ICS 第一次回课

Processor Architecture II: SEQ: Sequential Implementation

张致

October 25, 2023

目录

处理器状态和 SEQ 处理器

处理器状态

处理器状态由以下内容构成：

- Program counter register 程序计数器（pc）
- Condition code register 条件代码（CC）
- Register File 寄存器堆
- Memories 内存（指令内存/数据内存）

其中 pc, CC 是时钟存储器，寄存器堆和内存都是随机访问存储器。

SEQ 处理器

为了方便让处理器可以流水线式地去处理指令，我们把指令流分成如下阶段：

- Fetch: 读取指令，判断指令类型，获得 rA,rB,valC（如果有的话），并计算下一条指令地址 valP
- Decode: 通过 rA,rB 从寄存器堆里读取 valA,valB。特别的，当指令为 pop/ret 时，rB 为 R[%rsp]，当指令为 push/call 时，rA,rB 皆为 R[%rsp]。
- Execute: 进行计算，指令中所有的计算都在这一步进行，包括计算地址 (valE, Cnd)
- Memory: 读写内存（valM）
- Write Back: 写寄存器
- PC: 把 pc 设为 valP。特别的，在 call/ret 指令中，pc 被设为了 valC/valM

通过这个过程我们也知道，由于 Memory 在 Execute 之后执行，我们无法拥有如 mrOP 之类的指令。

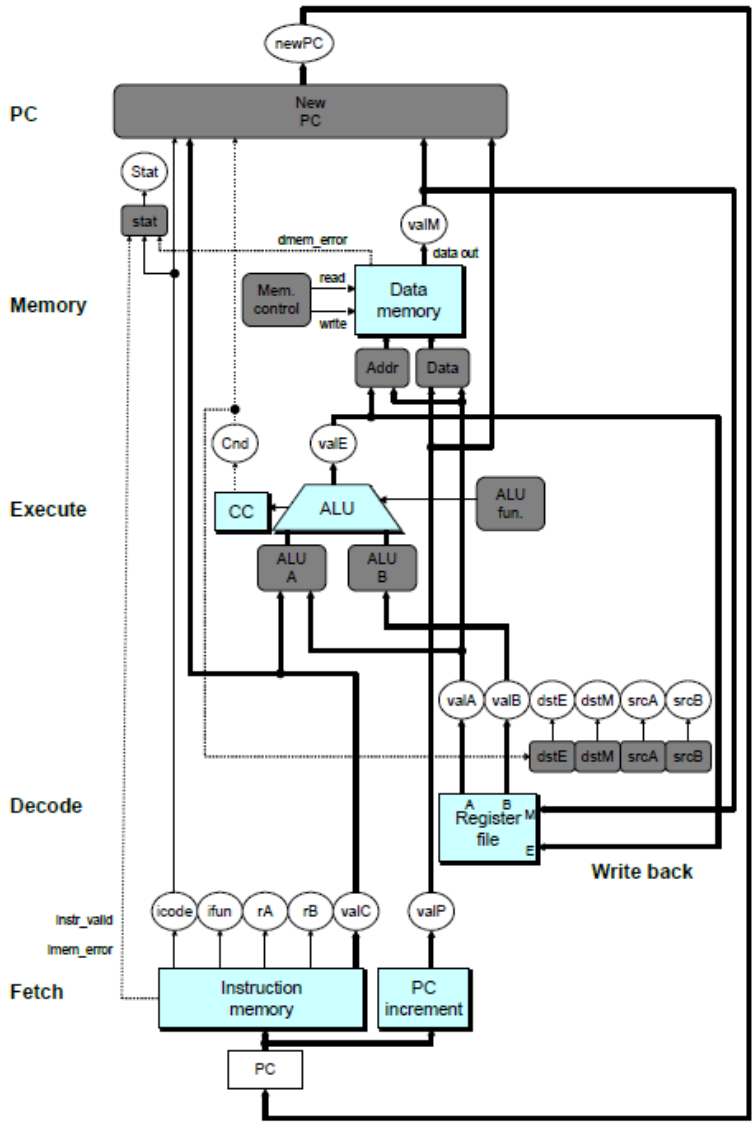
各种指令处理过程

Instruction	Fetch	Decode	Execute	Memory r/w	Writeback	PC update
cmovxx	ifun; rA; rB; valP=PC+2	valA=R[rA] valB=0	valE=valB+valA cnd=Cond(CC,ifun)		if(cnd): R[rB]=valE	PC=valP
irmov	rB; valC; valP=PC+10		valE=0+valC		R[rB]=valE	PC=valP
rmmov	rA; rB; valC; valP=PC+10	valA=R[rA] valB=R[rB]	valE=valB+valC	M_8[valE]=valA		PC=valP
mrmov	rA; rB; valC; valP=PC+10	valB=R[rB]	valE=valB+valC	valM=M_8[valE]	R[rA]=valM	PC=valP
OP	ifun; rA; rB; valP=PC+2	valA=R[rA] valB=R[rB]	valE=valA OP valB Set CC		R[rB]=valE	PC=valP
jxx	ifun; valC; valP=PC+9		cnd=Cond(CC,ifun)			PC=cnd? valC : valP
call	valC; valP=PC+9	valB=R[%rsp]	valE=valB+(-8)	M_8[valE]=valP	R[%rsp]=valE	PC=valC
ret		valA=R[%rsp] valB=R[%rsp]	valE=valB+8	valM=M_8[valA]	R[%rsp]=valE	PC=valM
push	rA; valP=PC+2	valA=R[rA] valB=R[%rsp]	valE=valB+(-8)	M_8[valE]=valA	R[%rsp]=valE	PC=valP
pop	rA; valP=PC+2	valA=R[%rsp] valB=R[%rsp]	valE=valB+8	valM=M_8[valA]	R[%rsp]=valE R[rA]=valM	PC=valP

