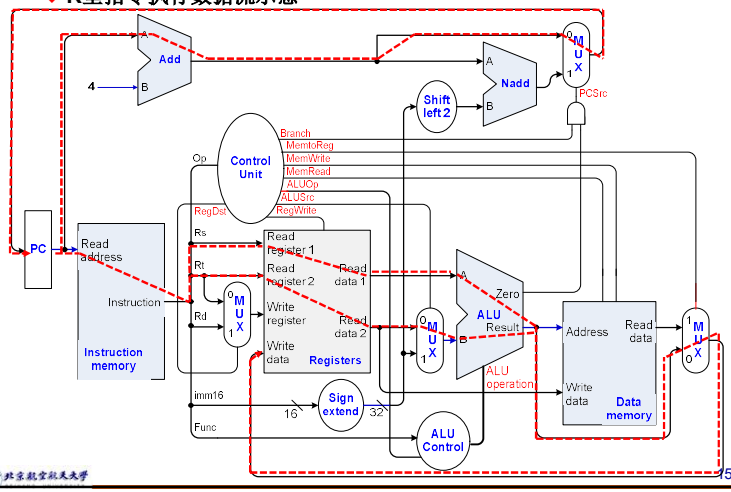
指令	Func字段	ALUop	ALU运算类型	ALU operation
Lw	XXXXXX	00	加	0010
Sw	XXXXXX	00	加	0010
Beq	XXXXXX	01	减	0110
Add	100 000	10	加	0010
Sub	100 010	10	减	0110
And	100 100	10	与	0000
Or	100 101	10	或	0001

ALU控制单元真值表

14

3.2 单周期控制器设计

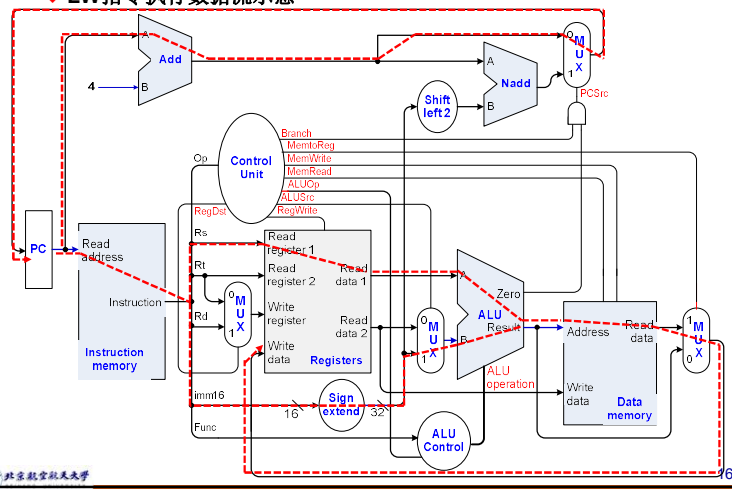
❖ R型指令执行数据流示意



15

3.2 单周期控制器设计

❖ LW指令执行数据流示意



16

MIPS模型机指令集

模型机指令集（8条指令）

指令格式	指令	功能	说明
R类型	add rd, rs, rt	R[rd]← R[rs] + R[rt]	加运算：寄存器 rs 和寄存器 rt 相加，结果送寄存器 rd
	sub rd, rs, rt	R[rd]← R[rs] - R[rt]	减运算：寄存器 rs 和寄存器 rt 相减，结果送寄存器 rd
	and rd, rs, rt	R[rd]← R[rs] & R[rt]	与运算：寄存器 rs 和寄存器 rt 按位与，结果送寄存器 rd
	or rd, rs, rt	R[rd]← R[rs] | R[rt]	或运算：寄存器 rs 和寄存器 rt 按位或，结果送寄存器 rd
I类型	lw rt, rs, imm16	Add = R[rs] +Signext(imm16) R[rt] ← M[Add]	取字：寄存器 rs 和立即数 imm16（符号扩展至 32 位）相加得到内存地址，从内存该地址单元读取数据送 rt
	sw rt, rs, imm16	Add = R[rs] +Signext(imm16) M[Add] ← R[rt]	存字：寄存器 rs 和立即数 imm16（符号扩展至 32 位）相加得到内存地址，寄存器 rt 数据写入内存该地址单元
	beq rs, rt, imm16	If (R[rs] – R[rt] =0) then PC ← PC + Signext(imm16)<<2	分支：如果寄存器 rs 与 rt 相等，则转移（imm16 符号扩展至 32 位），否则顺序执行。（取指令后，PC+4）
J类型	j target	PC(31:2)←PC(31:28) || target(25:0)	跳转：当前 PC 的高 4 位与 target（26 位）拼接成 30 位目标地址送 PC（31:2）。（取指令后，PC+4）。

