CPU设计任务

设计任务

设计任务：思考并设计一台简单计算机系统，自己设计CPU，连接存储器、基本的I/O)，要求实现Minisys指令系统中的31条指令。

提 纲

- 1 Minisys概述
- 2 Minisys指令解析
- 3 数据通路设计
- 4 流水线设计思想*

MiniSys指令系统简介

- MiniSys采用32位MIPS指令中最常用的31条指令，其寄存器组织，指令格式等均采用MIPS指令系统相同的格式。
  - 共有32个32位寄存器
  - 32位定长格式指令
  - 4种寻址方式

MiniSys寄存器组

寄存器名	寄存器号	约定用途
$zero	0	常数0，该寄存器永远只返回0。
$at	1	用做汇编器的暂时变量。
$v0~$v1	2~3	用来存放一个子程序(函数)的非浮点运算的结果或返回值。
$a0~$a3	4~7	存放子程序(函数)调用时的非浮点参数。
$t0~$t7	8~15	暂时变量，子程序(函数)使用时不保存这些寄存器的值，因此调用后它们的值会被破坏。
$s0~$s7	16~23	8个子程序用寄存器。子程序(函数)必须在返回之前恢复这些寄存器的值以保证其没有变化。

MiniSys寄存器组

寄存器名	寄存器号	约定用途
$t8~$t9	24~25	暂时变量，子程序(函数)使用时不保存这些寄存器的值，因此调用后它们的值会被破坏。
$i0~$i1	26~27	分别保存两个中断到来时程序的返回地址。（该两寄存器定义和MIPS中的有所不同）
$s9	28	第10个子程序用寄存器。（该定义和MIPS中的不同）
$sp	29	堆栈指针，对它的调整必须显式的通过指令来实现，硬件不支持堆栈指针的调整。
$s8	30	第9个子程序用寄存器。（该定义和MIPS中的不同）
$ra	31	存放调用子程序(函数)时的返回地址。

MiniSys寻址方式

- 立即数寻址
  - 指令中第3操作数可使用16位二进制立即数
- 相对寻址
  - 操作数是下一条指令的PC值（PC+4）加上一个32位偏移量
- 寄存器寻址
  - 操作数存放在寄存器中，指令里放的是寄存器号
- 寄存器相对寻址
  - 操作数存放在数据存储器中，其有效地址由两部分组成，基地址放在一个寄存器中，偏移部分为一个16位的立即数

MiniSys指令目录

- 算术指令－add, addu, addi, addiu, sub, subu
- 逻辑指令－and, andi, or, ori, xor, xori, nor, sll, srl, sra, sllv, srlv, srav
- 数据传送指令－lw, sw, lui
- 比较、条件转移指令－beq, bne, slt, slti, sltu, sltiu
- 无条件转移指令－j, jr, jal

MiniSys指令格式

- （1）R-format

add $1, $2, $3 # $1=$2+$3

6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
op	rs	rt	rd	shamt	funct
0	2	3	1	0	32
000000	00010	00011	00001	00000	100000

MiniSys指令格式

- （2）I-format

lw $1, 10($2)　　# $1=Memory[$2 +10]

6-bit	5-bit	5-bit	16-bit
op	rs	rt	Address/Immediate
35	2	1	10
100011	00010	00011	0000 0000 0000 1010

MiniSys指令格式

- （3）J-format

j 10000 # go to 10000

6-bit 26-bit

op	Target/ Address
2	2500

000010 00000 00000 0000 1001 1100 0100

提 纲

- 1 Minisys概述
- 2 Minisys指令解析
- 3 数据通路设计
- 4 流水线设计思想*

部分MiniSys指令详解(1)

- 加法指令（ R-format）

add $s1, $s2, $s3 # $s1=$s2+$s3

6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
op	rs	rt	rd	shamt	funct
0	18	19	17	0	32
000000	10010	10011	10001	00000	100000

部分MiniSys指令详解(2)

- 减法指令（R-format）

```
sub  $s1,  $s2,  $s3   #  $s1=$s2-$s3
```

6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
op	rs	rt	rd	shamt	funct
0	18	19	17	0	34
000000	10010	10011	10001	00000	100010

部分MiniSys指令详解(3)

- 逻辑与指令（R-format）

```
and  $s1,  $s2,  $s3   #  $s1=$s2  &  $s3
```

6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
op	rs	rt	rd	shamt	funct
0	18	19	17	0	36
000000	10010	10011	10001	00000	100100

部分MiniSys指令详解(4)

- 逻辑或操作（R-format）

or $s1, $s2, $s3 # $s1=$s2 | $s3

6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
op	rs	rt	rd	shamt	funct
0	18	19	17	0	37
000000	10010	10011	10001	00000	100101

部分MiniSys指令详解(5)

- 有符号立即数加（I-format）

```
addi  $s1,  $s2,  100   #  $s1=$s2  +  100
```

6-bit	5-bit	5-bit	16-bit
op	rs	rt	Immediate
8	18	17	100
001000	10010	10001	0000 0000 0110 0100

立即数做符号扩展

部分MiniSys指令详解(6)

- 立即数逻辑与指令（I-format）

andi $s1, $s2, 100 # $s1=$s2 & 100

6-bit	5-bit	5-bit	16-bit
op	rs	rt	Immediate
12	18	17	100
001100	10010	10001	0000 0000 0110 0100

立即数做0扩展

部分MiniSys指令详解(7)

- 立即数逻辑或指令（I-format）

ori $s1, $s2, 100 # $s1=$s2 | 100

6-bit	5-bit	5-bit	16-bit
op	rs	rt	Immediate
13	18	17	100
001101	10010	10001	0000 0000 0110 0100

立即数做0扩展

部分MiniSys指令详解(8)

- 逻辑左移指令（R-format）

```
sll  $s1,  $s2,  10   #  $s1= shift（$s2）left  logic
                         10 bits
```

6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
op	rs	rt	rd	shamt	funct
0	0	18	17	10	0
000000	00000	10010	10001	01010	000000

部分MiniSys指令详解(9)

- 逻辑右移指令（R-format）

srl $s1, $s2, 10 # $s1= shift（$s2）right logic 10 bits

6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
op	rs	rt	rd	shamt	funct
0	0	18	17	10	2
000000	00000	10010	10001	01010	000010

部分MiniSys指令详解(10)

- 算术右移指令（R-format）

```
sra  $s1, $s2, 10   #  $s1= shift（$s2）right
                       arithmatic 10 bits
```

6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
op	rs	rt	rd	shamt	funct
0	0	18	17	10	3
000000	00000	10010	10001	01010	000011

部分MiniSys指令详解(11)

- 存储器读（I-format）

lw $s1, 100($s2)　　#　$s1=Memory[$s2 +100]

6-bit	5-bit	5-bit	16-bit
op	rs	rt	Offset
35	18	17	100
100011	10010	10001	0000 0000 0110 0100

立即数做符号扩展

部分MiniSys指令详解(12)

- 存储器写（I-format）

sw $s1, 100($s2) # Memory[$s2 +100] = $s1

6-bit	5-bit	5-bit	16-bit
op	rs	rt	Offset
43	18	17	100
101011	10010	10001	0000 0000 0110 0100

立即数做符号扩展

部分MiniSys指令详解(13)

- 相等则转移指令（I-format）

beq $s1, $s2,100 # if $s1=$s2, goto PC+4+100

6-bit	5-bit	5-bit	16-bit
op	rs	rt	Offset= immediate/4
4	17	18	25

000100 10001 10010 0000 0000 0001 1001

立即数做符号扩展

部分MiniSys指令详解(14)

- 不相等则转移转移指令（I-format）

bne $s1, $s2,100 # if $s1 ≠ $s2, goto PC+4+100

6-bit	5-bit	5-bit	16-bit
op	rs	rt	Offset = immediate/4
5	17	18	25
000101	10001	10010	0000 0000 0001 1001

立即数做符号扩展

部分MiniSys指令详解(15)

- 小于则设置指令（R-format）

slt $s1, $s2, $s3 # if $s2<$s3, $s1=1; else $s1=0

6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
op	rs	rt	rd	shamt	funct
0	18	19	17	0	42
000000	10010	10011	10001	00000	101010

部分MiniSys指令详解(16)

- 无条件转移指令（J-format）

```
j  10000        #  go to   10000
```

6-bit 26-bit

op	Target = Address/4
2	2500

000010 00000 00000 0000 1001 1100 0100

部分MiniSys指令详解(17)

- 过程调用指令（J-format）

jal 10000　　# $31=PC+4; go to 10000

6-bit	26-bit
op	Target = Address/4
3	2500

000011　00000　00000　0000　1001　1100　0100

部分MiniSys指令详解(18)

- 按寄存器内容转移指令（R-format）

```
jr  $ra   #  jump  register  $ra
```

6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
op	rs	rt	rd	shamt	funct
0	31	0	0	0	8
000000	11111	00000	00000	00000	001000

部分MiniSys指令详解(19)

- 立即数赋值指令（I-format）

lui $s1,100 # $s1 = 100 << 16

6-bit	5-bit	5-bit	16-bit
op	rs	rt	immediate
15	0	17	100
001111	00000	10001	0000 0000 0110 0100

