北京大学
PEKING UNIVERSITY

ICS讨论班回课

Processor Architecture II:

SEQ: Sequential Implementation

程羽 2019.10.17

- 目的：

  以一个顺序结构实现Y86-64处理器的运行。

内容

- 处理指令阶段的划分
- SEQ硬件结构
- SEQ时序
- 各个阶段的实现

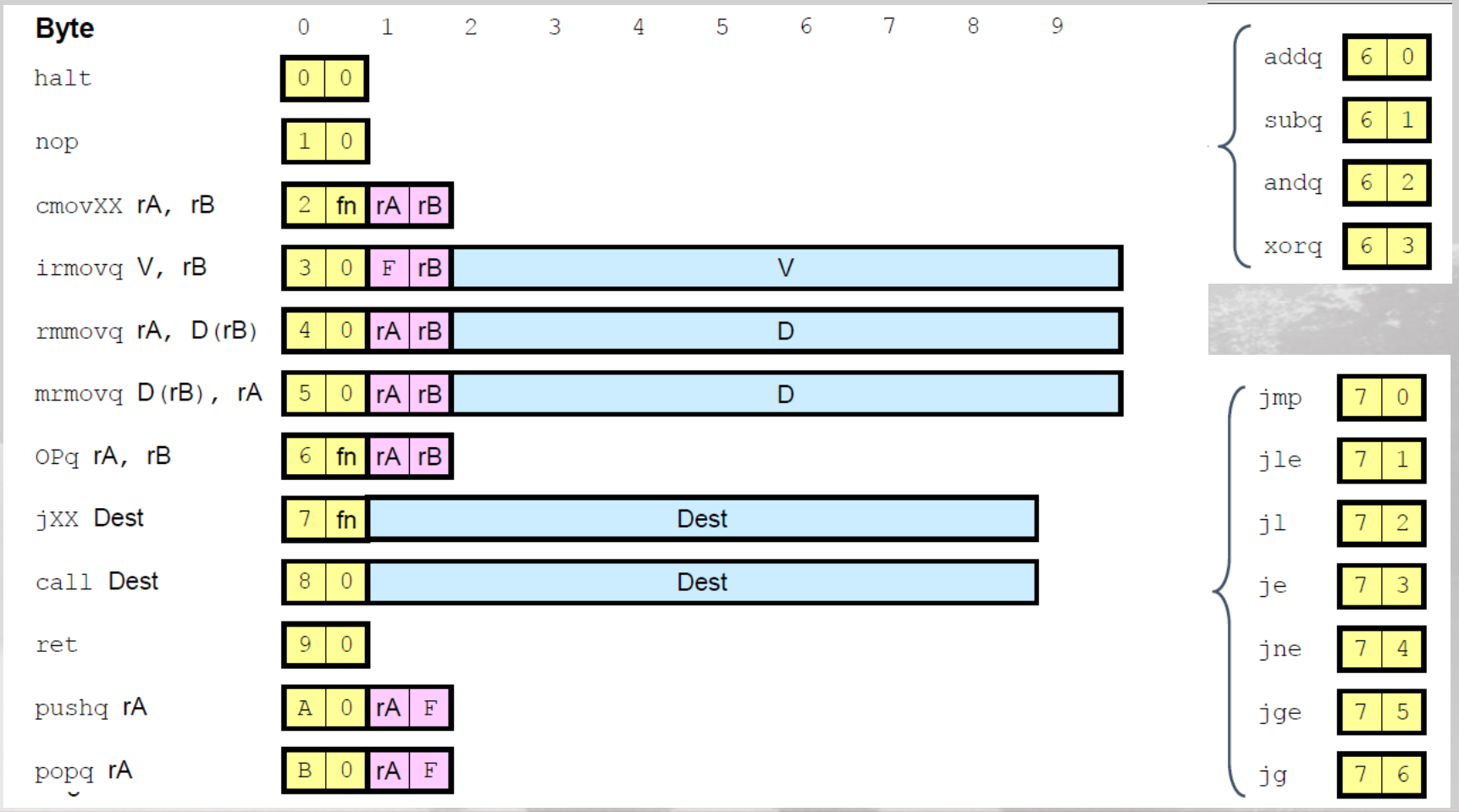
处理指令阶段的划分

- 取指(fetch)
- 译码(decode)
- 执行(execute)
- 访存(memory)
- 写回(write back)
- 更新PC(PC update)

取指(Fetch)

- 按照PC的值，从内存读取指令；
- 指令编码的1个字节分为icode(指令代码)和ifun(指令功能)，分别代表指令的类和小类；
- 之后还可能有寄存器编码或常数；
- 按照指令长度计算下一条指令的pc值valP.