17

3.2 单周期控制器设计

❖MIPS数据通路（扩展实现跳转指令 j ）

- j add26
- 功能描述
  - PC← PC+4 [31:28] || add26<<2
- 功能部件：Adder，移位器，PC

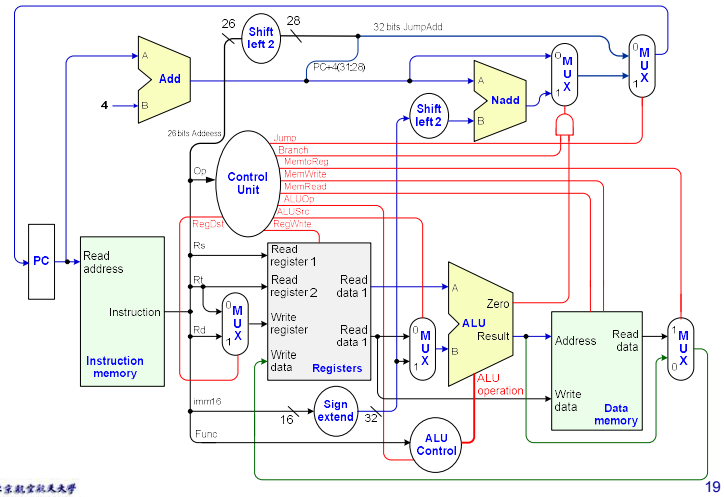
Op (31-26)	26 bit Address (for Jump Instruction) (25-0)

指令	Adder		PC	IM Add.	Registers				ALU		DM		Sign-ext	Nadd	
	A	B			Reg1	Reg2	Wreg	Wdata	A	B	Add.	Wdata			
合并	PC	4	Adder | Nadd	PC	Rs	Rt	Rd | Rt	ALU | DM	Rdata1	Rdata2 | Sign-ext	ALU	Rdata2	imm16	Adder	Shift
J指令	PC	4	Jumpadd	PC											

PC输入数据源又多了一个选择，
增加一个二选一MUX

18

3.2 单周期控制器设计（包含跳转指令的数据通路）



19

小结：数据通路设计的一般性方法

单指令数据通路构造

```
for each 指令
    for each 新增需求
        case 可以合并至已有部件：
            修改部件设计描述、HDL建模：{F', I', O'}
        case 需要新增部件：
            建立新部件设计描述、HDL建模：{F, I, O}
            增加新部件
    for each 部件
        设置输入来源
```

多数据通路综合

```
按垂直方向合并数据通路，并去除相同项
for each 输入来源多于1个的输入端
    部署1个MUX(MUX的输入规模为输入来源数)
    MUX设计定义、HDL建模
```

系统实现

HDL建模：连接所有的部件及所有的MUX

20

部件描述示例——PC

4.1.1. 基本描述

PC 模块的主要功能是将 NPC[31:0]的值保存并输出。PC 的各种取值将根据所执行的指令、外部状态(中断)及处理器控制器的当前状态的不同，由数据通路其他部件生成。

4.1.2. 模块接口

表 4-1 PC 接口信号定义

信号名	方向	描述
Clk	I	MIPS-C 处理器时钟
Reset	I	复位信号
NextPC[31:0]	I	下一个 PC 值
PCWr	I	PC 写使能
PC[31:0]	O	PC 输出

4.1.3. 功能定义

PC 模块的核心是一个寄存器。该寄存器在 PCWr 有效时将 NextPC[31:0]锁存并输出。

表 4-2 PC 功能需求定义

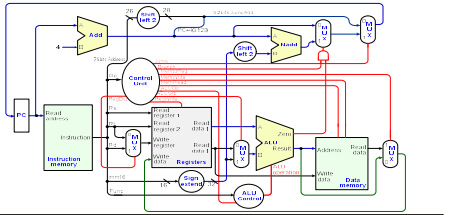
编号	功能名称	功能描述
1	初始化	当 Reset 信号有效后，PC 输出 0xBFC00000。
2	PC 更新	当时钟上升沿到来时，PCWr 有效则将 NPC 写入 PC 内部，并且从 PC 端口输出。