提 纲

- 1 Minisys概述
- 2 Minisys指令解析
- 3 数据通路设计
- 4 流水线设计思想*

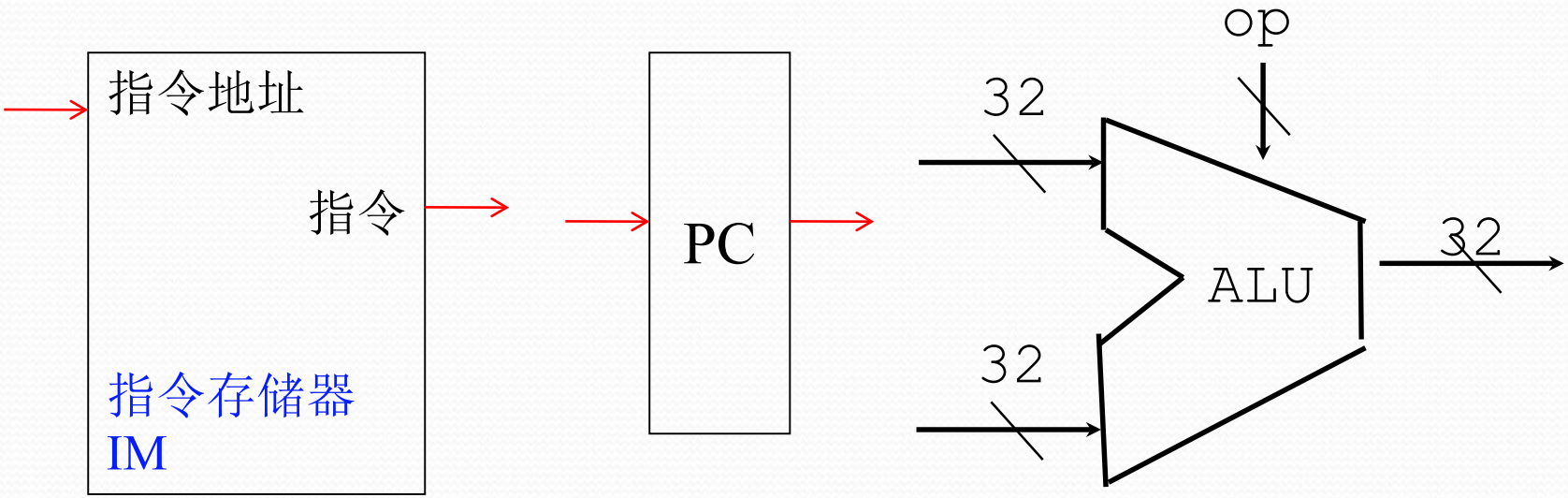
取指令的数据通路

首先确定数据通路需要的数据通路部件：

(1) 存储程序指令的部件—指令存储器IM

(2) 存放当前指令地址的单元--PC

(3) 实现增加PC值以指向下一条指令的地址—Adder

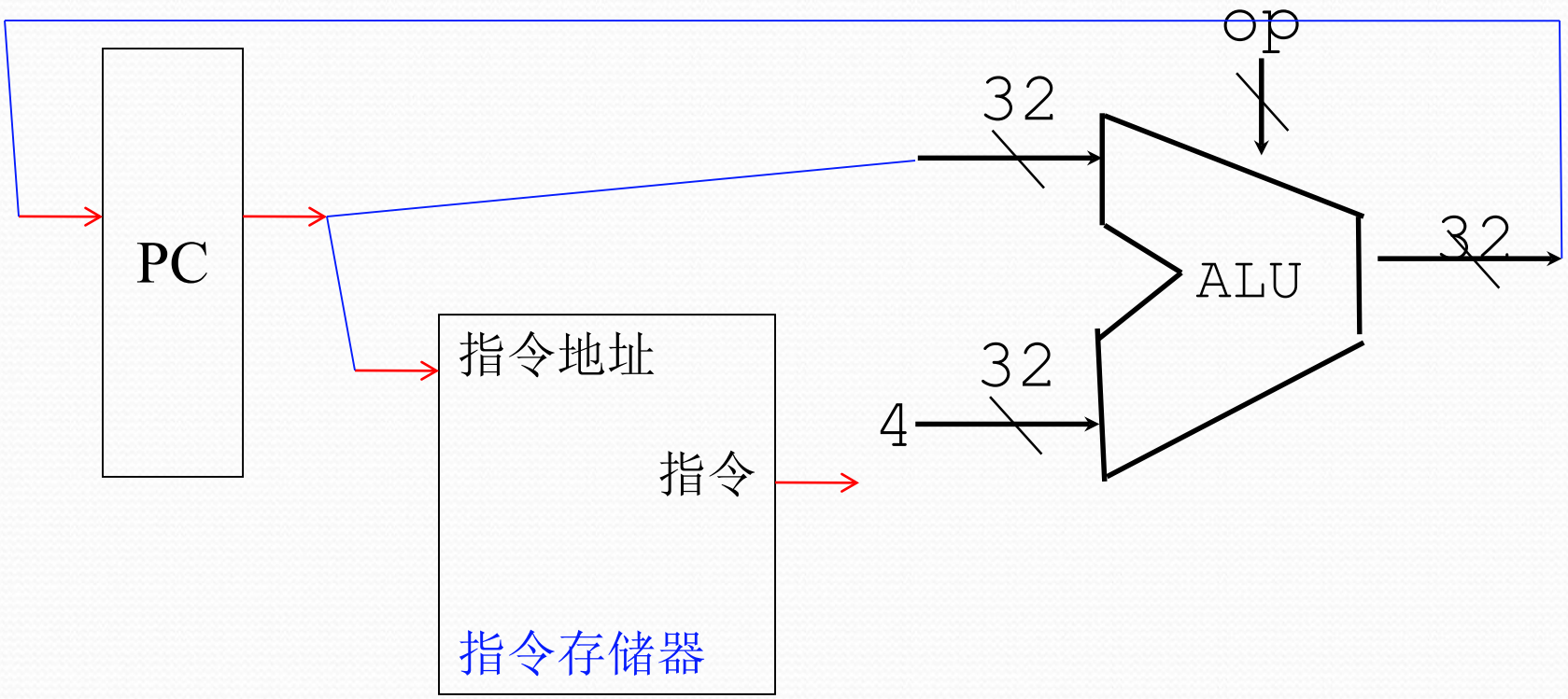


取指令的数据通路

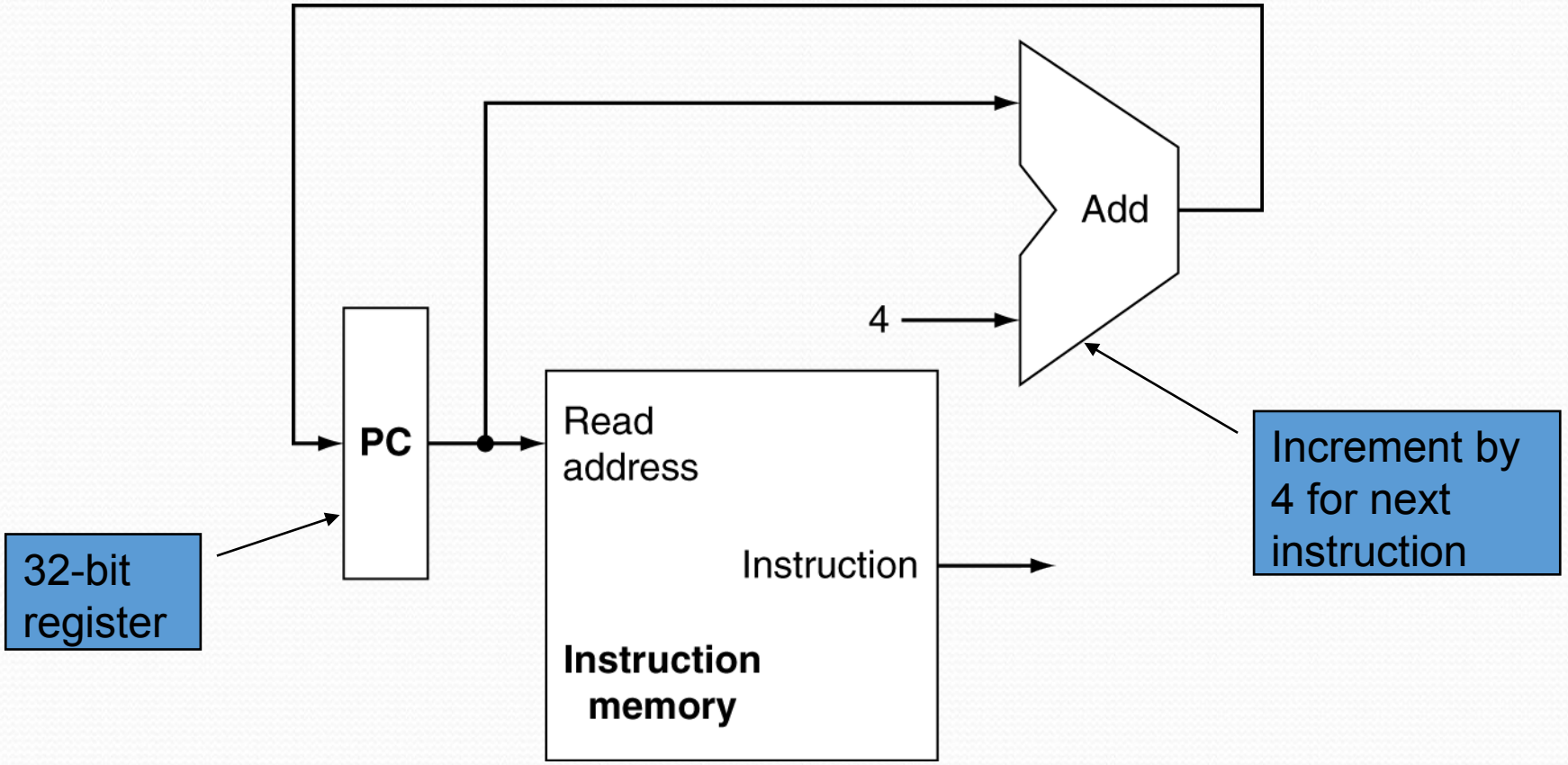
执行任何一条指令

从存储器中取出一条指令

为准备取下一条指令，PC必须指向下一条指令地址



取指令的数据通路


Add
4
PC
Read
address
Instruction
Instruction
memory
32-bit
register
Increment by
4 for next
instruction

各类指令的数据通路

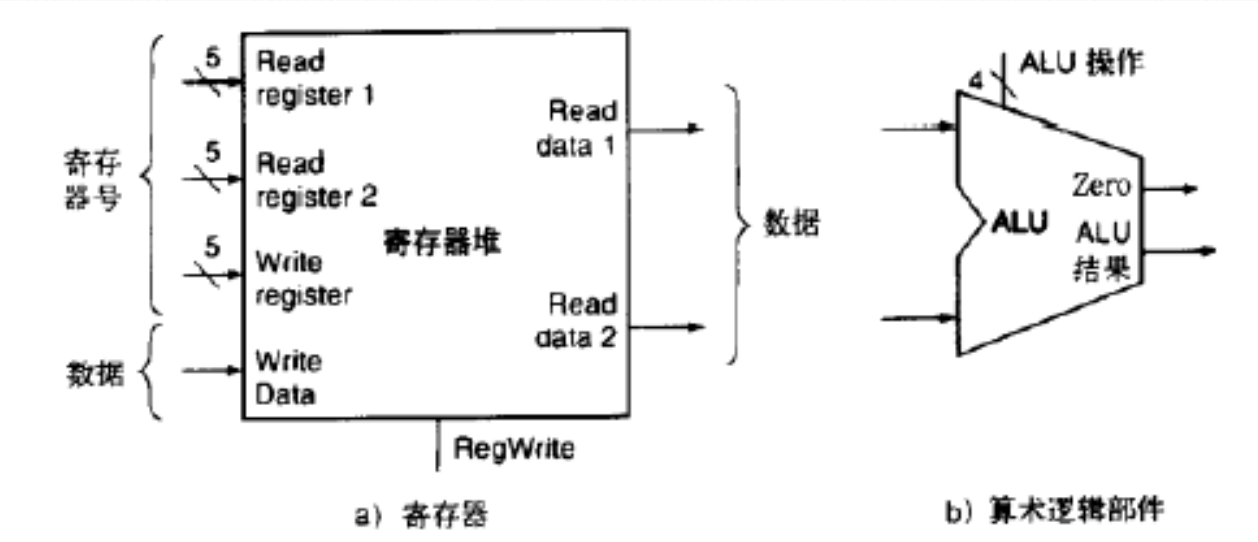
(1) R型指令，包括add, sub, and, or, etc.

典型格式：add $t1，$t2，$t3; 读$t2，$t3，结果写$t1。

32个通用寄存器位于1个寄存器堆(register file)结构中。

寄存器堆结构是一个寄存器的集合，其中的寄存器通过相应的寄存器序号进行读写。

另外，需要一个ALU对从寄存器中读出的数据运算。



ALU:两个32位输入数据，1个32位输出数据，1个结果为0时的标识

各类指令的数据通路

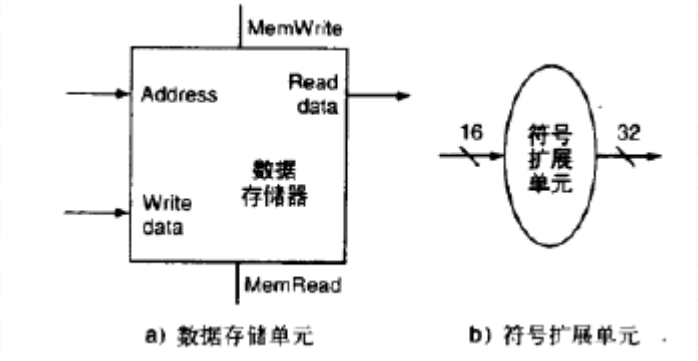
(2) 取字和存字指令，包括lw, sw, etc.