SEQ 处理器硬件结构

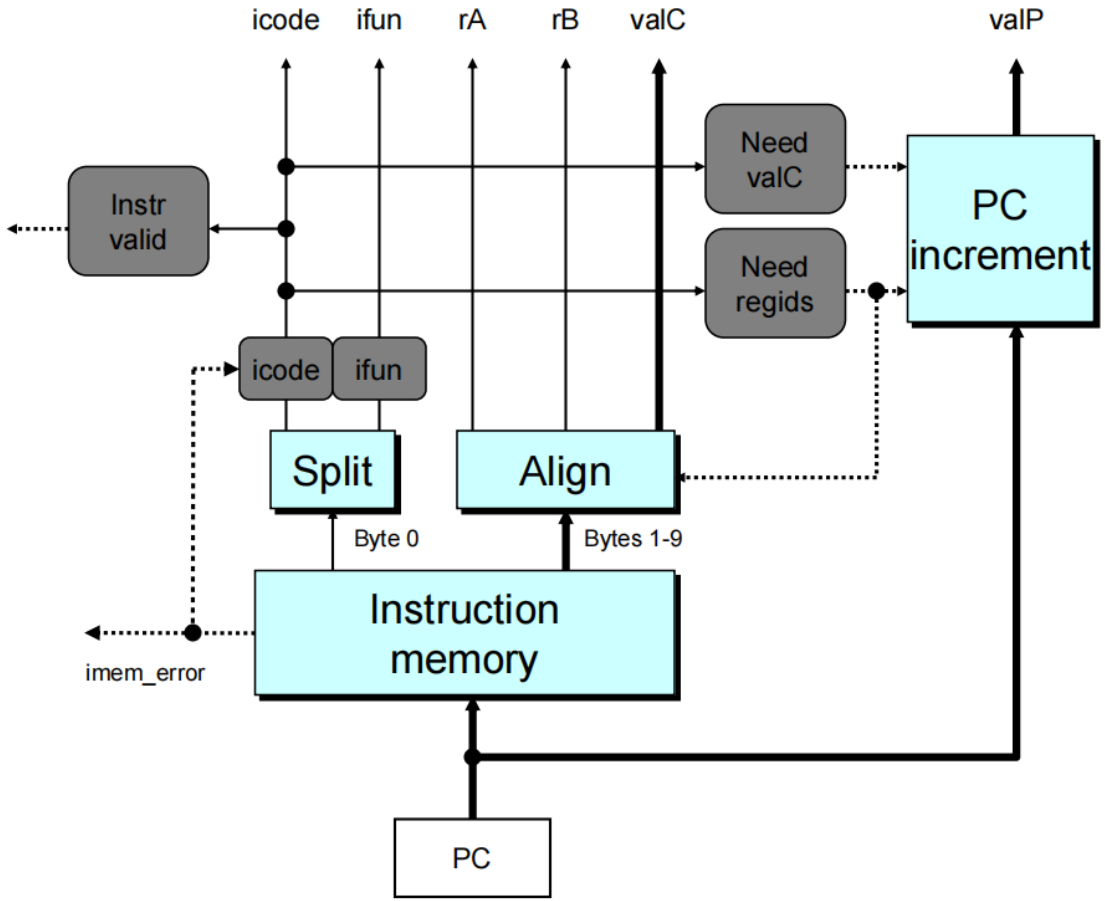
SEQ 处理器

- 蓝色区块是预先设计的硬件组件
- 黑色区块是控制逻辑电路（用 HCL 描述）
- 白色区块是变量标号
- 粗线路传输 64 位，细线路传输 4/8 位，虚线传输 1 位



