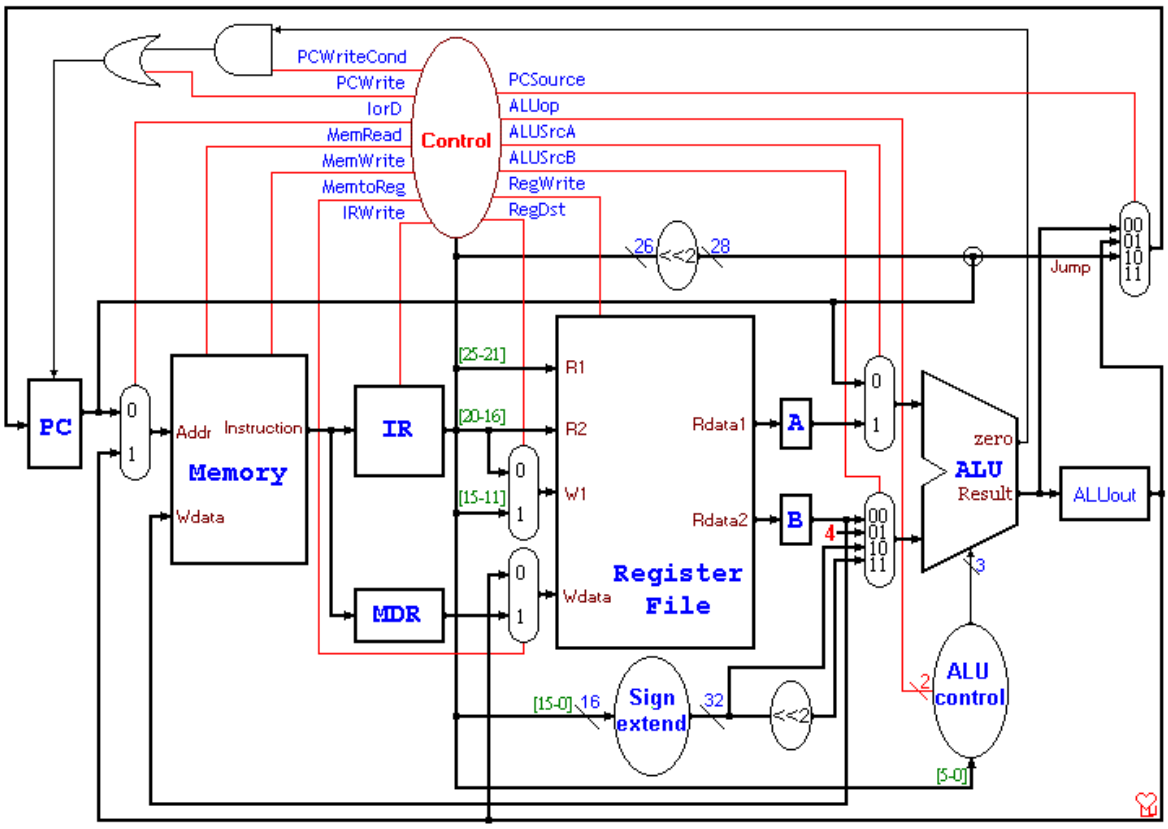
多周期DataPath



一般认为寄存器组要1单位时间，而存储器和ALU需要2单位时间

R 指令

Step 1: 注意不能每次一个时钟都读指令，而是一个阶段读一个指令。

[1] 取指令

IorD: Instruction or Data，当输入0时为读指令，输入1时为写数据

```
IorD = 0    #PC读指令
MemoryRead = 1 #读存储器
IRWrite = 1   #指令存放到寄存器中
```

[2] PC+=4

```
ALUSrcA = 0   #接PC而不是接Rdata1
ALUSrcB = 01   #4
ALUop = 00 #加运算
PCSource = 00   #回写指令
PCWrite = 1   #?
```