典型格式：lw $t1，offset($t2)

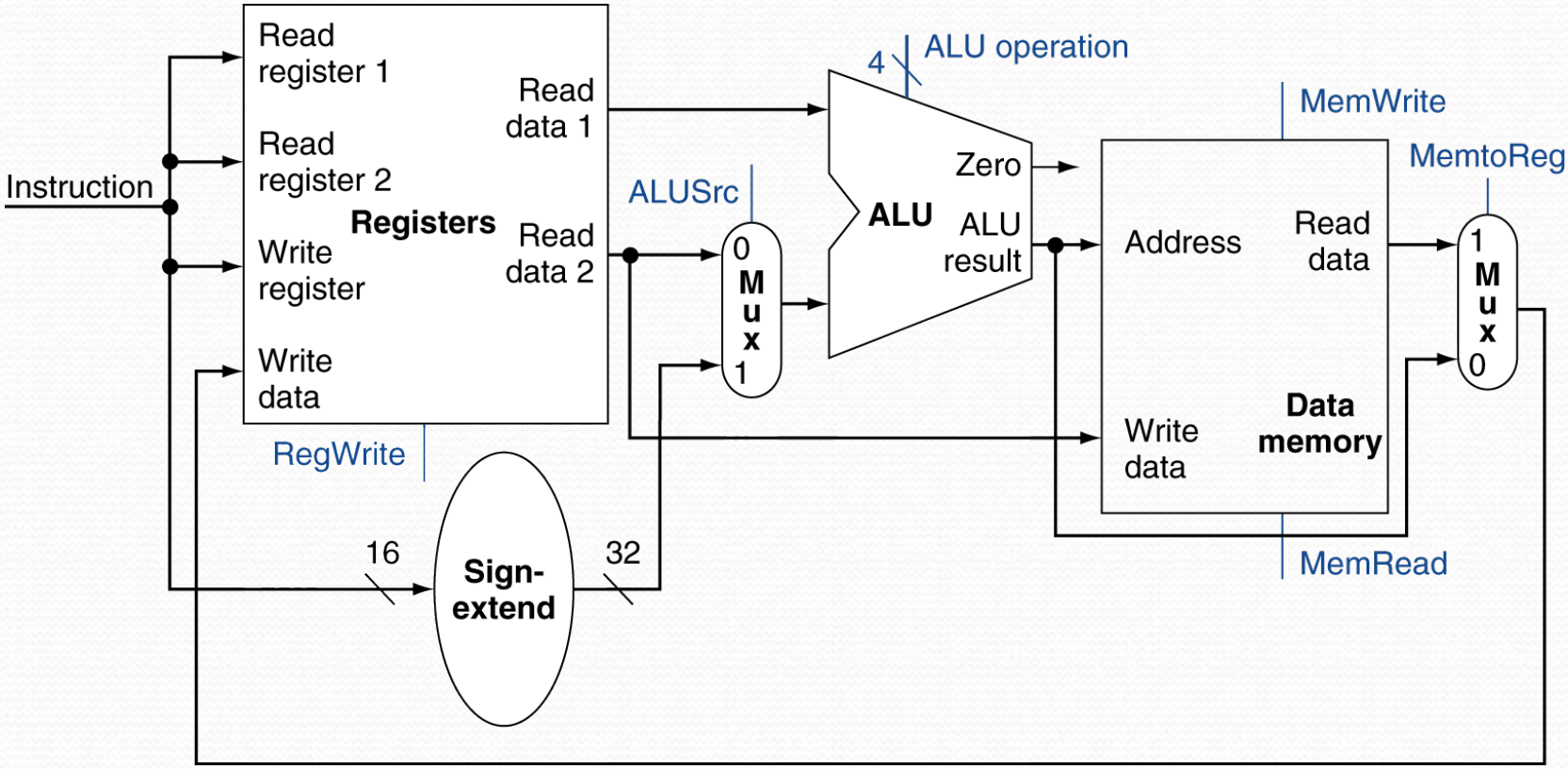
sw $t1，offset($t2)