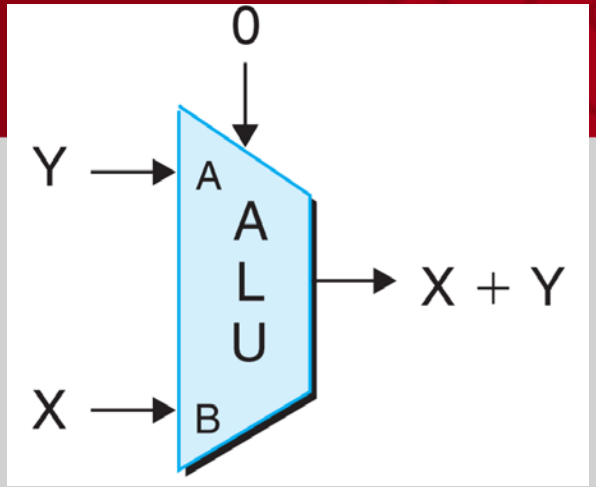
取指(Fetch)


Byte
0 1 2 3 4 5 6 7 8 9
halt 0 0
nop 1 0
cmovXX rA, rB 2 fn rA rB
irmovq V, rB 3 0 F rB V
rmmovq rA, D(rB) 4 0 rA rB D
mrmovq D(rB), rA 5 0 rA rB D
OPq rA, rB 6 fn rA rB
jXX Dest 7 fn Dest
call Dest 8 0 Dest
ret 9 0
pushq rA A 0 rA F
popq rA B 0 rA F
addq 6 0
subq 6 1
andq 6 2
xorq 6 3
jmp 7 0
jle 7 1
jl 7 2
je 7 3
jne 7 4
jge 7 5
jg 7 6

译码(decode)

- 就是从寄存器中取出值，得到valA和valB；
- 最多读入两个操作数；
- 可以读入栈指针%rsp的值；
- 如果操作数为F,则表示不需要寄存器.

执行(execute)



算术/逻辑单元(ALU)根据读出的valA和valB以及选择的ifun进行运算，得到的值记作valE；

可能设置条件码；

如果是传送指令，会检查条件码和传送指令是不是匹配；

如果是跳转指令，会决定是不是要选择分支.

访存(memory)

- 将数据写入内存；
- 从内存中读出数据，得到的值为valM.

写回(write back)

- 将内容写入寄存器中

更新PC(PC update)

将PC设置为下一条指令的地址.

实例

	OPq rA, rB
Fetch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2
Decode	valA ← R[rA] valB ← R[rB]
Execute	valE ← valB OP valA Set CC
Memory	
Write back	R[rB] ← valE
PC update	PC ← valP

- 注意：
- M1表示从内存中取出一个字节的内容；
- 执行的时候valB都写在valA之前.

实例

	rmmovq rA, D(rB)
Fetch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC+10
Decode	valA ← R[rA] valB ← R[rB]
Execute	valE ← valB + valC
Memory	M_8[valE] ← valA
Write back	
PC update	PC ← valP

实例

Stage	pushq rA
Fetch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC + 1]
	valP ← PC + 2
Decode	valA ← R[rA] valB ← R[%rsp]
Execute	valE ← valB + (−8)
Memory	M_8[valE] ← valA
Write back	R[%rsp] ← valE
PC update	PC ← valP

- Push和pop操作指令都只有两个字节；
- 在decode阶段会读入栈指针%rsp的值;
- 执行阶段计算的是栈指针变化后的值；
- 在我们的对Y86-64系统的假设中，push操作先将栈指针-8再存入数据，而在x86-64系统中，真实情况是相反的(见练习题4.7).
- 即，真实情况下push %rsp压入栈中的是改变前的栈指针值.