Step 2：可以读出寄存器的值

Step 3：进行运算

```
ALUSrcA = 1  #数据来源于寄存器
ALUSrcB = 00 #数据来源于寄存器
ALUop = 10   #R类型指令操作码
```

Step 4： 回写

```
RegDst = 1  #回写寄存器
MemtoReg = 0 #回写数据
RegWrite = 1 #配置
```

LW 指令

Step 1: 取指令

1 取指令

```
IorD = 0   #PC读指令
MemoryRead = 1
IRWrite = 1  #指令存放到寄存器中
```

2 PC+=4

```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 01  #4
PCSource = 00  #回写指令
ALUop = 00  #加运算
PCWrite = 1  #设定回写PC
```

Step 2: 等待，有了指令后可以直接有立即数但是还没有寄存器，所以要等到第三个时钟

闲着没事不如跟Beq一样计算一下偏移量，这样加着倒不用处理，好处是更有规律了。

```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 11 #扩展移位偏移量
ALUop = 00 #加法运算，将加好以后的结果放在寄存器ALUout中
```

Step 3：寄存器与立即数相加计算地址

```
ALUSrcA = 1  #寄存器
ALUSrcB = 10  #扩展地址(不需要移位)
ALUop = 00  #加法运算
```

Step 4：内存中读数据

```
IorD = 1  #地址从ALUout中来
MemRead = 1  #读内存
```

Step 5：回写

```
RegDst = 0  #写寄存器
MemtoReg = 1 #回写数据来源于MDR
RegWrite = 1 #写
```

Beq 指令

Step 1: 取指令

[1] 取指令

```
IorD = 0   #PC读指令
MemoryRead = 1
IRWrite = 1  #指令存放到寄存器中
```

[2] PC+=4

```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 01  #4
PCSource = 00  #回写指令
ALUop = 00  #加运算
PCWrite = 1  #?
```

Step 2: 从寄存器中取数据，并计算地址偏移量

[1] 取数据

[2] 计算偏移量

```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 11 #扩展移位偏移量
ALUop = 00 #加法运算，将加好以后的结果放在寄存器ALUout中
```

Step 3：减法

```
ALUSrcA = 1  #寄存器A
ALUSrcB = 00 #寄存器B
ALUop = 01 #减法运算
PCWriteCond = 1  #Branch与门
PCSource = 01 #数据来源于寄存器ALUout
```

J 指令

Step 1: 取指令

[1] 取指令

```
IorD = 0   #PC读指令
MemoryRead = 1
IRWrite = 1  #指令存放到寄存器中
```

[2] PC+=4

```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 01  #4
PCSource = 00  #回写指令
ALUop = 00  #加运算
PCWrite = 1  #?
```

Step 2: 为了规律性J只能很冤地再白走一步计算偏移量

```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 11 #扩展移位偏移量
ALUop = 00 #加法运算，将加好以后的结果放在寄存器ALUout中
```

Step 3：跳转

```
PCSource = 10  #移位送指令
PCWrite = 1
```

Addi 指令

Step 1: 取指令

1 取指令

```
IorD = 0   #PC读指令
MemoryRead = 1
IRWrite = 1  #指令存放到寄存器中
```

2 PC+=4

```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 01  #4
PCSource = 00  #回写指令
ALUop = 00  #加运算
PCWrite = 1  #
```

Step 2: 可以读出寄存器的值

Step 3: 进行运算

```
ALUSrcA = 1  #数据来源于寄存器
ALUSrcB = 10 #扩展立即数
ALUop = 00   #加法
```

Step 4： 回写

```
RegDst = 0  #回写寄存器
MemtoReg = 0 #回写数据
RegWrite = 1 #配置
```

Jal 指令

`$ra=$ra+4; j label`

Step 1: 取指令

1 取指令

```
IorD = 0   #PC读指令
MemoryRead = 1
IRWrite = 1  #指令存放到寄存器中
```

2 PC+=4

```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 01  #4
PCSource = 00  #回写指令
ALUop = 00  #加运算
PCWrite = 1  #?
```