将基址寄存器$t2中的内容和16位有符号偏移字段相加得到存储器地址。存数指令根据计算地址和$t1内容写存储器；取数指令根据计算地址读存储器写入$t1。

需要用到的器件：寄存器堆、ALU、数据存储器、符号扩展单元。



R-Type/Load/Store Datapath



各类指令的数据通路

(3) 转移指令beq

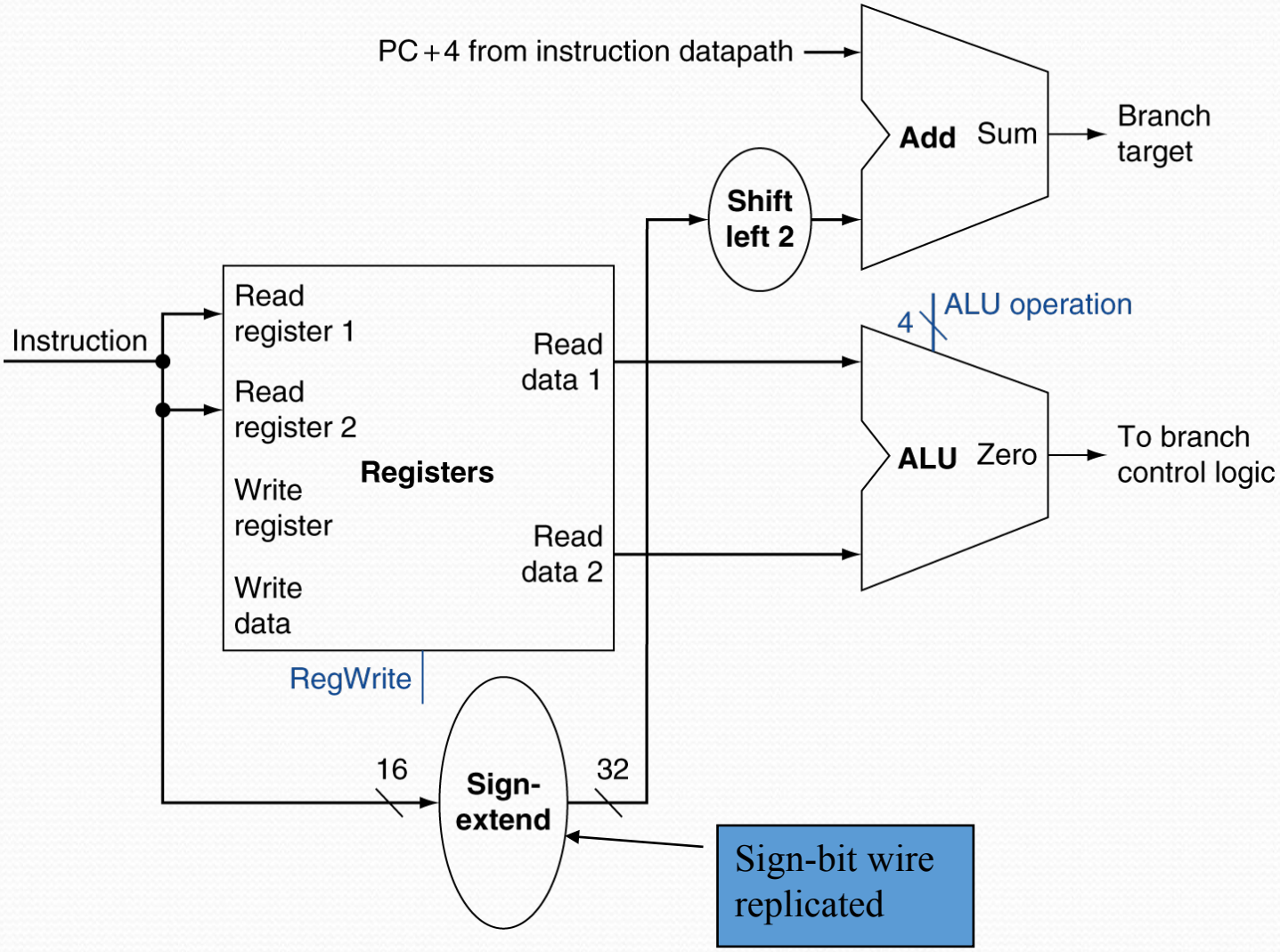
典型格式：beq $t1, $t2, offset

比较$t1和 $t2中数值，相等的话转移至PC+4+offset为目标地址处；不相等则执行后续指令。

具体做法: 用ALU计算$t1-$t2，根据结果转移

需要计算PC+4+offset

Branch Instructions



各类指令的数据通路

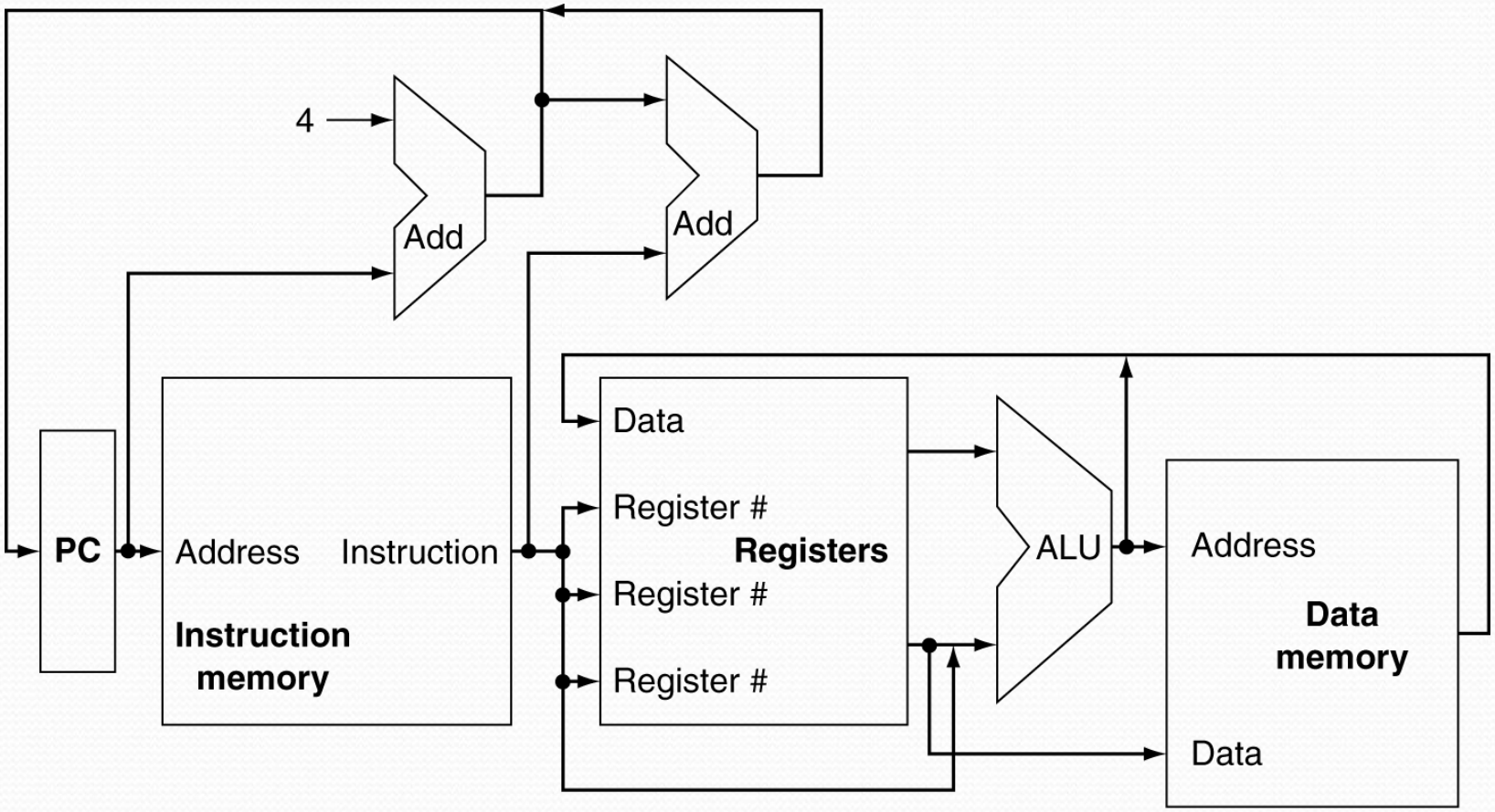
根据以上3类指令，构建简单的数据通路。

现在已经检视过单个指令类需要的数据通路部件，我们把它们组合到单个数据通路并加上控制来完成实现。

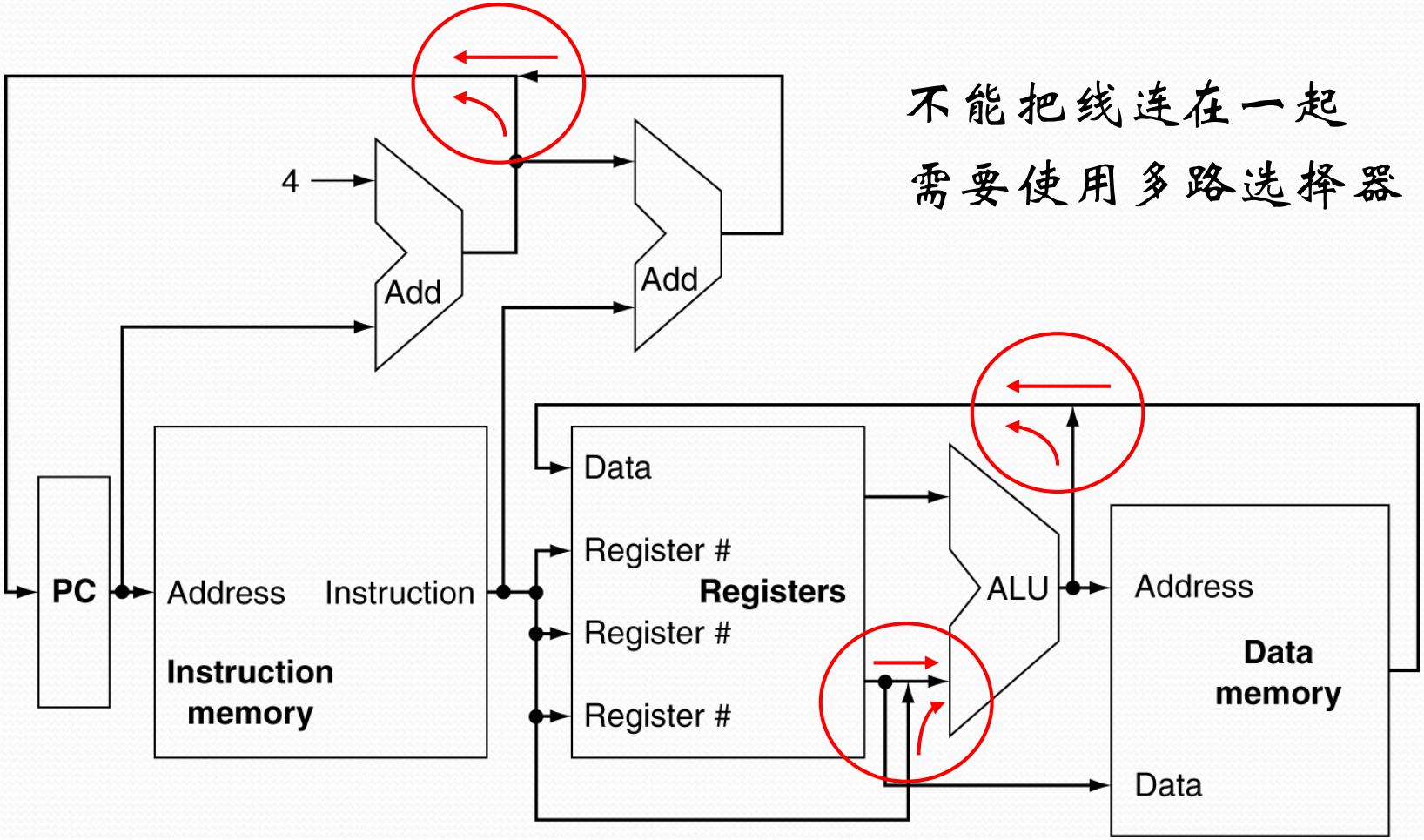
最简单的数据通路会尝试在一个时钟周期执行完整一条指令，数据通路资源不能重复使用。因此任何要使用不止一次的单元都要复制。因此还需要一个数据存储器之外的指令存储器。

要在两个不同的指令类间共享数据通路单元，需要允许多个到功能单元输入端的连接，使用多路复用器和控制信号在多个输入之间选择。

CPU概況


4
Add
Add
PC
Address
Instruction
Instruction memory
Data
Register #
Registers
Register #
Register #
ALU
Address
Data memory
Data

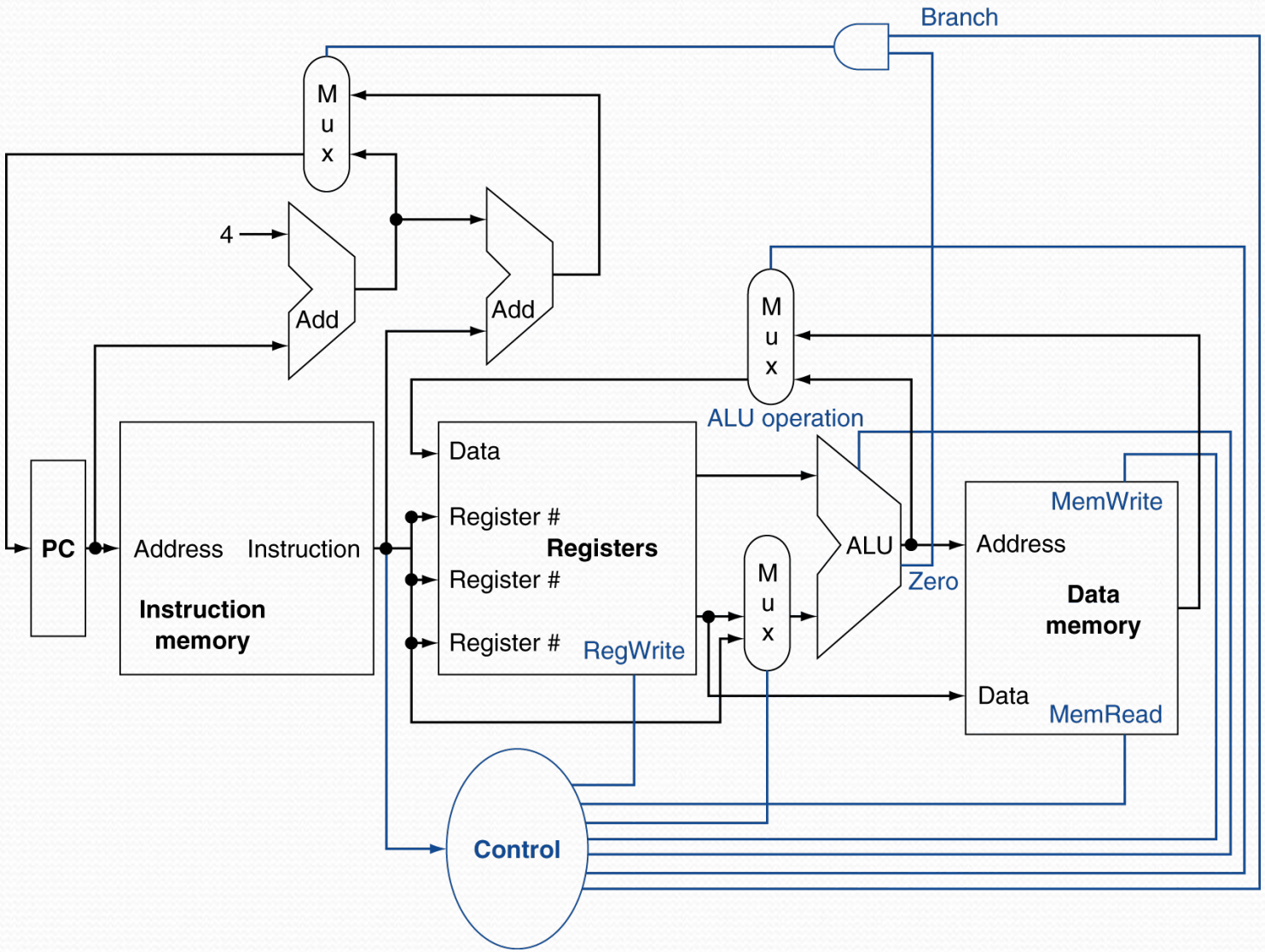
多路选择器



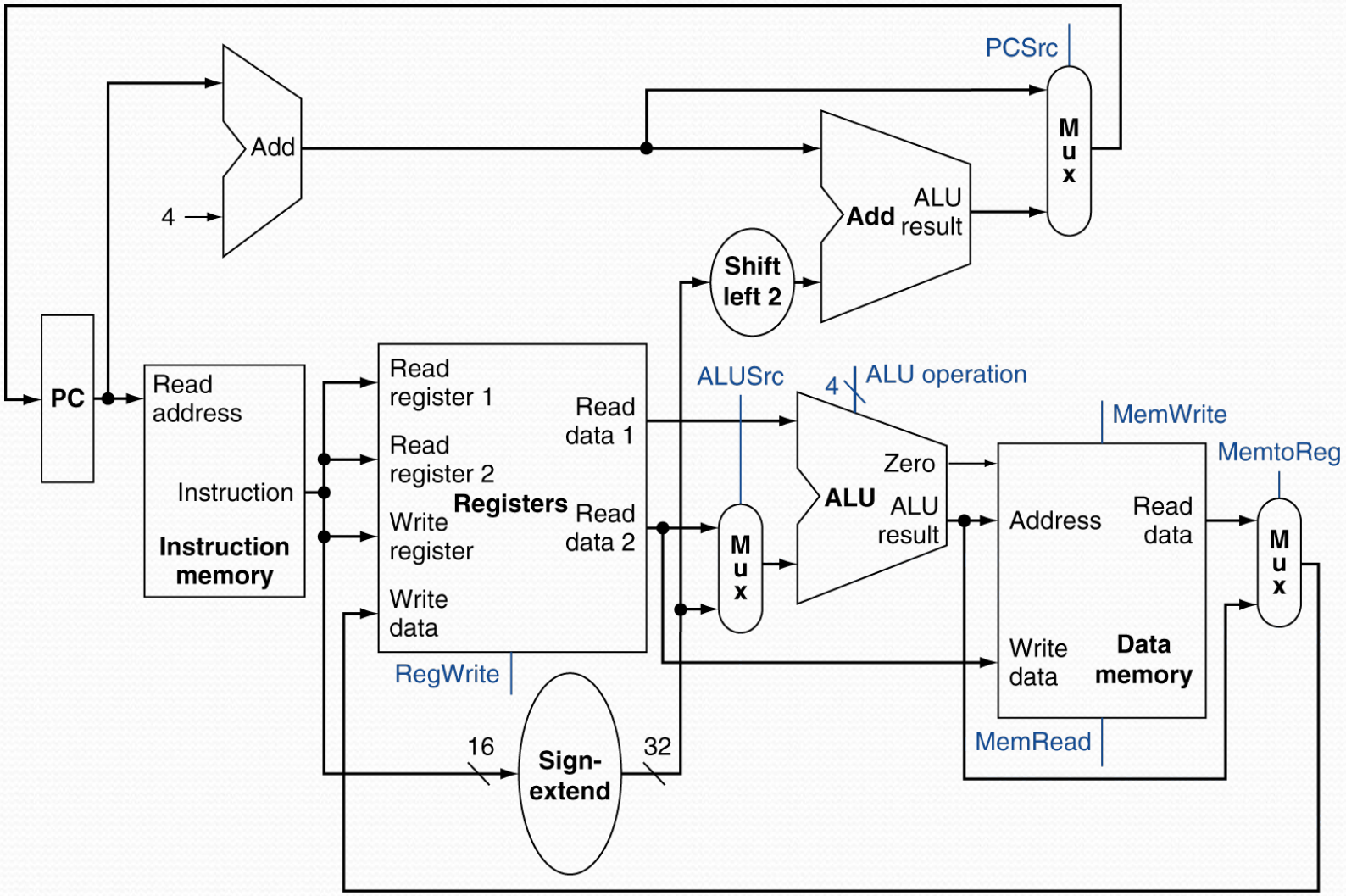
不能把线连在一起

需要使用多路选择器

控制单元


Branch
M
u
x
4
Add
Add
M
u
x
ALU operation
PC
Address
Instruction
Instruction
memory
Data
Register #
Registers
Register #
Register #
RegWrite
M
u
x
ALU
Zero
MemWrite
Address
Data
memory
Data
MemRead
Control

含控制信号的数据通路


PCSrc
Add
4
M
u
x
Add
ALU
result
Shift
left 2
PC
Read
address
Instruction
Instruction
memory
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU operation
M
u
x
Zero
ALU
ALU
result
MemWrite
MemtoReg
Address
Read
data
Write
data
Data
memory
MemRead
M
u
x
16
Sign-
extend
32

一个简单的实现方案

这里讨论一个最简单的MIPS子集实现方案

将把前面的各部分集成起来，并加入必要的控制线路，得到简单的数据通路及其控制，实现的指令包括取字(lw)，存字(sw)，等值分支(beq)，算术逻辑指令add、sub、and、or等。

(1) ALU的控制

ALU有4个控制输入，有些未被编码，这16种在该子集中使用其中的6种组合

ALU 控制	功能
0000	与(AND)
0001	或(OR)
0010	加(add)
0110	减(subtract)
0111	小于置位(set-on-less-than)
1100	异或(NOR)