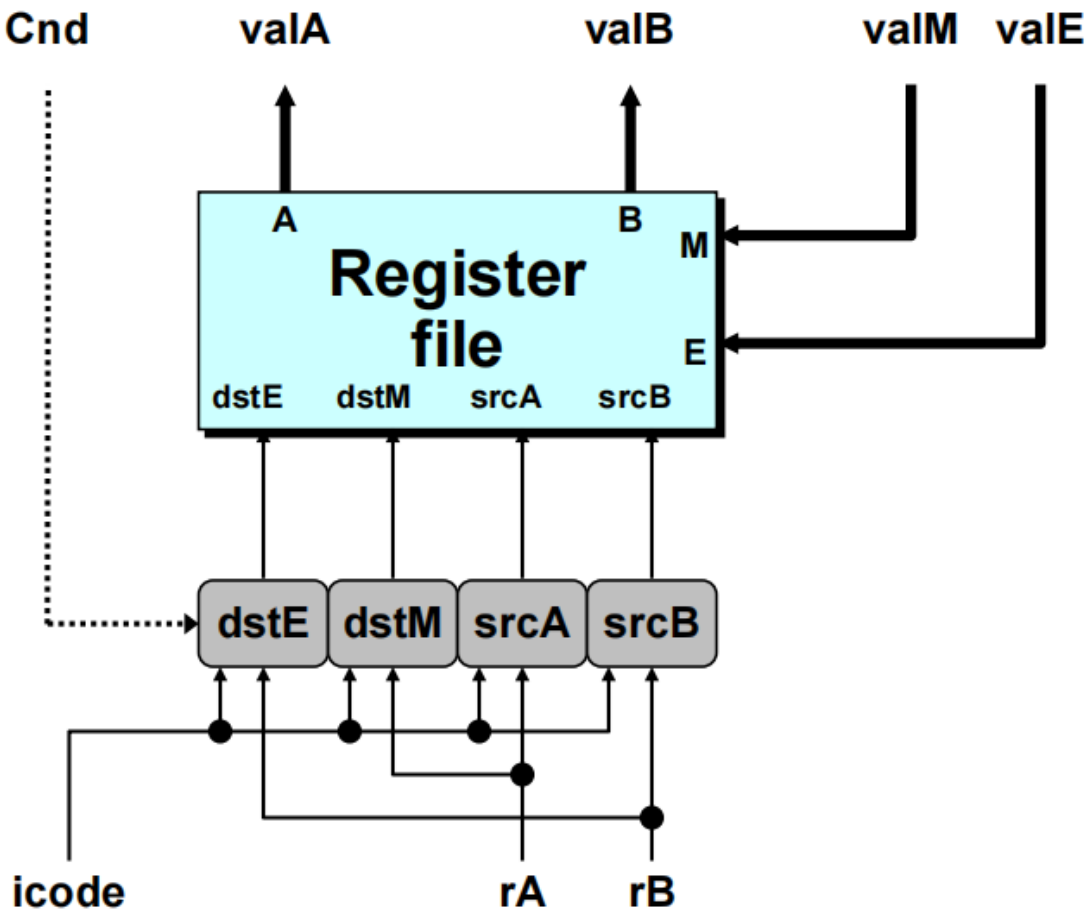
Fetch

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



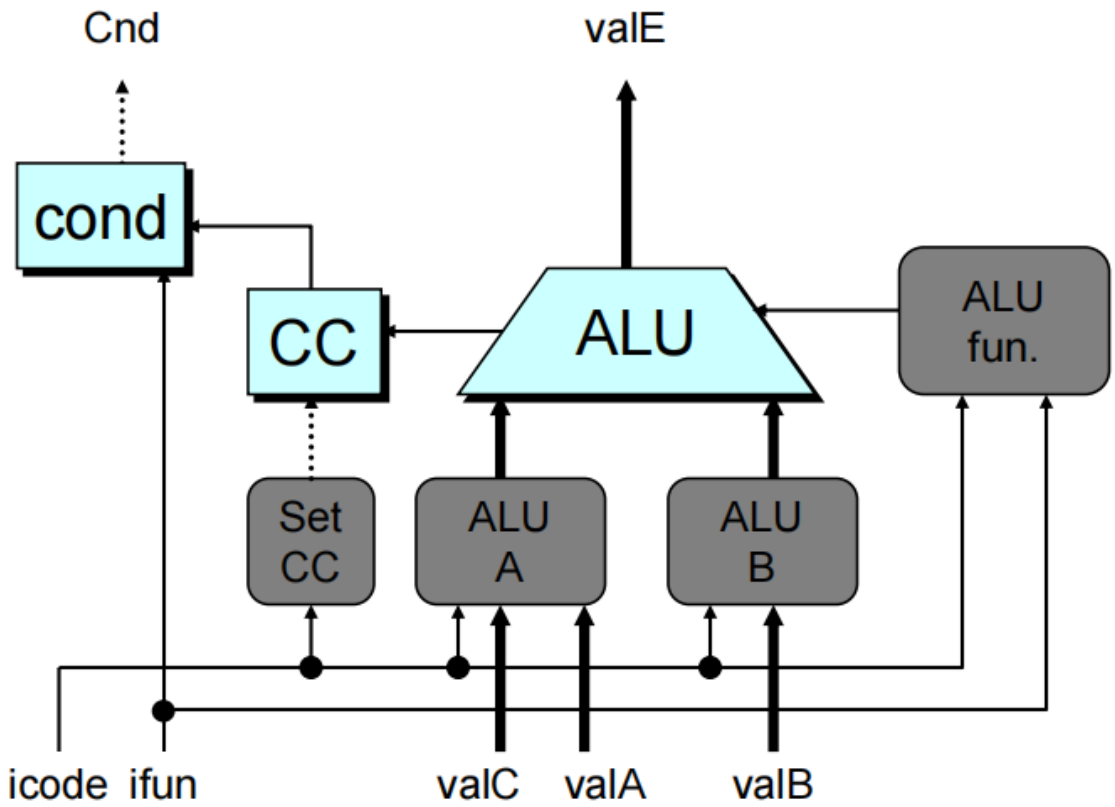
Decode/Write Back

```
word srcA = [
    icode in { IRRMOVQ, IRMMOVQ, IOPQ, IPUSHQ } : rA;
    icode in { IPOPQ, IRET } : RRSP;
    1 : RNONE;
];
word srcB = [
    icode in { IMRMOVQ, IRMMOVQ, IOPQ } : rB;
    icode in { IPOPQ, IRET, IPUSHQ, ICALL } : RRSP;
    1 : RNONE;
];
word dstE = [
    icode in { IRRMOVQ } && Cnd : rB;
    icode in { IIRMOVQ, IOPQ} : rB;
    icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
    1 : RNONE;
];
word dstM = [
    icode in { IPOPQ, IMRMOVQ } : rA;
    1 : RNONE;
];
```