部件HDL建模示例——PC

```
16  `timescale  1ns/1ns
17
18  module PC( CLK_I, Reset_I, Addr_I, PCWrite_I, PC_O ) ;
19      input               CLK_I;          // system clock
20      input               Reset_I;        // reset signal
21      input [31:0]        Addr_I;         // next PC
22      input               PCWrite_I;      // write enable
23      output[31:0]        PC_O;           // PC output
24
25
26      /*  internal reg and wire   */
27      reg [31:0]  addr ;                  // latch the address
28
29      /* read register */
30      assign PC_O = addr;
31
32      always@ ( posedge CLK_I or posedge Reset_I )
33      begin
34          if(Reset_I)
35              addr <= 'hBFC00000;
36          else if( PCWrite_I )
37              addr <= Addr_I;
38      end
39
40  endmodule
```

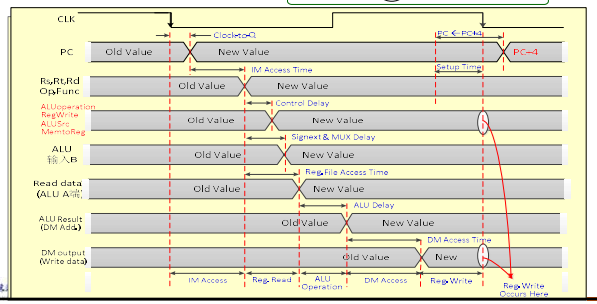

第六讲 MIPS处理器设计

3.3 单周期数据通路性能分析



R

LW



3.3 单周期数据通路性能分析

❖指令周期（指令执行时间）

- ➢R指令周期
  - 取指令（IM Access Time）
  - 读寄存器（Register File Access Time）
  - ALU运算（ALU Operation）
  - 写寄存器（Register File Access Time)
- ➢LW指令周期
  - 取指令（IM Access Time）
  - 读寄存器（Register Access Time）
  - ALU运算（ALU Operation）
  - 读数据（DM Access Time)
  - 写寄存器（Register File Access Time)

实际上，不同类型的指令可能具有不同的指令周期

3.3 单周期数据通路性能分析

MIPS不同类型指令的指令周期

Instruction class	Functional units used by the instruction class				
R-type	Instruction fetch	Register access	ALU	Register access	
Load word	Instruction fetch	Register access	ALU	Memory access	Register access
Store word	Instruction fetch	Register access	ALU	Memory access	
Branch	Instruction fetch	Register access	ALU		
Jump	Instruction fetch				

数据通路各部分以及各类指令的执行时间

Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	50	100	0	50	400 ps
Load word	200	50	100	200	50	600 ps
Store word	200	50	100	200		550 ps
Branch	200	50	100	0		350 ps
Jump	200					200 ps

3.3 单周期数据通路性能分析

❖指令执行时间计算

1. 方式一：采用单周期，即所有指令周期固定为单一时钟周期
   - 时钟周期有最长的指令决定（LW指令），为 600ps
   - 指令平均周期 = 600ps
2. 方式二：不同类型指令采用不同指令周期（可变时钟周期）
   - 假设指令在程序中出现的频率
     - lw指令 ：25%
     - sw指令 ：10%
     - R类型指令：45%
     - beq指令 ：15%
     - j指令 ：5%
   - 平均指令执行时间

     600*25%+550*10%+400*45%+350*15%+200*5% = 447.5ps

—若采用可变时钟周期，时间性能比单周期更高；
—但控制比单周期要复杂、困难，得不偿失。
—改进方法：改变每种指令类型所用的时钟数，即采用多周期实现

机器如何处理指令?

❖处理指令是什么意思?

❖冯诺依曼模型/结构

A = 指令执行之前程序员可见的体系结构状态

↓

处理指令

↓

A’ = 指令执行之后程序员可见的体系结构状态

❖处理指令：根据ISA的指令规范将A变换成A’

“处理指令”的步骤

- ISA 抽象地说明给定一条指令和A，A’ 应该是什么
  - 定义一个抽象的有限状态机
    - 状态 = 程序员可见的状态
    - 次态逻辑 = 指令执行的规范
  - 从 ISA 的视角，指令执行的过程中A和A’之间没有“中间状态”
    - 每条指令对应一个状态转换
- 微体系结构实现 A 向 A’的转换
  - 有很多种实现方式的选择
  - 我们可以加入程序员不可见的状态来优化指令执行的速度：每条指令有多个状态转换
    - 选择 1：A → A’（在一个时钟周期内完成 A 到 A’ 的转换）
    - 选择 2：A → A+MS1 → A+MS2 → A+MS3 → A’（使用多个时钟周期完成 A 到 A’的转换）