实例

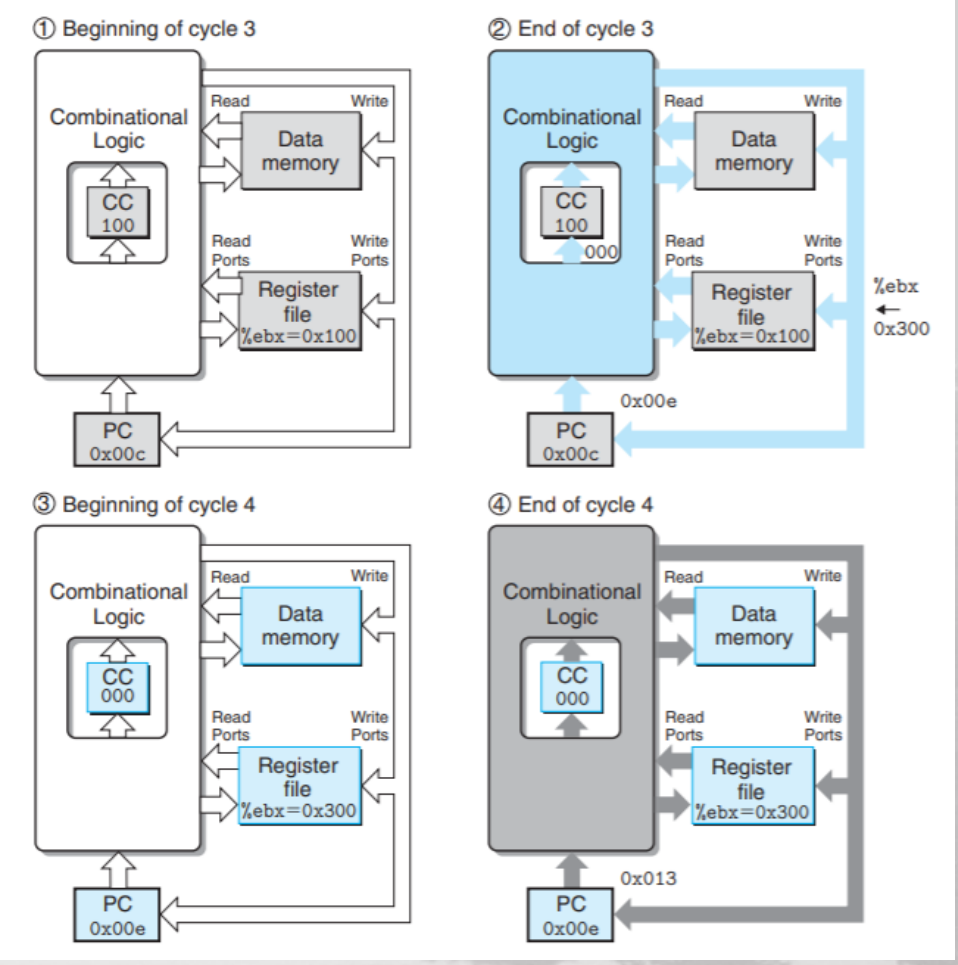
Stage	popq rA
Fetch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC + 1]
	valP ← PC + 2
Decode	valA ← R[%rsp] valB ← R[%rsp]
Execute	valE ← valB + 8
Memory	valM ← M_4[valA]
Write back	R[%rsp] ← valE R[rA] ← valM
PC update	PC ← valP

- 在decode阶段，取出两个值valA和valB值都为栈指针的值，保证了系统先pop再对栈指针+8，这与x86-64运行的真实情况相符.(见练习题4.8)
- 可用popq %rsp验证.
- 在写入阶段E口和M口同时写入数据.

SEQ时序

- 原则：从不回读
- 处理器从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.
- 原因：整个循环是由时钟控制的.