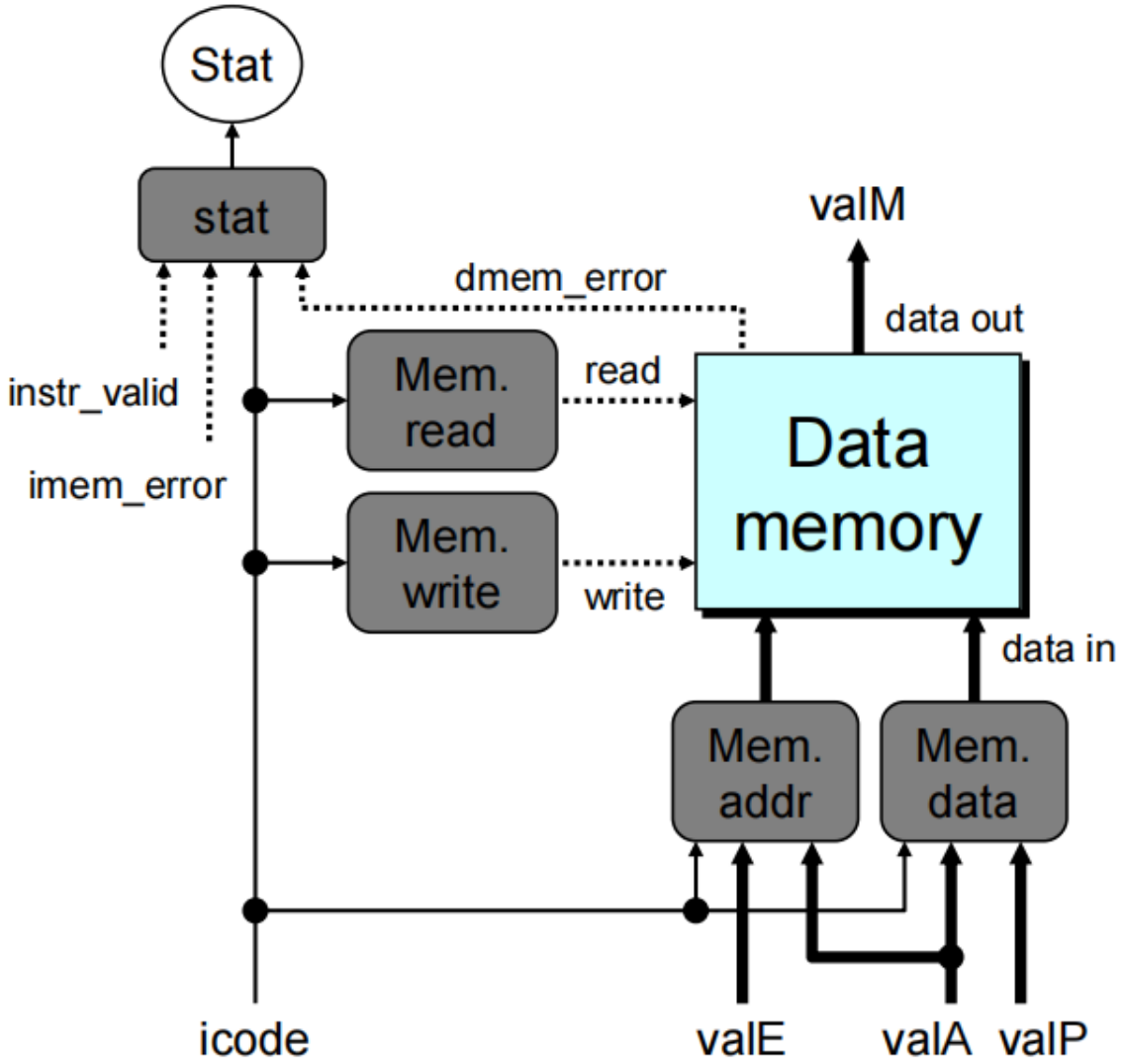
Execute

```
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 dont need ALU
];
word aluB = [
    icode in { IRMMOVQ, IMRMOVQ, IOPQ,
               ICALL, IPUSHQ, IRET, IPOPQ} : valB;
    icode in { IRRMOVQ, IIRMOVQ } : 0;
    // Other instructions dont need ALU
];
int alufun = [
    icode == IOPQ : ifun;
    1 : ALUADD;
];
bool set_cc = icode in { IOPQ };
```