一个简单的实现方案

对于lw和sw指令，ALU进行加法运算获得存储器地址；

对于R型指令，根据指令低6位功能字段，确定ALU执行

对于等值分支、不等分支指令，ALU做减法；

为了控制更快，使用一个小的控制单元生成4位ALU控制信号，这个功能单元的输入为2位，称为ALUOp。

操作码	ALUOp	指令操作	功能字段	期望ALU动作	ALU 控制
lw	00	取字	XXXXXX	add	0010
sw	00	存字	XXXXXX	add	0010
beq	01	相等分支	XXXXXX	subtract	0110
R-type	10	加	100000	add	0010
		减	100010	subtract	0110
		与	100100	AND	0000
		或	100101	OR	0001
		小于置1	101010	set-on-less-than	0111

一个简单的实现方案

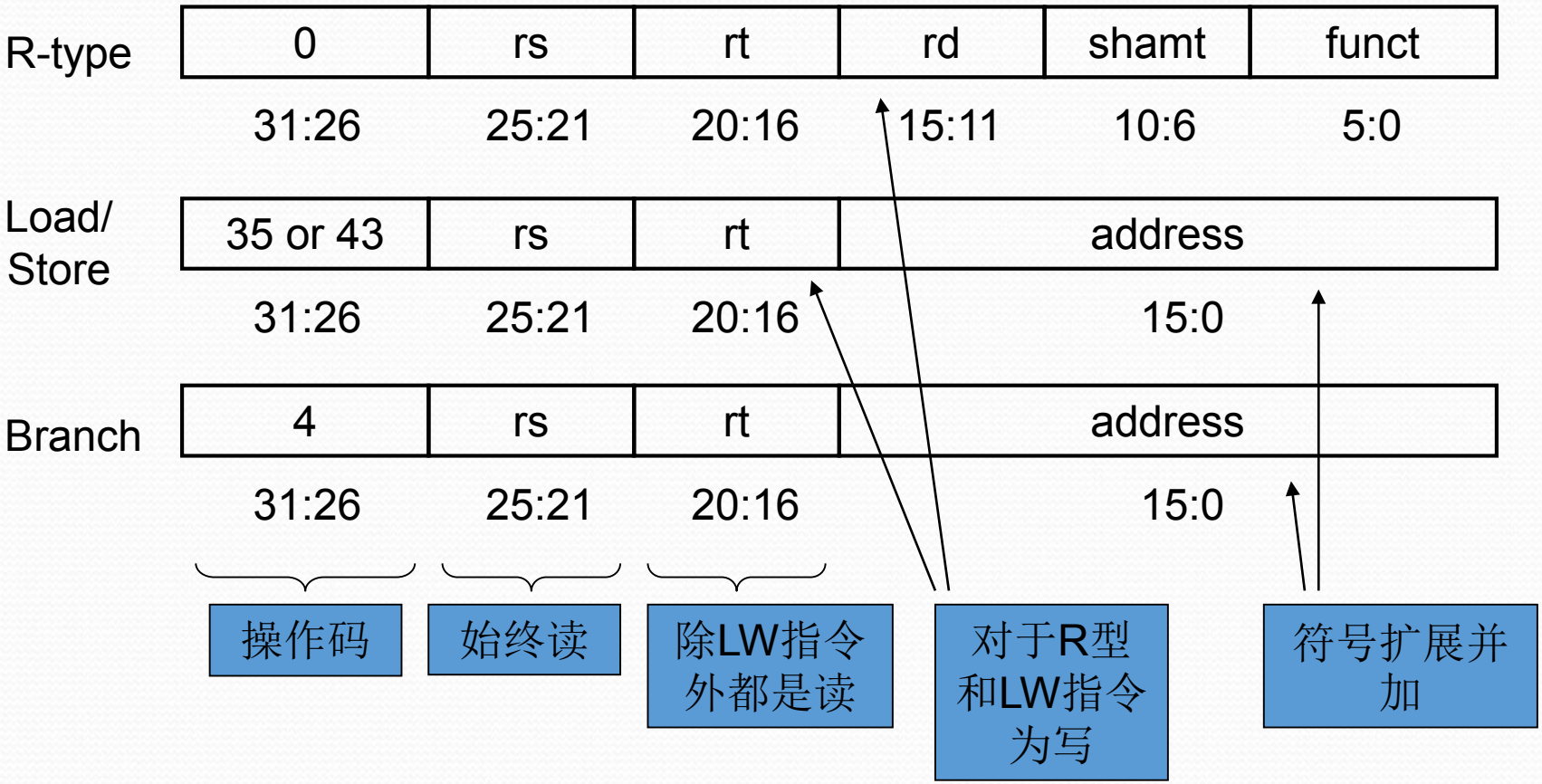
从上图中可见ALU控制输入最高位为0，因此需要根据2位的ALUOp和6位的功能字段映射出3位的ALU控制位

为了设计这个逻辑，有必要将ALUop和功能字段的有意义的组合生成一张真值表

ALUOp		功能字段						操作
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
0	1	X	X	X	X	X	X	0110
1	0	X	X	0	0	0	0	0010
1	0	X	X	0	0	1	0	0110
1	0	X	X	0	1	0	0	0000
1	0	X	X	0	1	0	1	0001
1	0	X	X	1	0	1	0	0111

主控单元设计

控制信号来源于指令。



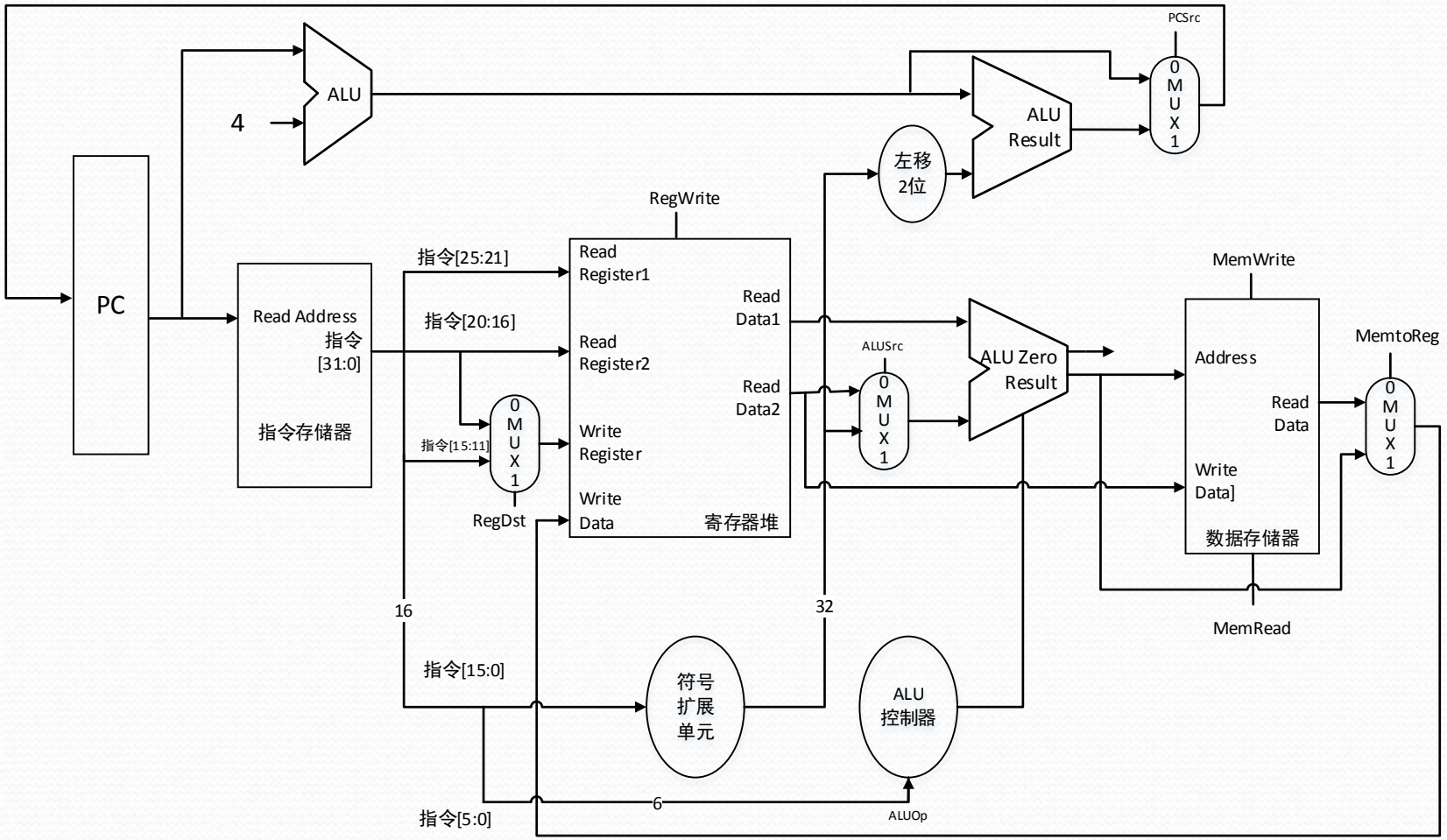
一个简单的实现方案

遵循以下几条指令格式的主要规则：

- op字段：总是31:26位，用op[5:0]表示
- 对于R型指令、分支指令、存数指令，要读取的两个寄存器为rs和rt字段，分别为25:21和20:16位
- 存数和取数指令的基址寄存器字段在25:21位
- 等值分支指令、存数取数指令的偏移量在15:0位
- 有两个地方存放目标寄存器。存数20:16、R型15:11位

根据这些信息，在简单数据通路上 加上指令标记和一个额外的多路复用器，就可以给出下图所示的数据通路

带控制信号的数据通路


PCSrc
ALU
4
M
U
X
0
1
ALU
Result
左移
2位
RegWrite
PC
Read Address
指令
[31:0]
指令存储器
指令[25:21]
指令[20:16]
指令[15:11]
Read
Register1
Read
Register2
Write
Register
Write
Data
Read
Data1
Read
Data2
寄存器堆
RegDst
ALUSrc
ALU Zero
Result
MemWrite
Address
Read
Data
Write
Data]
数据存储器
MemtoReg
MemRead
16
32
指令[15:0]
符号
扩展
单元
ALU
控制器
6
指令[5:0]
ALUOp

一个简单的实现方案

图中给出了7条1位控制线和1条2位(ALUOp) 控制信号。

- ALUOp前面已经定义
- 其他7条1位控制线的功能

信号名	失效时的作用	有效时的作用
RegDst	对于寄存器写的目的寄存器序号来自于rt字段(20:16)	对于寄存器写的目的寄存器序号来自于rd字段(15:11)
RegWrite	无	写寄存器的寄存器输入由读数据的输入值写入
ALUSrc	第二个ALU操作数来自第二个寄存器堆输出（读数据2）	第二个ALU操作数是经过符号扩展的指令的低16位
PCSrc	PC的值由加法器的输出替换为PC+4	PC的值由加法器输出的分支目的地址替换
MemRead	无	地址输入指定的数据存储器内容放置到读数据输出上
MemWrite	无	地址输入指定的数据存储器内容被写数据输入的值替换
MemtoReg	送往寄存器写数据输入的值来源于ALU	送往寄存器写数据输入的值来自于数据存储器