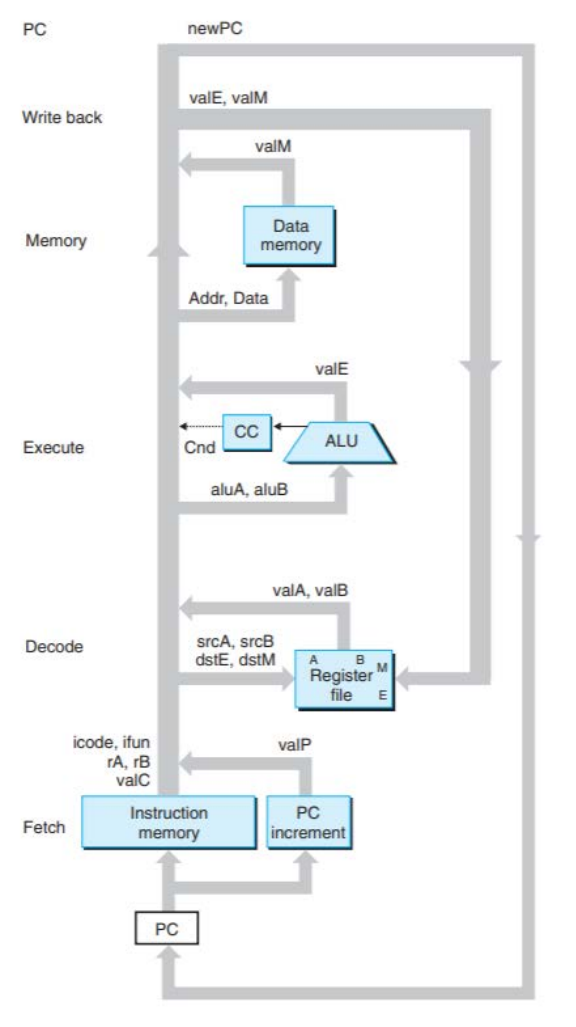
SEQ时序

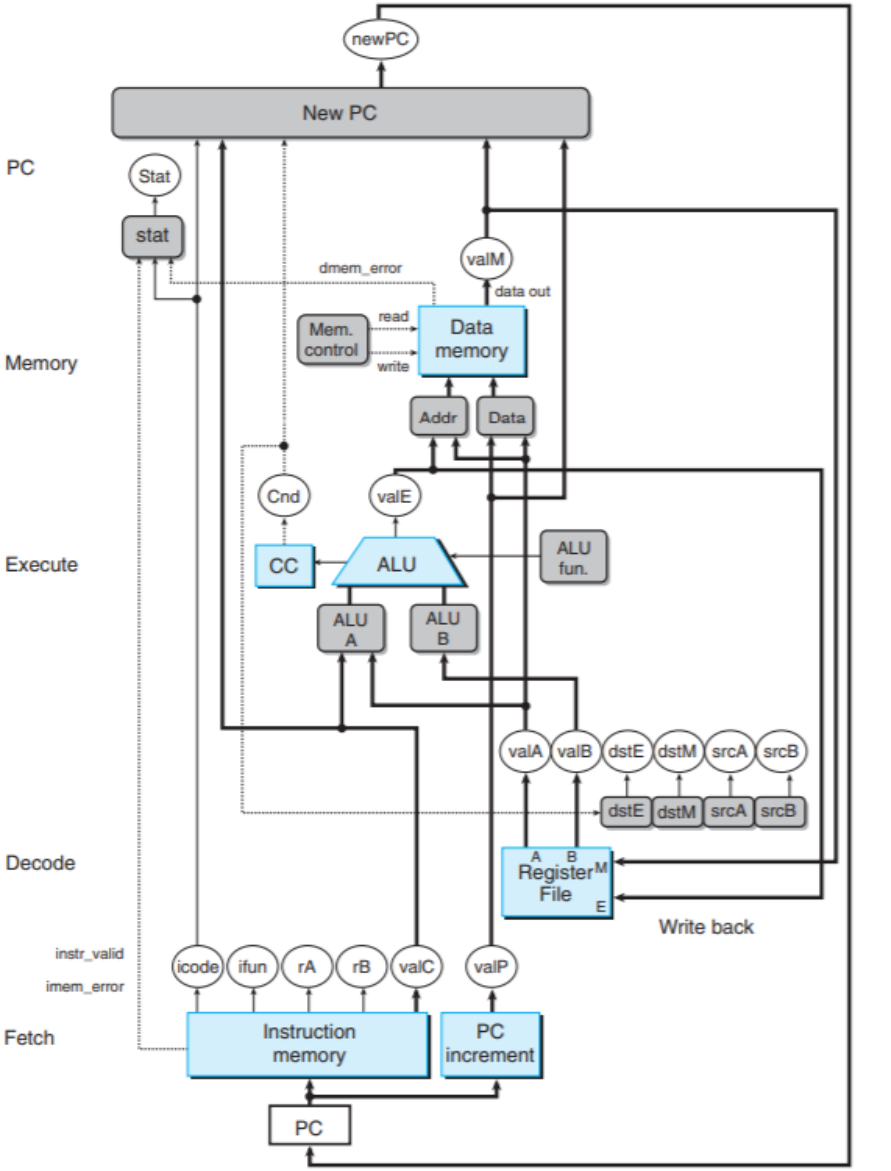


- 组合逻辑和读操作可以认为是依赖于PC并且是“瞬时”的；
- 写操作由时钟控制.