Step 2: $ra += 4

```
ALUSrcA = 1  #接Rdata1
ALUSrcB = 01 #4
ALUop = 00 #$ra+4

R1前加多路选择器，给一个31的选项
RegDst改为4路选择器，给一个31的选项
```

Step 3：回写

```
RegDst = ?  #回写寄存器
MemtoReg = 0 #回写数据
RegWrite = 1 #配置
```

Step 4：跳转

```
PCSource = 10  #移位送指令
PCWrite = 1
```

Bne 指令

Step 1: 取指令

1 取指令

```
IorD = 0   #PC读指令
MemoryRead = 1
IRWrite = 1  #指令存放到寄存器中
```

2 PC+=4

```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 01  #4
PCSource = 00  #回写指令
ALUop = 00  #加运算
PCWrite = 1  #
```

Step 2: 从寄存器中取数据，并计算地址偏移量

1 取数据

2 计算偏移量

```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 11 #扩展移位偏移量
ALUop = 00 #加法运算，将加好以后的结果放在寄存器ALUout中
```

Step 3：减法

```
ALUSrcA = 1  #寄存器A
ALUSrcB = 00 #寄存器B
ALUop = 01 #减法运算
PCWriteCond = 1  #Branch与门
PCSource = 01 #数据来源于寄存器ALUout

增加信号eon? 对zero输出的信号做异或操作
     zero  eon  ans
beq   1     0    1
beq   0     0    0
bne   0     1    1
bne   1     1    0
```

Jalr 指令

```
Jalr $rd,$rs
(1) $rd = PC + 4
(2) PC = $rs
```

op	rs	rt	address
31~25	25~21	21~16	16~0

Step 1: 取指令

1 取指令

```
IorD = 0   #PC读指令
MemoryRead = 1
IRWrite = 1  #指令存放到寄存器中
```

2 PC+=4

```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 01  #4
PCSource = 00  #回写指令
ALUop = 00  #加运算
PCWrite = 1  #
```

Step 2: 从寄存器中取数据，并计算地址偏移量(为了规律性，实际无用)