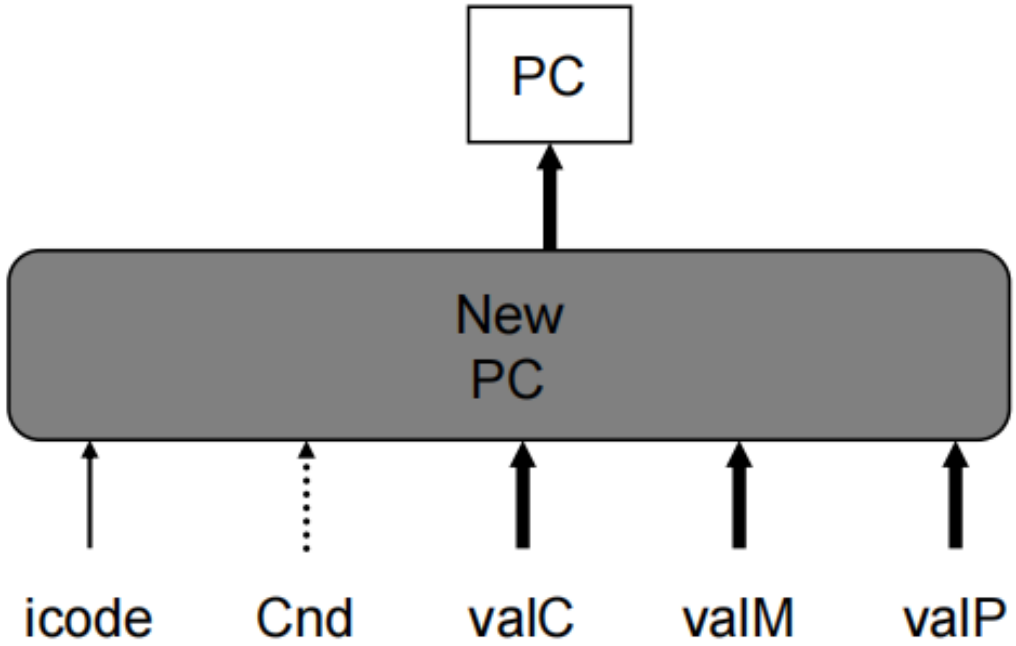
Memory

```
int Stat = [
    imem_error || dmem_error : SADR;
    !instr_valid: SINS;
    icode == IHALT : SHLT;
    1 : SAOK;
];
int mem_addr = [
    icode in { IRMMOVQ, IPUSHQ, ICALL, IMRMOVQ } : valE;
    icode in { IPOPQ, IRET } : valA;
    // Other instructions dont need address
];
int mem_data = [
    icode in { IRMMOVQ, IPUSHQ } : valA;
    icode == ICALL : valP;
    // Other instructions dont need address
];
bool mem_read = icode in { IMRMOVQ, IPOPQ, IRET };
bool mem_write = icode in { IRMMOVQ, IPUSHQ, ICALL };
```



PC

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



一些注意事项

SEQ Operation：在时钟上跳沿，同时处理上条命令的所有写操作，逻辑电路立刻计算出该次操作需要写入的数据在写入端口等着下次上跳沿。

可以发现，在写回时，只会有 R[rA]=valM, R[rB]=valE 的情况（如果我们在 call/ret/push/pop 时把 rB 看作%rsp 的话）