北京大学
PEKING UNIVERSITY

SEQ硬件结构



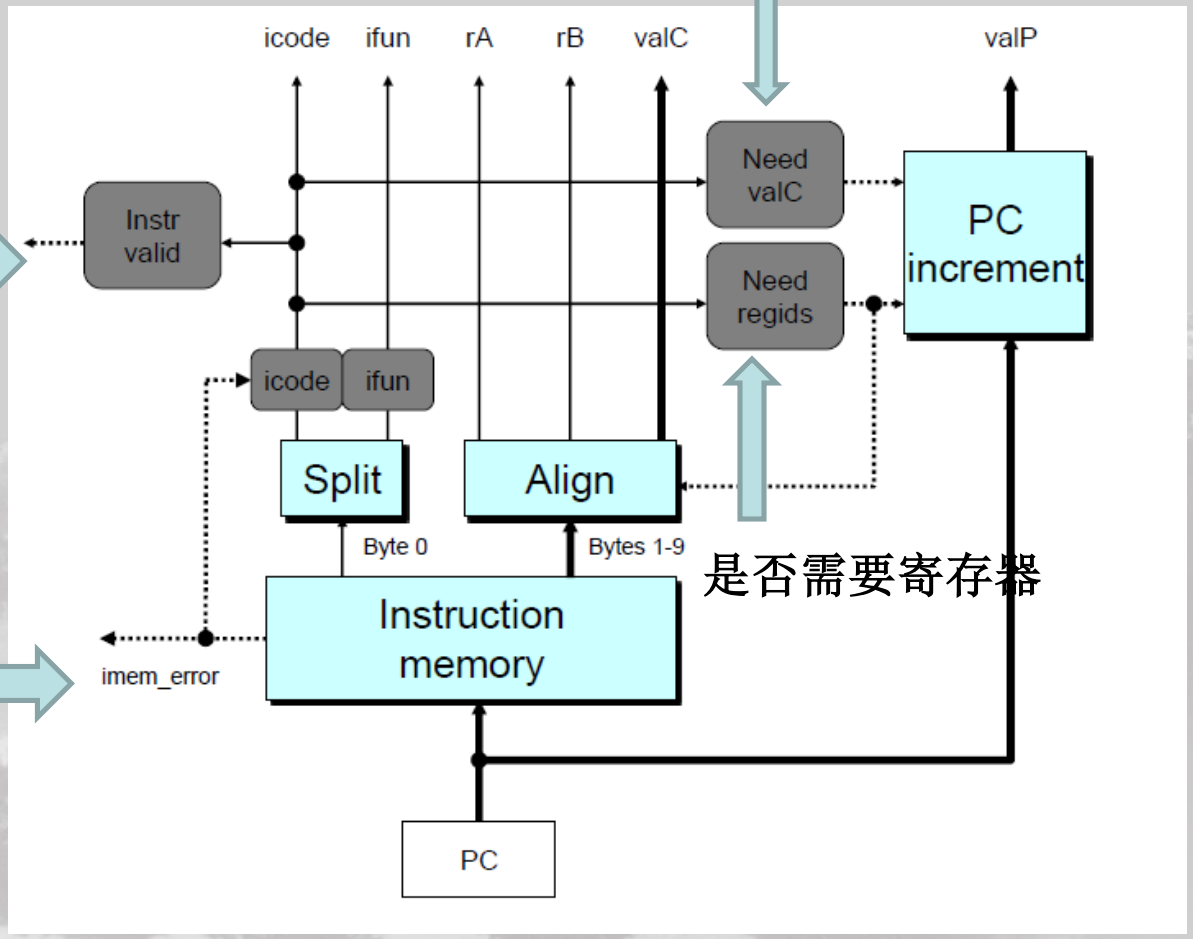


SEQ硬件结构：取指

是否需要常数

判断是否是一个有效指令

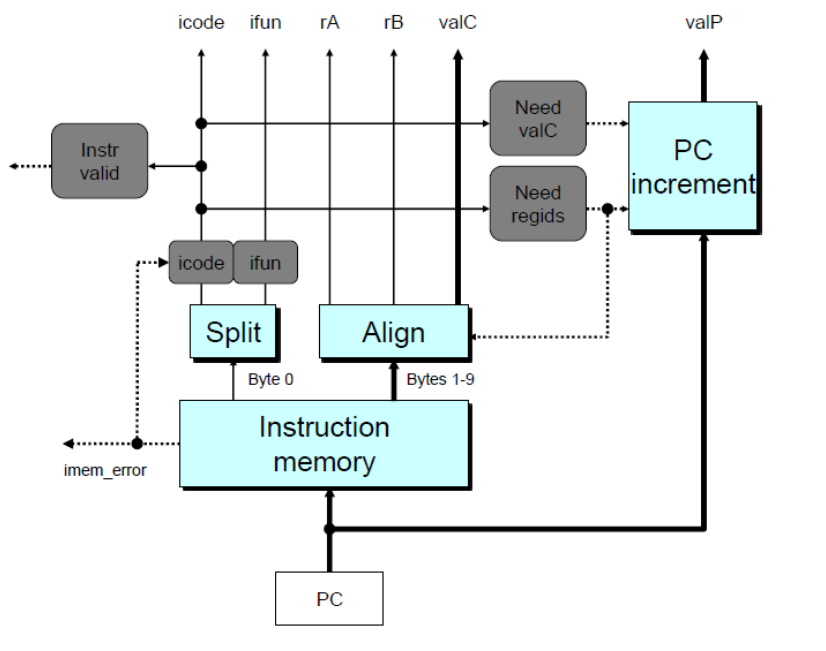
指令地址是否合法，如果不合法，会报告状态寄存器（在下一个访存阶段），同时产生一个**nop**命令.