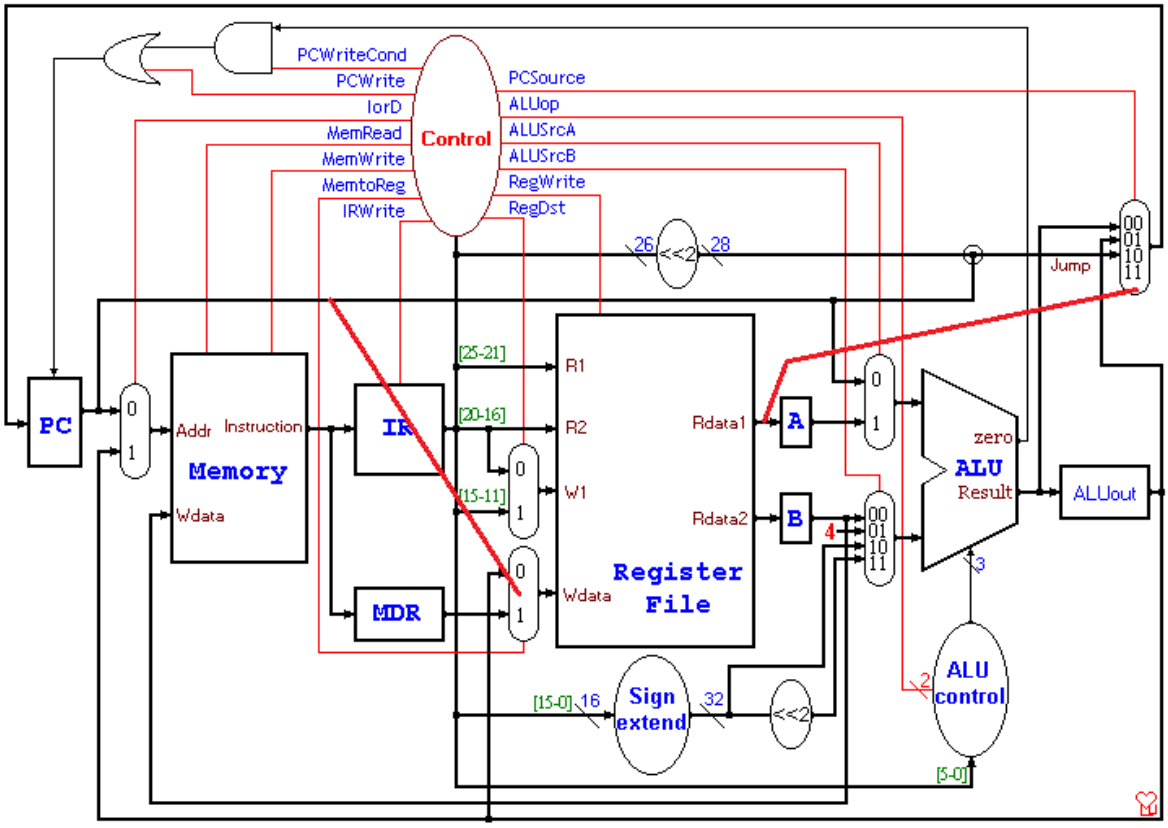
1 取数据

2 计算偏移量

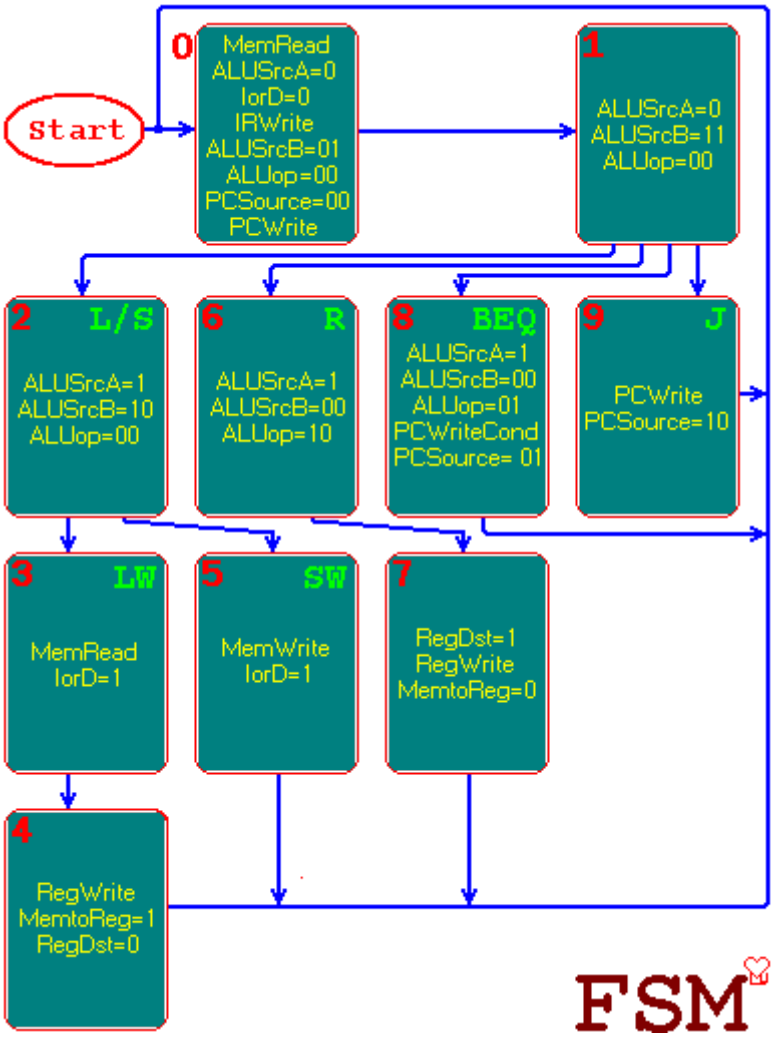
```
ALUSrcA = 0  #接PC而不是接Rdata1
ALUSrcB = 11 #扩展移位偏移量
ALUop = 00 #加法运算，将加好以后的结果放在寄存器ALUout中
```