一个简单的实现方案

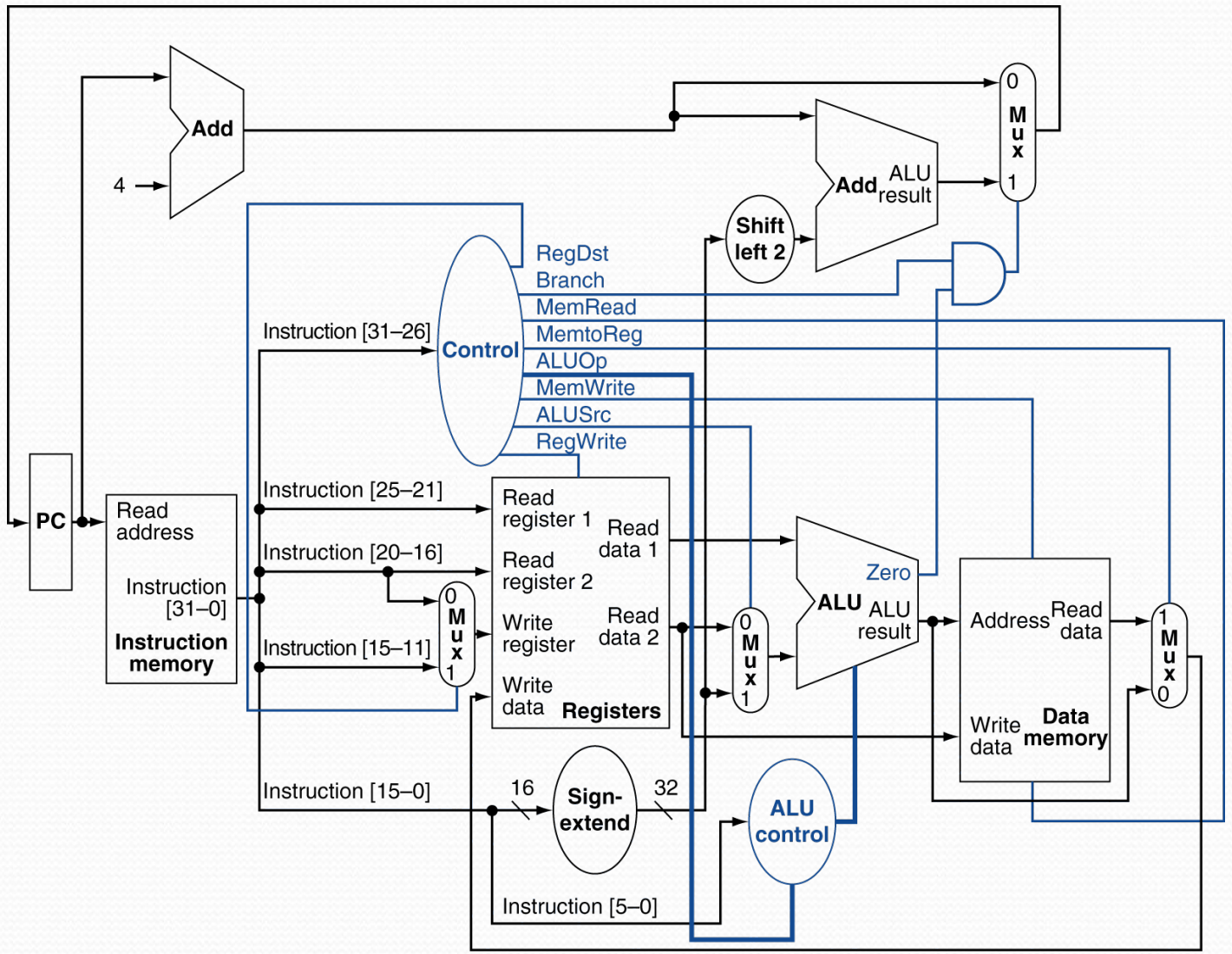
图中给出了7条1位控制线和1条2位(ALUOp) 控制信号

9个控制信号的状态可根据控制单元的6个输入信号(即操作码)来设置。

在为控制单元建立一组方程或者一个真值表之前，应该先非正式地定义一下控制功能。由于控制线的状态只能由操作码决定，定义在每种操作码值下各控制信号应取0、1或是不关心(“x”)。

指令	RegDst	ALUSrc	MemtoReg	RegWrite	MenRead	MemWrite	Branch	ALUOp1	ALUOp0
R型	1	0	0	1	0	0	0	1	0
Lw	0	1	1	1	1	0	0	0	0
Sw	X	1	X	0	0	1	0	0	0
Beq	X	0	X	0	0	0	1	0	1

带控制信号的数据通路


Add
4
0
M
u
x
1
Add
ALU result
Shift left 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
ALU control
Zero
ALU
ALU result
Address
Read data
Write data
Data memory

数据通路操作

R型指令的数据通路：

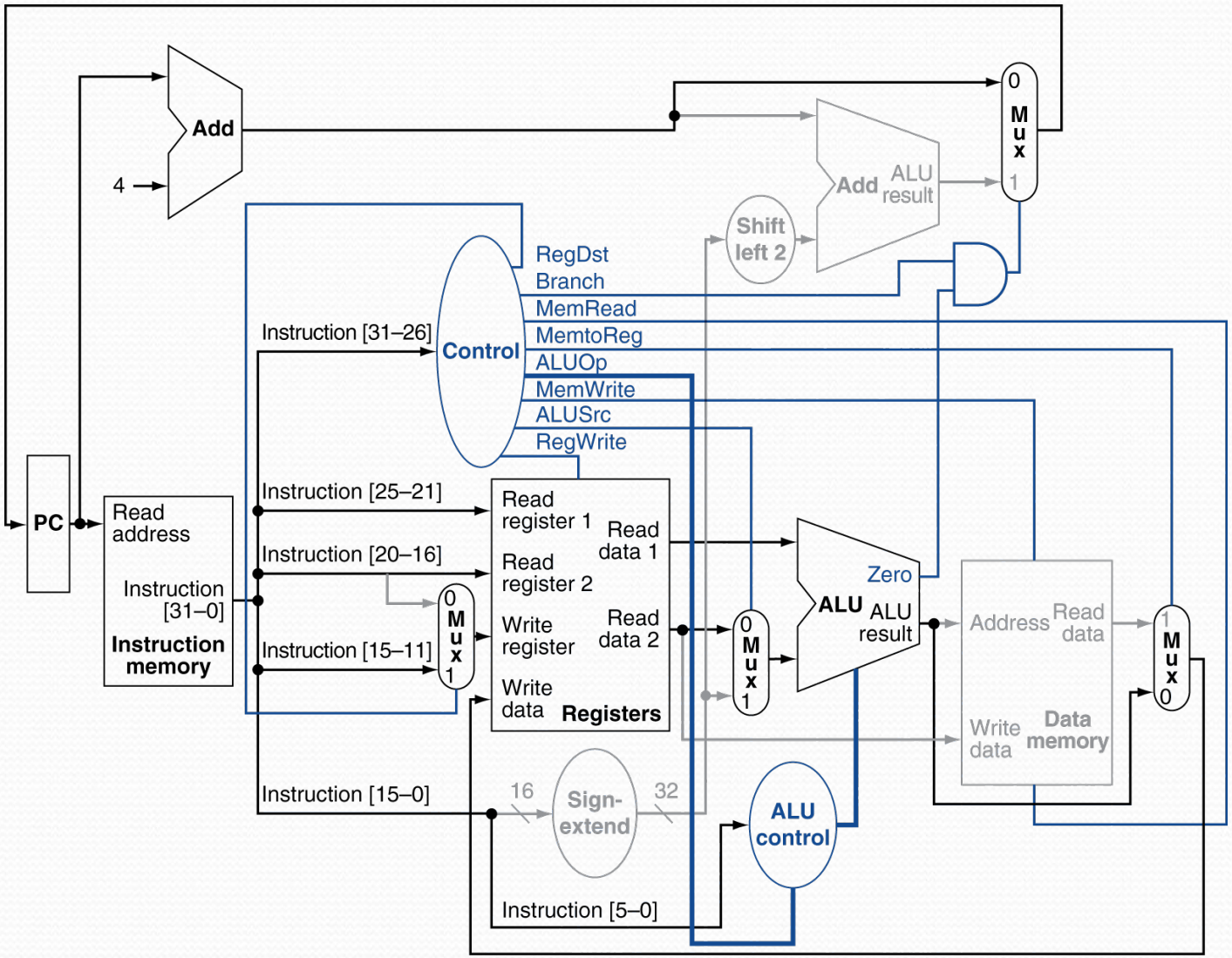
如　　add $t1, $t2, $t3

指令分为四步执行：

1) 从指令存储器中取出指令，PC值增加。
2) 寄存器$t2, $t3中的内容从寄存器堆中读出。
3) ALU根据功能码(指令的5:0位)确定ALU的功能，对从寄存器堆读出的数据进行操作。
4) ALU的结果被写入寄存器堆，目标寄存器($t1)根据指令的15:11位选择。

其数据通路如下图：

R-Type Instruction



数据通路操作

取字指令的数据通路：

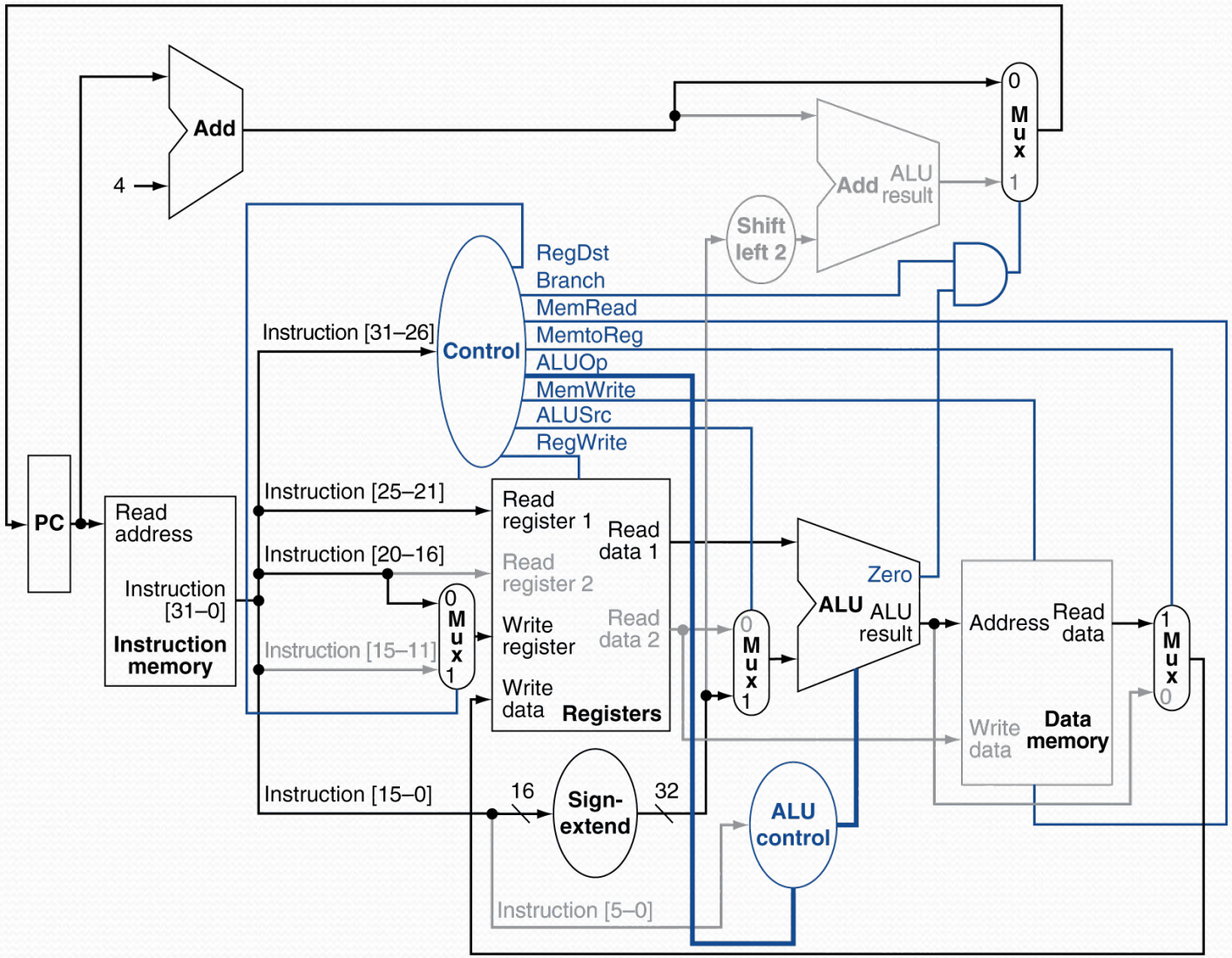
如　lw $t1, offset($t2)

指令分为5步执行：

1) 从指令存储器中取出指令，PC值增加。
2) 从寄存器堆读出$t2的值。
3) ALU将从寄存器堆读出的值与符号扩展后的指令低16位(offset)值相加。
4) 将ALU的结果作为数据存储器的地址。
5) 存储单元的数据写入寄存器堆；目标寄存器由指令的20:16位($t1)指出。

