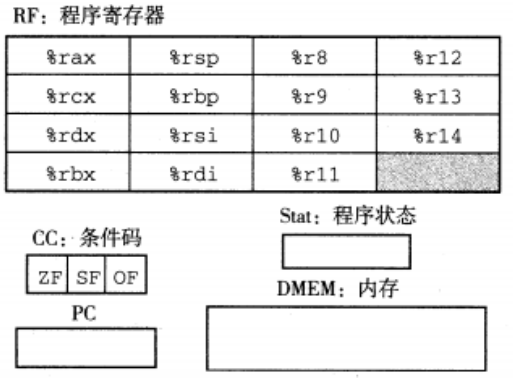
4.处理器体系结构

4.1 Y86-64指令集体系结构

4.1.1 程序员可见的状态



Y86-64有15个程序寄存器（%rax，%rbx...），每个寄存器存储一个64位的字。有3个一位的条件码，ZF,SF和OF。程序计数器PC存放当前正在执行指令的地址。

4.1.2 Y86-64指令

只包含8字节整数操作。

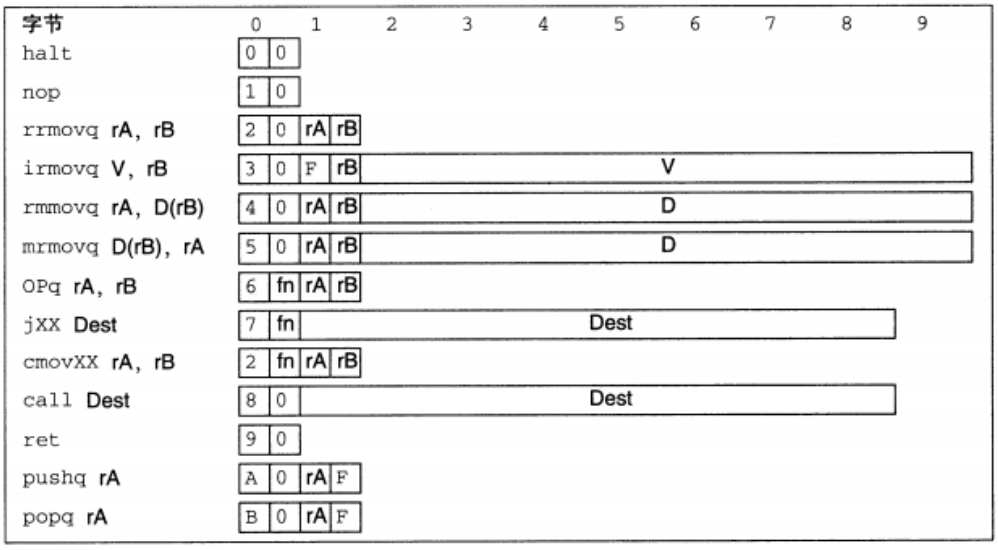


图 4-2 Y86-64 指令集。指令编码长度从 1 个字节到 10 个字节不等。一条指令含有一个单字节的指令指示符，可能含有一个单字节的寄存器指示符，还可能含有一个 8 字节的常数字。字段 fn 指明是某个整数操作(OPq)、数据传送条件(cmovXX)或是分支条件(jXX)。所有的数值都用十六进制表示

注意，mrmovq中操作数顺序是反的!!rB在前rA在后，但是在字节编码表示中是ra在前rb在后
用了一个寄存器，如果另一个寄存器不用，字节编码中就要用F来代替
分支指令和调用指令的目的是一个绝对地址
注意使用在Dest中对齐时是使用符号位扩展!
算术计算只能使用寄存器，不能使用立即数
OPq指令会自动设置状态码
movq指令被分成了四个不同的指令。在地址计算中，不支持第二变址寄存器和任何寄存器值得伸缩值。同样也不允许从一个内存地址直接传送到另一个内存地址。

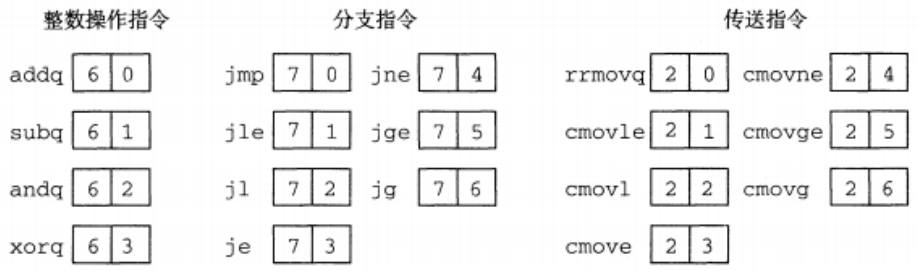


图 4-3 Y86-64 指令集的功能码。这些代码指明是某个整数操作、分支条件还是数据传送条件。这些指令是图 4-2 中所示的 OPq、jXX 和 cmovXX

数字	寄存器名字	数字	寄存器名字
0	%rax	8	%r8
1	%rcx	9	%r9
2	%rdx	A	%r10
3	%rbx	B	%r11
4	%rsp	C	%r12
5	%rbp	D	%r13
6	%rsi	E	%r14
7	%rdi	F	无寄存器