SEQ硬件结构：取指

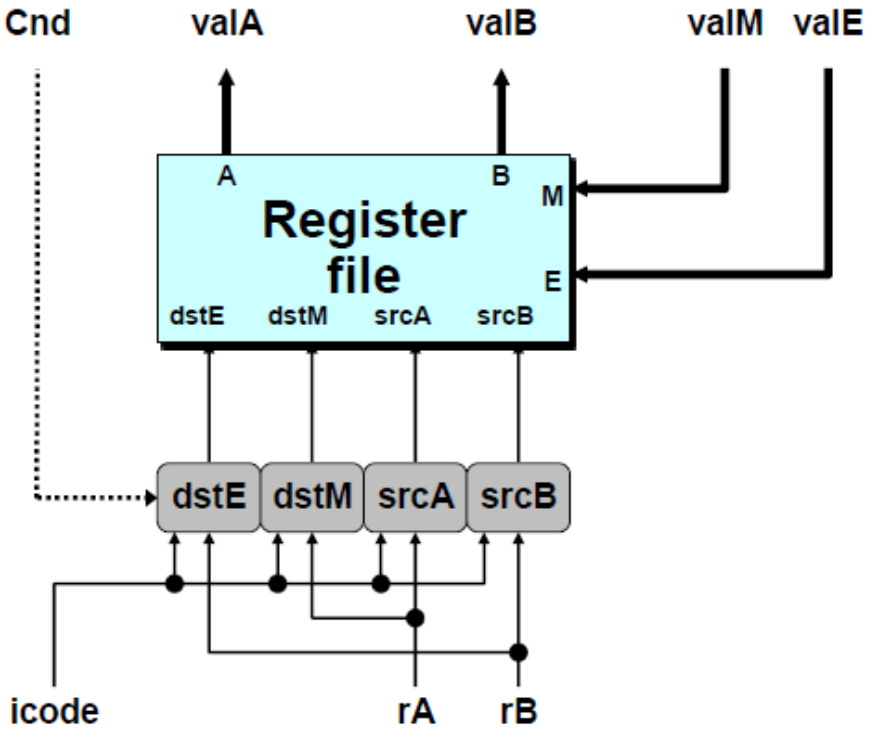
- need_regids的HCL描述：

```
bool need_regids =
        icode in {IRRMOVQ, IOPQ, IPUSHQ, IPOPQ,
                  IIRMOVQ, IRMMOVQ, IMRMOVQ };
```



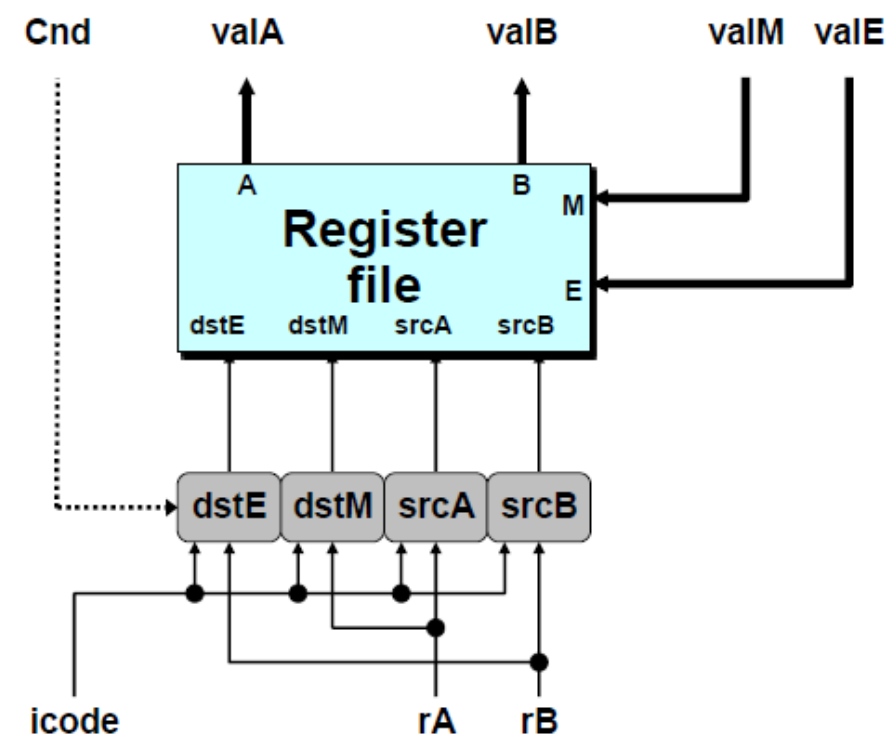
- PC = p
- need_regids = r
- need_valC = i
- valP = p + 1 + r + 8i.