同时，在 Memory 过程中只会执行至多一次读/写。写入时内存地址一定是 valE，读出时内存地址如果需要计算就是 valE，否则就是 valA。

valE 必须由 aluA 和 aluB 经过 ALU 计算获得，不能直接被 valA 或 valB 赋值。aluB 只可能为 0 或 valB。

从不回读原则：处理器不需要为了完成一条指令的执行而去读由该指令更新的状态。

SEQ 实现的缺点：需要保证在一个时钟周期把一条命令所有部分都执行完，一个时钟周期很长。

小题

10. 在 Y86 的 SEQ 实现中，对仅考虑 IRMMOVQ，ICALL，IPOPQ，IRET 指令，对 mem_addr 的 HCL 描述正确的是：

```
word mem_addr = [
    icode in { (1), (2) } : valE;
    icode in { (3), (4) } : valA;
];
```

A. (1) IRMMOVQ (2) IPOPQ (3) IRET (4) ICALL

B. (1) IRMMOVQ (2) IRET (3) IPOPQ (4) ICALL

C. (1) ICALL (2) IPOPQ (3) IRMMOVQ (4) IRET

D. (1) IRMMOVQ (2) ICALL (3) IPOPQ (4) IRET

10. 在 Y86 的 SEQ 实现中，对仅考虑 IRMMOVQ，ICALL，IPOPQ，IRET 指令，对 mem_addr 的 HCL 描述正确的是：

```
word mem_addr = [
    icode in { (1), (2) } : valE;
    icode in { (3), (4) } : valA;
];
```

A. (1) IRMMOVQ (2) IPOPQ (3) IRET (4) ICALL

B. (1) IRMMOVQ (2) IRET (3) IPOPQ (4) ICALL

C. (1) ICALL (2) IPOPQ (3) IRMMOVQ (4) IRET

D. (1) IRMMOVQ (2) ICALL (3) IPOPQ (4) IRET

答案： D

10. Y86 指令 popl rA 的 SEQ 实现如下图所示，其中❶和❷分别为：

Fetch	icode:ifun ← M_1[PC] ra:rb ← M_1 [PC+1] valP ← ❶
Decode	valA ← R[%esp] valB ← R[%esp]
Execute	valE ← ❷
Memory	valM ← M_4 [valA]
Write Back	R[%esp] ← valE R[ra] ← valM
PC Update	PC ← valP

A) PC + 4 valA + 4

B) PC + 4 valA + (- 4)

C) PC + 2 valB + 4

D) PC + 2 valB + (- 4)

10. Y86 指令 **popl rA** 的 SEQ 实现如下图所示，其中❶和❷分别为：

Fetch	icode:ifun ← M_1[PC] ra:rb ← M_1 [PC+1] valP ← ❶
Decode	valA ← R[%esp] valB ← R[%esp]
Execute	valE ← ❷
Memory	valM ← M_4 [valA]
Write Back	R[%esp] ← valE R[ra] ← valM
PC Update	PC ← valP

A) PC + 4 valA + 4

B) PC + 4 valA + (- 4)

C) PC + 2 valB + 4

D) PC + 2 valB + (- 4)

答案：C

请分析 Y86 ISA 中定义的两条指令（cmovXX、call）和一条新加入 Y86 ISA 的 IA32 指令（decl：将操作数减 1）。若在教材所描述的 SEQ 处理器上执行这些指令，请按下表填写每个阶段进行的操作。需说明的信号包括：icode, ifun, rA, rB, valA, valB, valC, valE, valP；the register file R[], data memory M[], Program counter PC, condition codes CC。

注 1、所用到的指令编码为：

cmovXX rA, rB	2	fn	rA	rB	
call Dest	8	0	Dest		
decl rA	C	0	rA	F	

Stage	cmovXX rA, rB	call Dest	decl rA
Fetch			
Decode			
Execute			
Memory			
Write back			
PC update			