最基本的指令处理引擎

- 每条指令花费一个时钟周期来执行
- 只用组合逻辑来实现指令的执行
  - *没有中间的、程序员不可见的状态更新*

A = 时钟周期开始时的体系结构状态（程序员可见）

↓

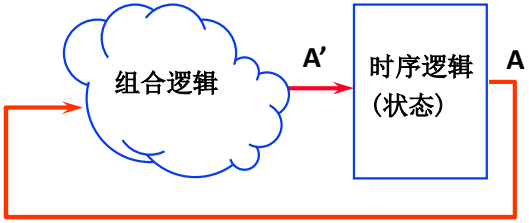
在一个时钟周期内处理指令

↓

A’ = 时钟周期结束时的体系结构状态(程序员可见)

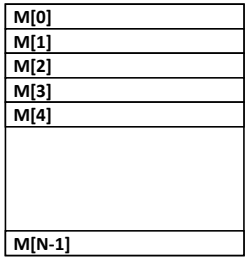
最基本的指令处理引擎

- 单周期机器



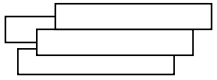
- 时钟周期长度由谁来决定?
- 组合逻辑中的关键路径由谁来决定?

程序员可见(体系结构)的状态



内存

用地址索引的存储位置的阵列



寄存器

- 在ISA中会给寄存器命名(相当于地址)
- 通用和专用寄存器

程序计数器

当前指令的内存地址

- 指令和程序指定如何转换程序员可见的状态值

单周期 vs. 多周期

- 单周期的机器
  - 每条指令执行需要一个时钟周期
  - 所有状态的更新在指令执行结束的时刻完成
  - 劣势：最慢的指令决定时钟周期的长度 → 时钟周期时间长
- 多周期的机器
  - 指令处理分到多个周期/阶段中完成
  - 指令执行过程中可以更新状态
  - 但是体系结构状态的更新只能在指令执行结束的时刻完成
  - 与单周期相比的“优势”：最慢的 “阶段” 决定时钟周期长度

单周期和多周期在微体系结构层面都遵从冯诺依曼结构

指令处理“周期”

- 指令在“控制单元”的指示下一步一步地处理
- 指令周期：指令处理的步骤序列
- 从根本上说，指令处理大约分为6个阶段：
  - 取指令
  - 译码
  - 计算地址
  - 取操作数
  - 执行
  - 存结果
- 不是所有的指令都需要所有6个阶段

指令处理“周期” vs. 机器时钟周期

- 单周期的机器：
  - 指令处理周期的所有阶段都在*一个机器时钟周期*中完成
- 多周期的机器：
  - 指令处理周期的所有阶段可以在*多个机器时钟周期*中完成
  - 实际上，每个阶段都可以在多个时钟周期中完成

单周期vs.多周期:控制&数据

- 单周期的机器：
  - 数据信号操作的同时产生控制信号（在同一个时钟周期内起作用）
  - 与一条指令相关的所有事情都发生在一个时钟周期内
- 多周期的机器：
  - 下一个周期需要的控制信号可以在前一个周期就产生
  - 数据通路上的延迟可以和控制处理的延迟重叠

数据通路和控制逻辑的设计方法很多

- 有很多方法可以用来设计数据通路和控制逻辑
- 单周期，多周期，流水线等
- 单总线vs. 多总线数据通路
- 硬连线/组合逻辑vs. 微码/微程序控制
  - 由组合逻辑电路产生控制信号
  - 在存储器结构中存储控制信号
- 控制信号和结构依赖于数据通路的设计

初步的性能分析

- 指令执行时间
  - {CPI} x {clock cycle time}
- 程序执行时间
  - 所有指令的[{CPI} x {clock cycle time}]之和
  - {指令数} x {平均 CPI} x {clock cycle time}
- 单周期微体系结构的性能
  - CPI = 1
  - Clock cycle time 长
- 多周期微体系结构的性能
  - CPI = 每条指令不同
    - 平均 CPI → 希望能很小
  - Clock cycle time 短

现在，我们有两个独立的自由度可以优化