SEQ硬件结构：译码和写回



- 寄存器文件有4个端口：2个读口A,B和2个写口M,E.
- M口可以写入内存中的数据valM，E口可以写入经过ALU运算得到的数据valE.
- 每个端口有一个地址连接和一个数据连接.
- 地址输入：srcA, srcB, dstE, dstM, 表示寄存器的ID，可以输入0xF.

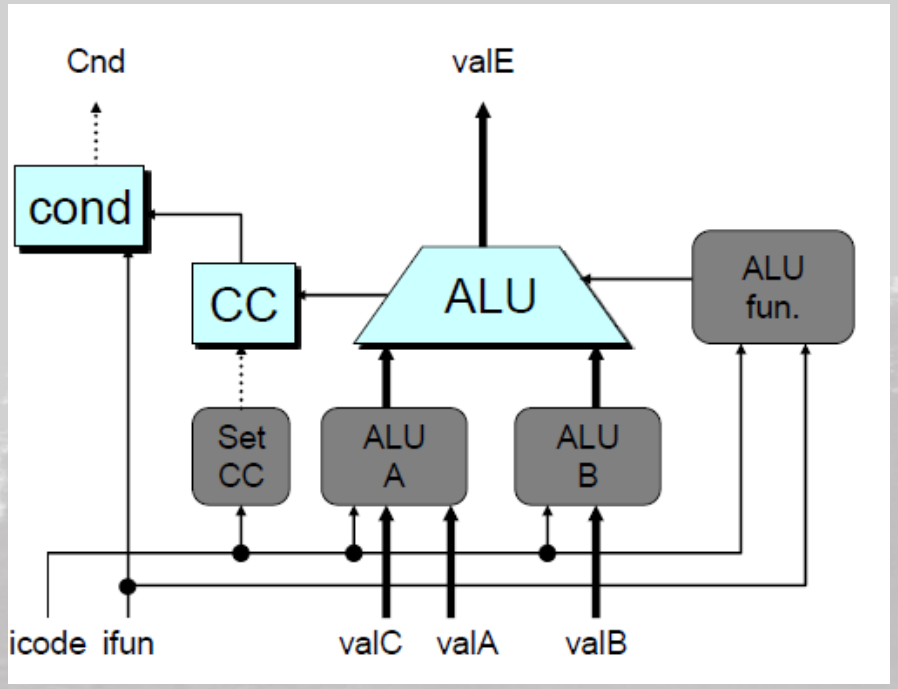
SEQ硬件结构：译码和写回



- 同时在E口和M口写入的例子：popq
- 如果这连个操作都在一个寄存器上发生，会产生冲突.会设置优先级M>E来保证只能写入一个数据。
- 例子：popq %rsp
- Cnd信号由ifun和条件码CC生成
- 会根据Cnd信号判断要不要写入
- 寄存器只有在时钟上升沿时才会写入数据.

SEQ硬件结构：执行

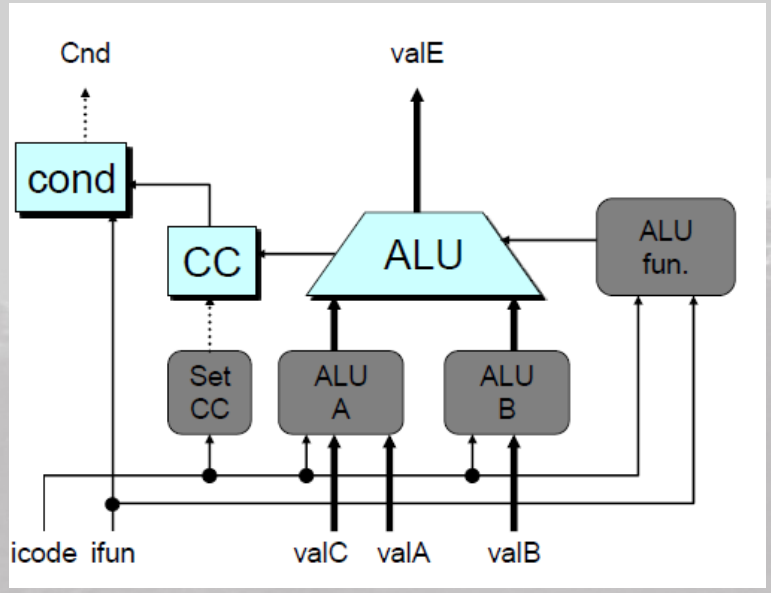
- 做哪种运算由ALU fun.决定
- 计算后会设置条件码CC.
- valA, valB:寄存器中取出的值.
- valC：指令输入的常数值
- ALUA可以是valA,valC,±8等.
- ALUB是valB或0.