Step 3：回写

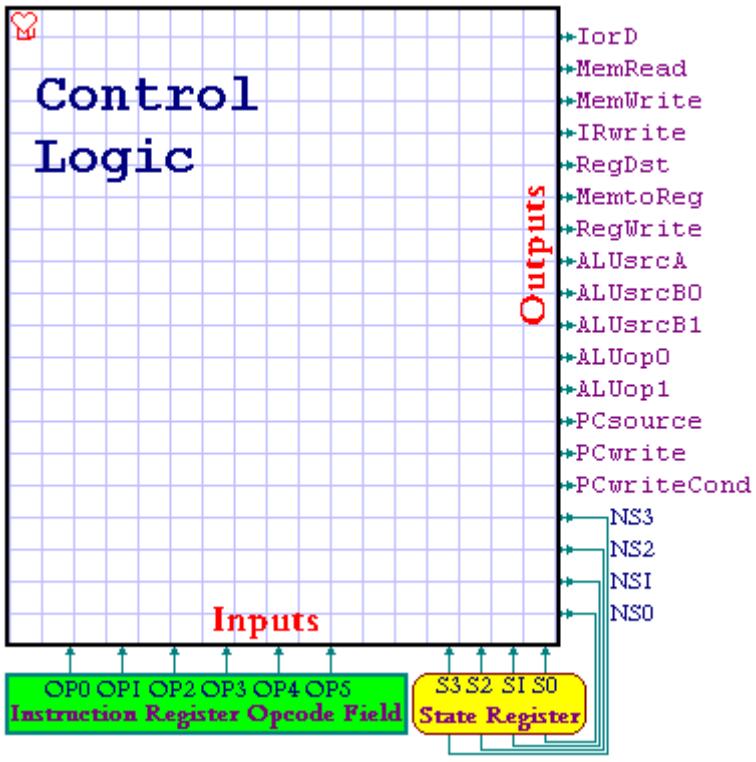
```
RegDst = 0 #第二个寄存器
MemtoReg = 10 #2路改4路后的结果
RegWrite = 1  # $rd = PC + 4
PCSource = 11 # PC = $rs
```



有限状态机Finite State Machine:


Start
0
MemRead
ALUSrcA=0
IorD=0
IRWrite
ALUSrcB=01
ALUop=00
PCSource=00
PCWrite
1
ALUSrcA=0
ALUSrcB=11
ALUop=00
2 L/S
ALUSrcA=1
ALUSrcB=10
ALUop=00
6 R
ALUSrcA=1
ALUSrcB=00
ALUop=10
8 BEQ
ALUSrcA=1
ALUSrcB=00
ALUop=01
PCWriteCond
PCSource= 01
9 J
PCWrite
PCSource=10
3 LW
MemRead
IorD=1
5 SW
MemWrite
IorD=1
7
RegDst=1
RegWrite
MemtoReg=0
4
RegWrite
MemtoReg=1
RegDst=0
FSM

信号控制：


Control
Logic
IorD
MemRead
MemWrite
IRwrite
RegDst
MemtoReg
RegWrite
ALUsrcA
ALUsrcB0
ALUsrcB1
ALUop0
ALUop1
PCsource
PCwrite
PCwriteCond
NS3
NS2
NSI
NS0
Outputs
Inputs
OP0 OP1 OP2 OP3 OP4 OP5
Instruction Register Opcode Field
S3 S2 SI S0
State Register