图 4-4 Y86-64 程序寄存器标识符。15 个程序寄存器中每个都有一个相对应的标识符(ID)，范围为 0～0xE。如果指令中某个寄存器字段的 ID 值为 0xF，就表明此处没有寄存器操作数

4.1.4 Y86-64异常

值	名字	含义
1	AOK	正常操作
2	HLT	遇到器执行 halt 指令
3	ADR	遇到非法地址
4	INS	遇到非法指令

图 4-5 Y86-64 状态码。在我们的设计中，任何 AOK 以外的代码都会使处理器停止

4.1.6 一些Y86-64指令的详情

执行pushq %rsp 时，将%rsp原始值压入栈。
执行popq %rsp时，将%rsp置为从栈中读出的值。

4.3 Y86-64的顺序实现

4.3.1 将处理组织成阶段

1.取指：取出下一条要执行的指令并解释，最后计算出下一条PC的值。
2.译码：从寄存器中取出数据。
3.执行：执行算术运算或者逻辑运算，判断跳转指令是否执行。
4.访存：将数据写入内存，或者从内存中取出数据。
5.写回：将数据写回寄存器。
6.更新PC：将PC设置为下一条指令的地址。

不需要读内存的movq**指令**

阶段	OPq rA, rB	rrmovq rA, rB	irmovqV, rB
取指	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC+10
译码	valA ← R[rA] valB ← R[rB]	valA ← R[rA]	
执行	valE ← valB OP valA Set CC	valE ← 0+valA	valE ← 0+valC
访存			
写回	R[rB]← valE	R[rB]← valE	R[rB]← valE
更新 PC	PC ← valP	PC ← valP	PC ← valP

图 4-18 Y86-64 指令 OPq、rrmovq 和 irmovq 在顺序实现中的计算。这些指令计算了一个值，并将结果存放在寄存器中。符号 icode:ifun 表明指令字节的两个组成部分，而 rA:rB 表明寄存器指示符字节的两个组成部分。符号 M_1[x]表示访问(读或者写)内存位置 x 处的一个字节，而 M_8[x]表示访问八个字节

注意立即数和寄存器，尽管没有被加数，执行阶段还是要加0

需要读内存的movq**指令**

阶段	rmmovq rA, D(rB)	mrmovq D(rB), rA
取指	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC+10	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC+10
译码	valA ← R[rA] valB ← R[rB]	 valB ← R[rB]
执行	valE ← valB+valC	valE ← valB+valC
访存	M_8[valE]← valA	valE ← M_8[valE]
写回		
		R[rA]← valM
更新 PC	PC ← valP	PC ← valP

图 4-19 Y86-64 指令 rmmovq 和 mrmovq 在顺序实现中的计算。这些指令读或者写内存

pushq和popq指令

阶段	pushq rA	popq rA
取指	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2
译码	valA ← R[rA] valB ← R[%rsp]	valA ← R[%rsp] valB ← R[%rsp]
执行	valE ← valB+(−8)	valE ← valB+8
访存	M_8[valE]← valA	valE ← M_8[valA]
写回	R[%rsp]← valE	R[%rsp]← valE R[rA]← valM
更新 PC	PC ← valP	PC ← valP

图 4-20 Y86-64 指令 pushq 和 popq 在顺序实现中的计算。这些指令将值压入或弹出栈

在执行阶段将栈指针+或-

popq指令在译码阶段读了两次栈指针的值噢

跳转指令和RET、CALL**指令**

阶段	jXX Dest	call Dest	ret
取指	icode:ifun ← M1[PC] valC ← M8[PC+1] valP ← PC+9	icode:ifun ← M1[PC] valC ← M8[PC+1] valP ← PC+9	icode:ifun ← M1[PC] valP ← PC+1
译码		valB ← R[%rsp]	valA ← R[%rsp] valB ← R[%rsp]
执行	Cnd ← Cond(CC, ifun)	valE ← valB+(−8)	valE ← valB+8
访存		M8[valE] ← valP	valM ← M8[valA]
写回		R[%rsp] ← valE	R[%rsp] ← valE
更新 PC	PC ← Cnd?valC:valP	PC ← valC	PC ← valM