SEQ硬件结构：执行

- ALUA的HCL表达：

```
word ALUA = [
        icode in { IRRMOVQ, IOPQ } : valA;
        icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ } : valC;
        icode in { ICALL, IPUSHQ } : -8;
        icode in { IRET, IPOPQ } : 8;
        //Other instructions don't need ALU.
]
```

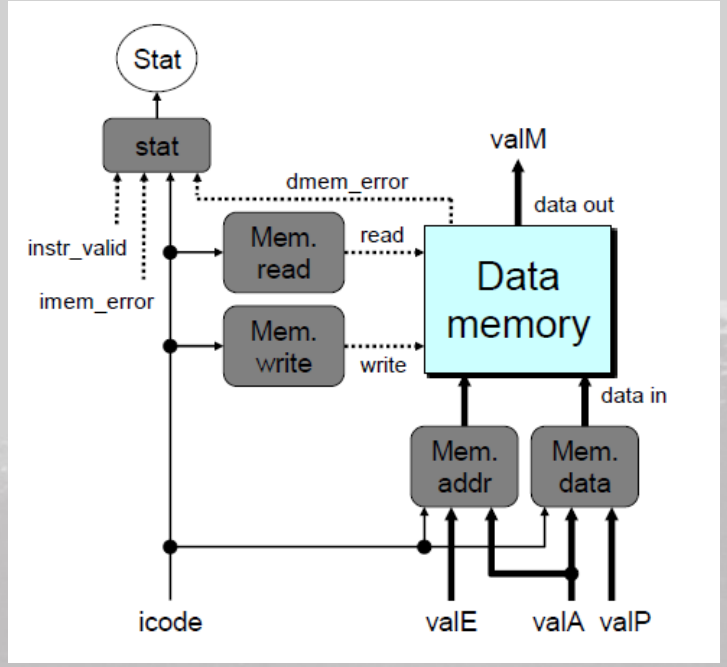


- ALU fun的HCL表达：

```
word alufun = [
        icode == IOPQ : ifun;
        1 : ALUADD;
]
```

SEQ硬件结构：访存

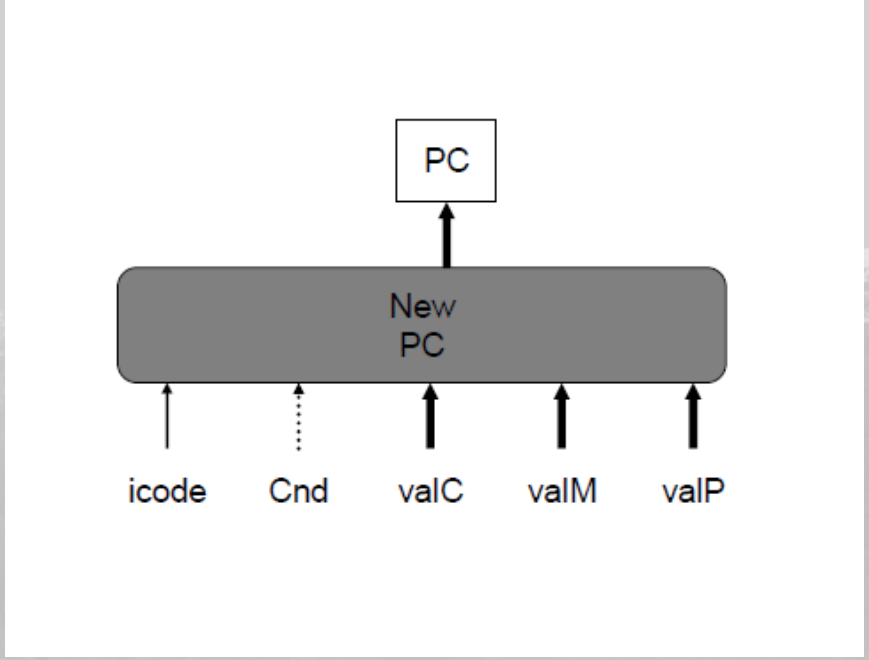
- 4个控制块:
- Mem.read/Mem.write控制读取还是写入；Memaddr控制写入地址，Memdata控制写入数据.
- 地址：valA / valE
- 读取内存：valM
- 写入数据：valA / valP(call)



SEQ硬件结构：更新PC阶段

- 新的PC值可能为：
- valC, valM, valP
- HCL描述：

```
word new_pc = [
    icode == ICALL : valC;
    icode == IJXX && Cnd : ValC;
    icode == IRET : valM;
    1 : valP;
];
```



北京大學
PEKING UNIVERSITY

Thanks!