其数据通路如下图：

Load Instruction



数据通路操作

相等则分支指令的数据通路：

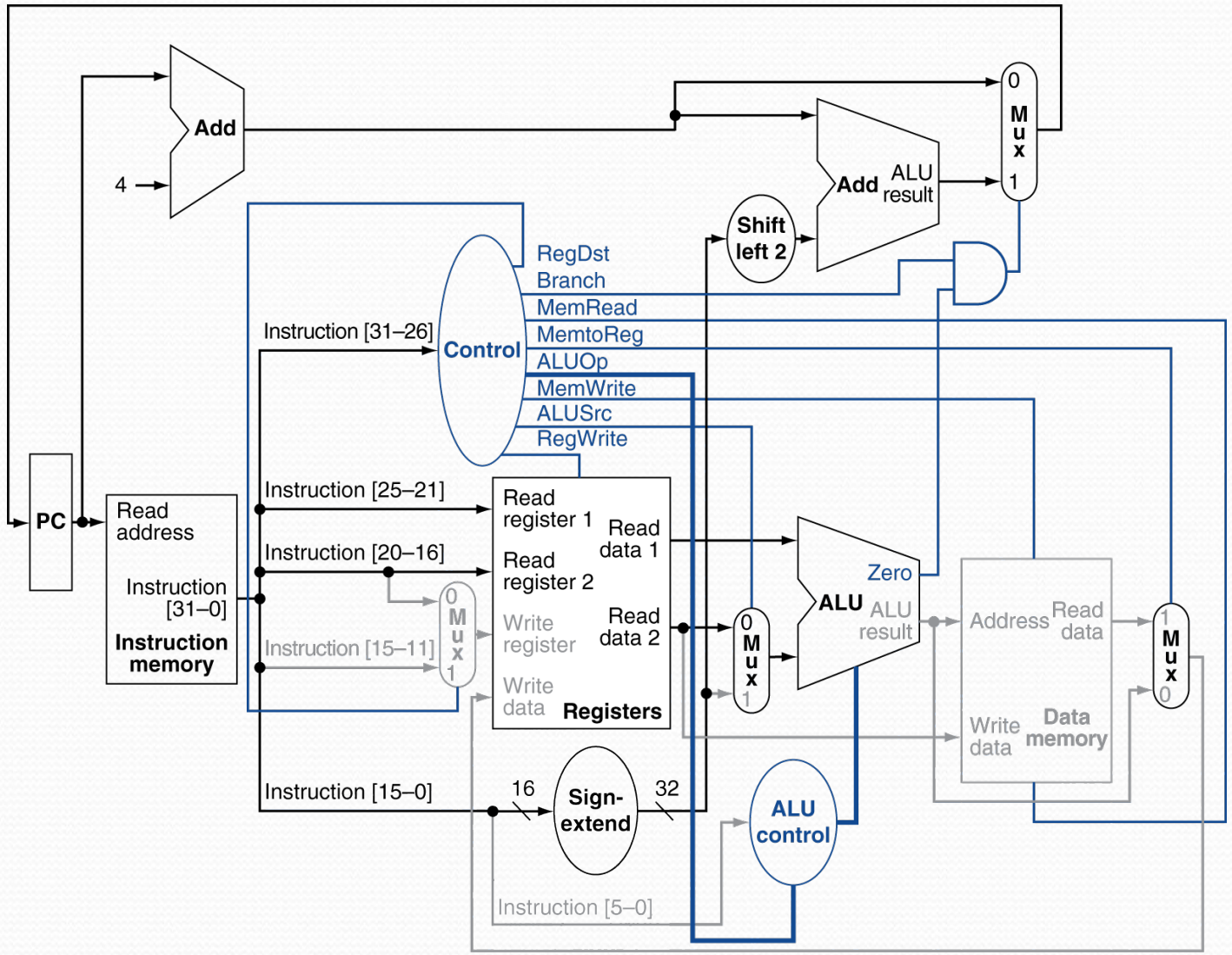
如　beq $t1, $t2, offset

指令分为5步执行：

1) 从指令存储器中取出指令，PC值增加。
2) 从寄存器堆读出$t1和 $t2的值。
3) ALU将从寄存器堆读出的两数相减。PC+4的值与符号扩展并左移2位后的指令低16位(offset)值相加；结果即分支目标地址。
4) 根据ALU的零输出决定哪个加法器的结果存入PC。

其数据通路如下图：

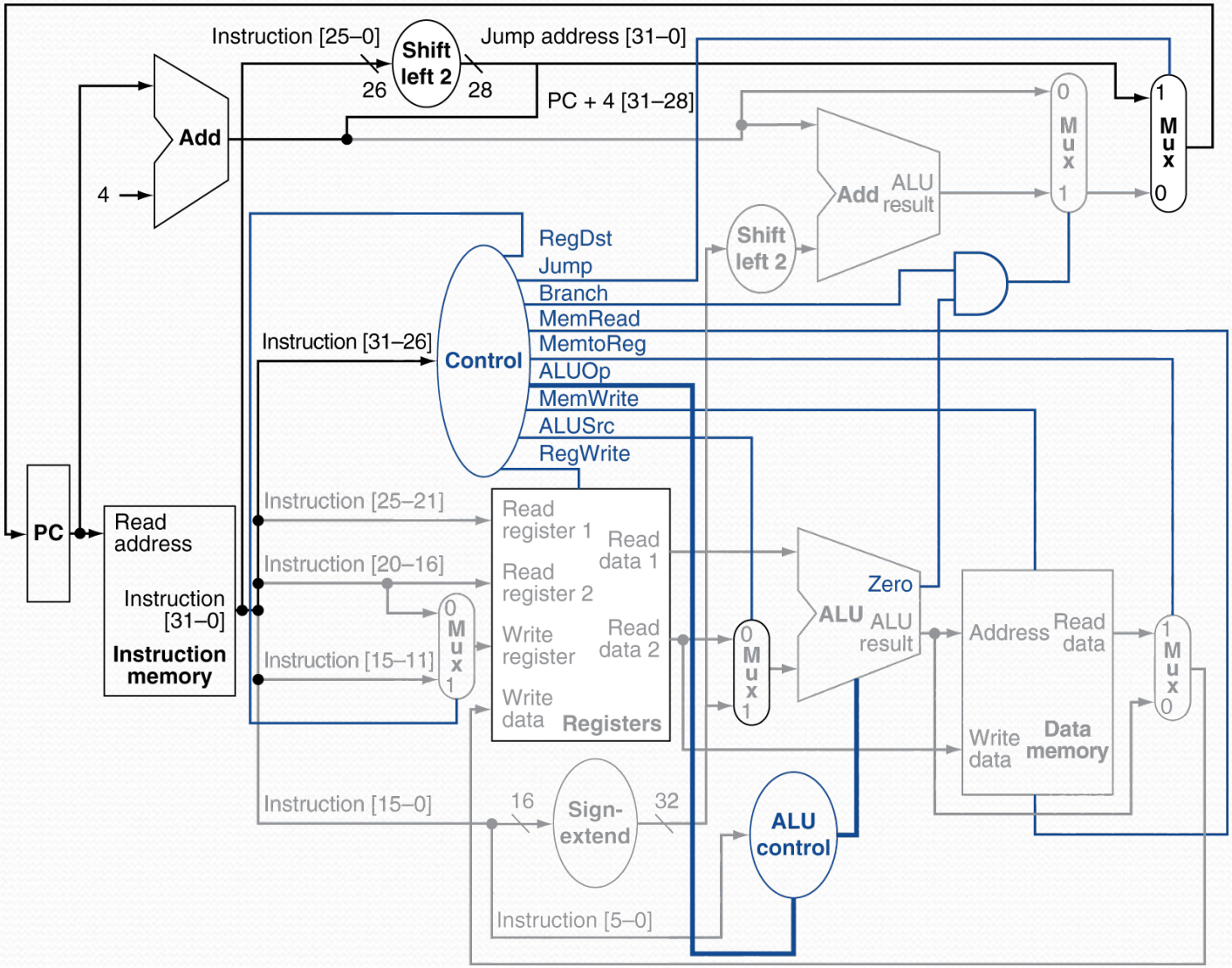
相等则分支指令


Add
4
Add
ALU result
0
M
u
x
1
Shift left 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
ALU
Zero
ALU result
ALU control
Address
Read data
Write data
Data memory

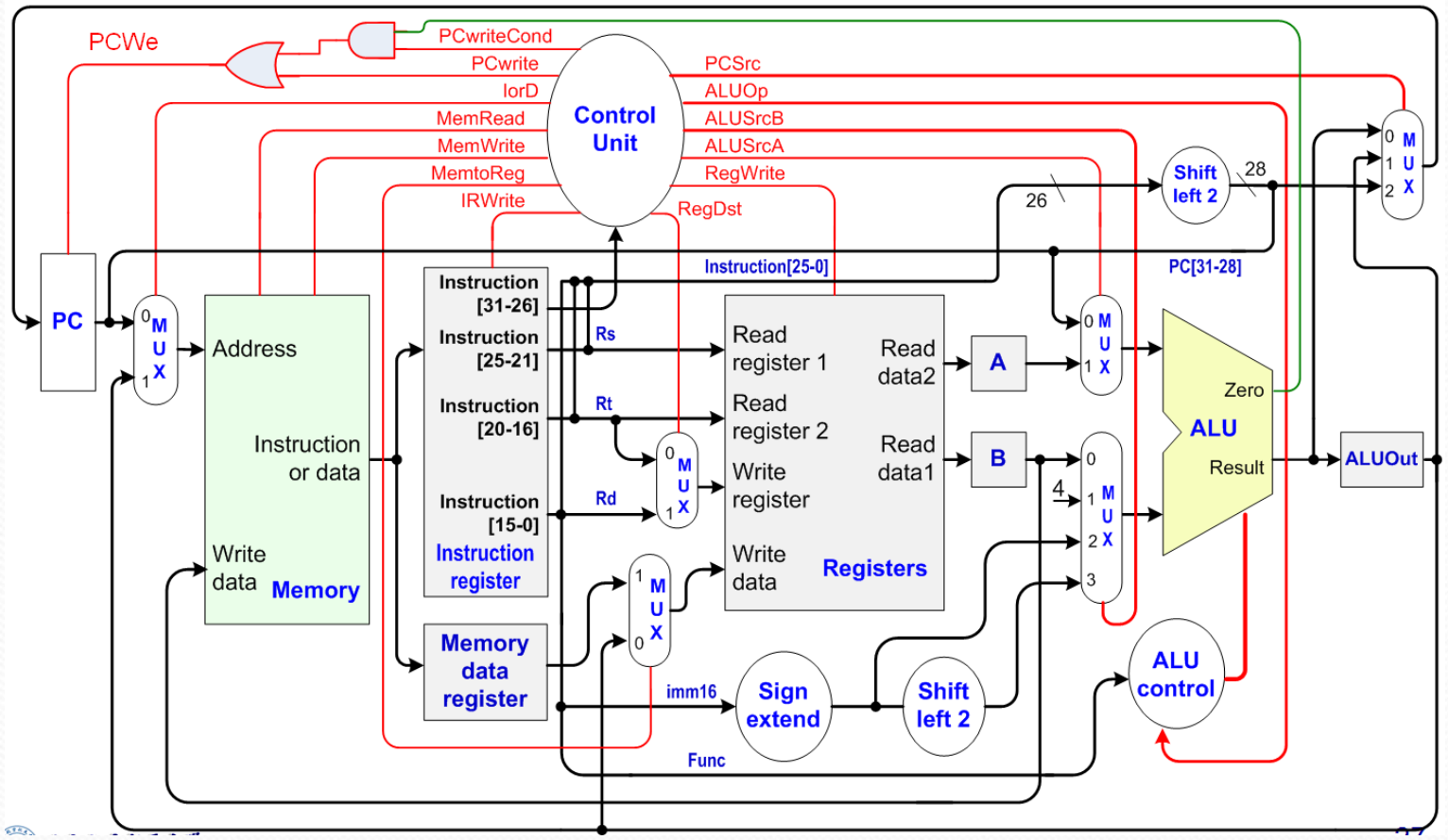
无条件转移指令的实现

- 使用字地址实现跳转
- 利用以下连接更新PC
  - PC值的高4位
  - 26-bit跳转地址
  - 00-最低两位0
- 需要通过对操作码阶码得到控制信号