图 4-21 Y86-64 指令 jXX、call 和 ret 在顺序实现中的计算。这些指令导致控制转移

跳转指令在执行阶段计算CND的值，并在更新PC阶段根据CND的值对PC进行赋值
ret也读了两次栈指针，只要是要使用两次栈指针的地方都读两次，格式对齐

4.3.2 SEQ硬件结构（就是顺序执行的硬件结构）

在SEQ中，所有硬件单元的处理都在一个时钟周期内完成。

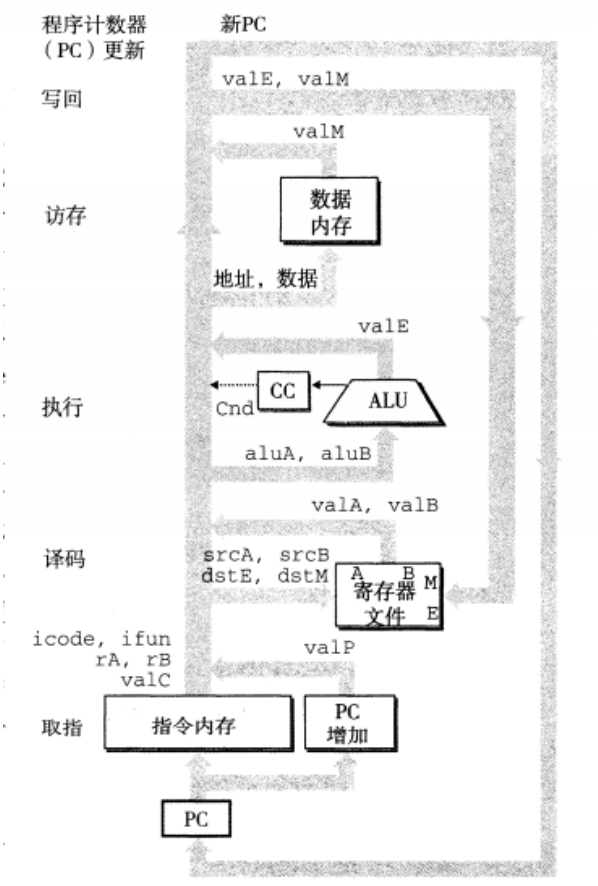


图 4-22 SEQ 的抽象视图，一种顺序实现。指令执行过程中的信息处理沿着顺时针方向的流程进行，从用程序计数器(PC)取指令开始，如图中左下角所示

译码和写回都有两个端口。
原则：从不回读：处理器从来不需要为了完成一条指令的执行而去读由该指令更新了的状态。如pushq %rsp和popq %rsp。所以整个指令系统可以一直顺序执行下去。
每次都是在上升沿更新寄存器或者内存中的数据，而组合逻辑在此之前已经更新过了。

4.3.4 SEQ阶段的实现

设计实现SEQ所需要的控制逻辑块HCL描述。

名称	值(十六进制)	含义
IHALT	0	halt 指令的代码
INOP	1	nop 指令的代码
IRRMOVQ	2	rrmovq 指令的代码
IIRMOVQ	3	irmovq 指令的代码
IRMMOVQ	4	rmmovq 指令的代码
IMRMOVQ	5	mrmovq 指令的代码
IOPL	6	整数运算指令的代码
IJXX	7	跳转指令的代码
ICALL	8	call 指令的代码
IRET	9	ret 指令的代码
IPUSHQ	A	pushq 指令的代码
IPOPQ	B	popq 指令的代码
FNONE	0	默认功能码
RRSP	4	% rsp 的寄存器 ID
RNONE	F	表明没有寄存器文件访问
ALUADD	0	加法运算的功能
SAOK	1	①正常操作状态码
SADR	2	②地址异常状态码
SINS	3	③非法指令异常状态码
SHLT	4	④halt 状态码

图 4-26 HCL 描述中使用的常数值。这些值表示的是指令、功能码、寄存器 ID、ALU 操作和状态码的编码

书本P287-P282

4.4 流水线的通用原理

流水线化的一个重要特性就是提高了系统的吞吐量。
吞吐量=1/（一个组合逻辑+寄存器时间）*1000

4.4.2 流水线操作的详细说明

4.4.3 流水线的局限性

1.不一致的划分
2.流水线过深，收益反而下降

4.4.4 带反馈的流水线系统

4.5 Y86-64的流水线实现

做了小改动，将PC的计算挪到了取值阶段，在各个阶段之间加上流水线寄存器。

4.5.4 预测下一个PC

遇到分支指令时，一般总是预测的选择条件分支，即预测PC的新值为valC。

4.5.5 流水线冒险

1.数据相关造成的数据冒险。
①用暂停来避免数据冒险：每次要把一条指令阻塞在译码阶段，就在执行阶段插入一个气泡。
②用转发来避免数据冒险

③加载/使用数据冒险：遇到读内存指令时不能单纯的用转发来解决，因为内存读在流水线发生的比较晚，就必须插入一个气泡，然后再转发。这种处理方法称为“加载互锁”
④避免控制冒险：控制冒险只会发生在ret指令和跳转指令。ret指令要在M阶段才能读出PC值，所以需要插入3个气泡，直到ret指令执行完M阶段才可以开启下一条指令的F阶段。
跳转指令在预测错误时，插入两个气泡，使跳转后的两条指令消除。

4.5.6 异常处理

由流水线中最深的指令引起的异常，优先级最高。

4.5.7 PIPE各阶段的实现

P308开始

4.5.8 流水线控制逻辑

有四种情况是其他机制不能处理的：
1.加载/使用冒险：暂停一个周期
2.处理ret：暂停3个周期
3.预测错误分支：取消错误预测的指令
4.异常：当异常指令到达写回阶段，停止执行。