无条件转移指令的数据通路



完整的数据通路


PCWe
PCwriteCond
PCwrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Control Unit
PCSrc
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Shift left 2
26
28
PC[31-28]
Instruction[25-0]
PC
M U X
Address
Instruction or data
Write data
Memory
Instruction [31-26]
Instruction [25-21]
Instruction [20-16]
Instruction [15-0]
Instruction register
Memory data register
Rs
Rt
Rd
Read register 1
Read register 2
Write register
Write data
Registers
Read data2
Read data1
A
B
4
Zero
ALU
Result
ALUOut
imm16
Sign extend
Shift left 2
ALU control
Func

提 纲

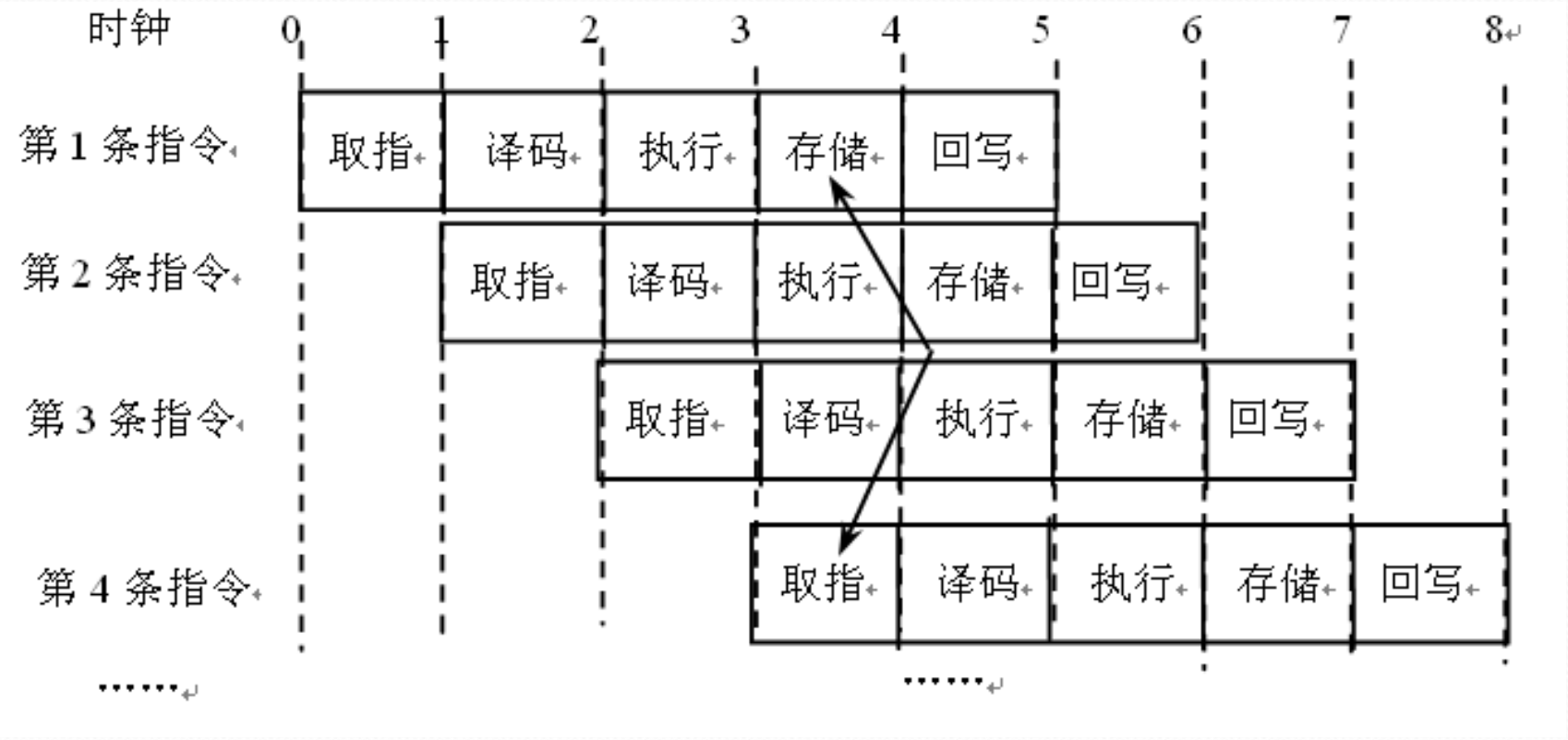
- 1 Minisys概述
- 2 Minisys指令解析
- 3 数据通路设计
- 4 流水线设计思想*

设计流水线处理器的考虑

- 选择合适的流水级数
- 尽量保持流水线各级延迟相等
- 流水线中的相关性

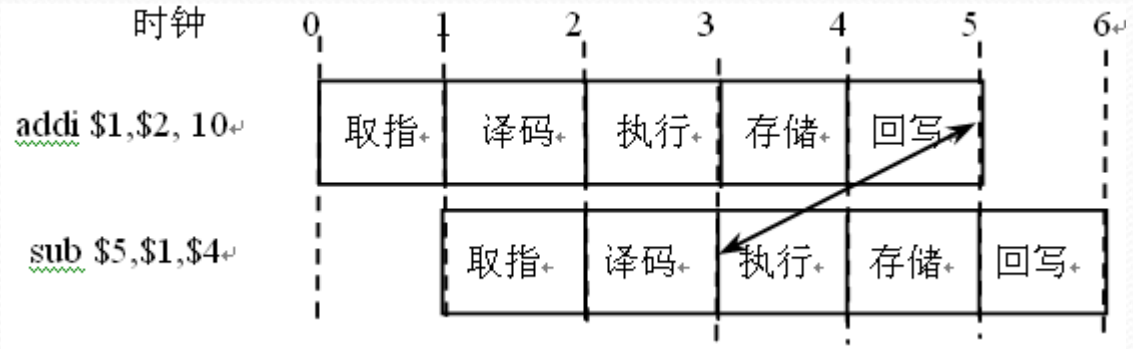
流水线相关

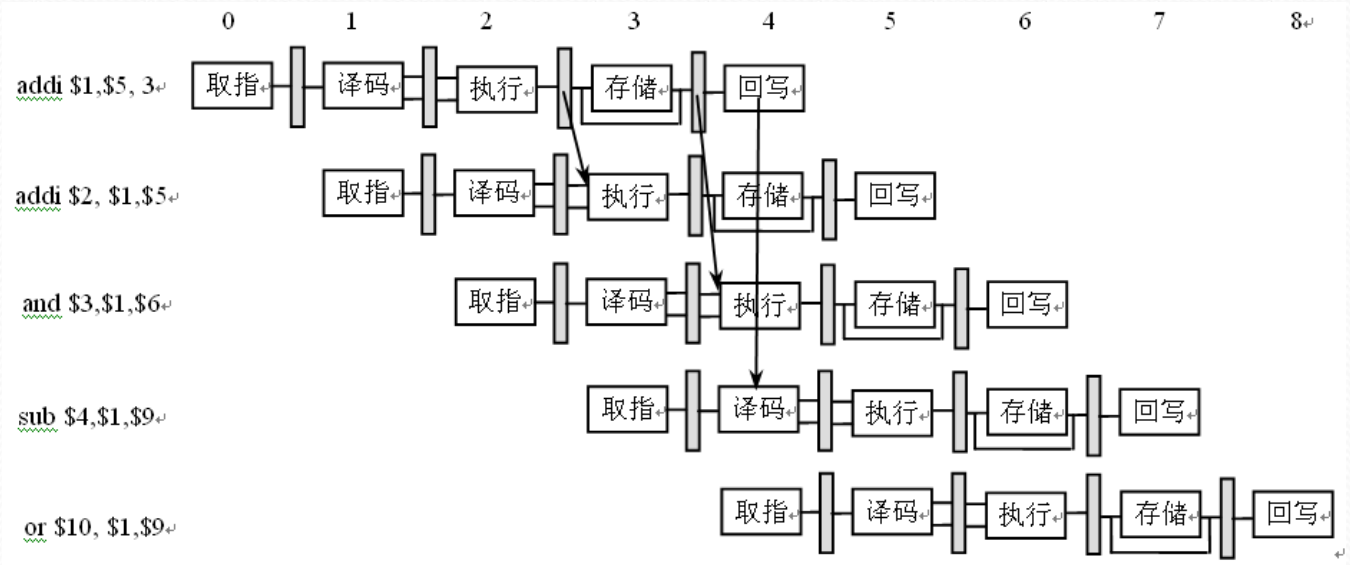
- 结构相关
  - ?



流水线相关

- 数据相关
  - 写后读相关(RAW)
    - 阻塞
    - 乱序
    - 定向转发法



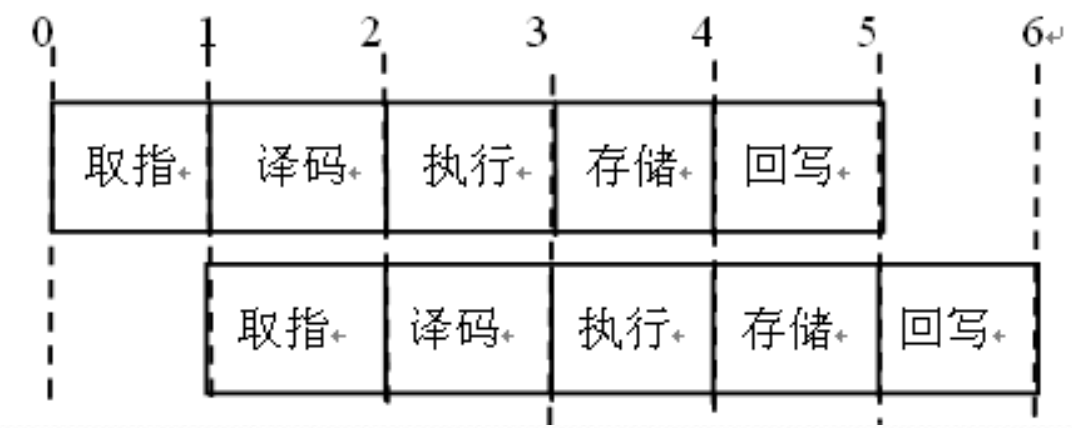


流水线相关

- 数据相关
  - 读后写相关(WAR)
    - ?

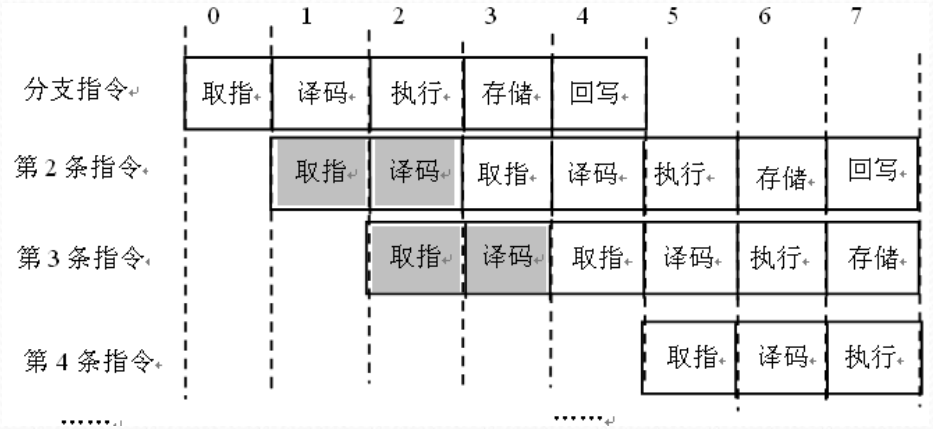
流水线相关

- 数据相关
  - 写后写相关(WAW)
    - ?



流水线相关

- 控制相关
  - 阻塞
  - 分支预测
    - 静态预测
    - 动态预测
  - 分支地址计算尽量前移
  - 延迟槽法



其他指令Ia～Im
转移指令In
延迟槽中指令(空指令)
其他指令Io～Iz

延迟转移法的目标代码

其他指令Ia～If、Ig～Im中与In有关的指令
转移指令In
延迟槽中指令(Ig～Im中与In无关的指令或空指令)
其他指令Io～Iz

优化延迟转移法的目标代码

谢谢！