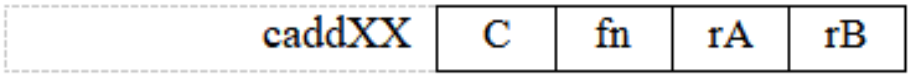
请分析 Y86 ISA 中定义的两条指令（cmovXX、call）和一条新加入 Y86 ISA 的 IA32 指令（decl：将操作数减 1）。若在教材所描述的 SEQ 处理器上执行这些指令，请按下表填写每个阶段进行的操作。需说明的信号包括：icode, ifun, rA, rB, valA, valB, valC, valE, valP；the register file R[], data memory M[], Program counter PC, condition codes CC。

注 1、所用到的指令编码为：

cmovXX rA, rB	2	fn	rA	rB	
call Dest	8	0	Dest		
decl rA	C	0	rA	F	

Stage	cmovXX rA, rB	call Dest	decl rA
Fetch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2	icode:ifun ← M_1[PC] valC ← M_4[PC+1] valP ← PC+5	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2
Decode	valA ← R[rA]	valB ← R[%esp]	valA ← R[rA]
Execute	valE ← 0+valA Cnd ← Cond(CC,ifun)	valE ← valB+(-4)	valE ← valA+(-1) Set CC
Memory	none	M_4[valE] ← valP	none
Write back	if(Cnd) R[rB] ← valE	R[%esp] ← valE	R[rA] ← valE
PC update	PC ← valP	PC ← valC	PC ← valP

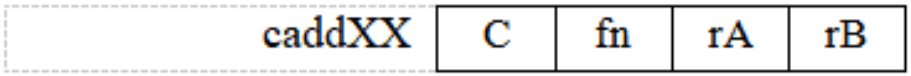
请分析 Y86 ISA 中新加入的一条指令：caddXX，条件加法。其功能可以参考 add 和 cmovXX 两条指令。



若在教材所描述的 SEQ 处理器上执行这条指令，请按下表填写每个阶段进行的操作。需说明的信号包括：icode, ifun, rA, rB, valA, valB, valC, valE, valP, Cnd；the register file R[], data memory M[], Program counter PC, condition codes CC。其中对存储器的引用必须标明字节数。如果在某一阶段没有任何操作，请填写 none 指明。

Stage	caddXX rA, rB
Fetch	
Decode	
Execute	
Memory	
Write back	
PC update	

请分析 Y86 ISA 中新加入的一条指令：caddXX，条件加法。其功能可以参考 add 和 cmovXX 两条指令。



若在教材所描述的 SEQ 处理器上执行这条指令，请按下表填写每个阶段进行的操作。需说明的信号包括：icode, ifun, rA, rB, valA, valB, valC, valE, valP, Cnd；the register file R[], data memory M[], Program counter PC, condition codes CC。其中对存储器的引用必须标明字节数。如果在某一阶段没有任何操作，请填写 none 指明。

Stage	caddXX 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 ← valA+valB Cnd ← Cond(CC,ifun)
Memory	none
Write back	if(Cnd) R[rB] ← valE
PC update	PC ← valP

（1）拟新加入指令leave。其语义相当于

```
rrmovq   %rbp, %rsp
popq     %rbp
```

请按下表补全SEQ处理器每个阶段的操作。需说明的信号可能会包括：icode, ifun, rA, rB, valA, valB, valC, valE, valP, Cnd；寄存器堆R[], 存储器M[], 程序计数器PC, 条件码CC。其中对存储器的引用必须标明字节数。（合计16分，详见下表内）

取指	icode:ifun ← M_1[PC] valP ← PC + 1
译码	
执行	
访存	
写回	
更新PC	PC ← valP

（1）拟新加入指令leave。其语义相当于

```
rrmovq   %rbp, %rsp
popq     %rbp
```

请按下表补全SEQ处理器每个阶段的操作。需说明的信号可能会包括：icode, ifun, rA, rB, valA, valB, valC, valE, valP, Cnd；寄存器堆R[], 存储器M[], 程序计数器PC, 条件码CC。其中对存储器的引用必须标明字节数。（合计16分，详见下表内）

取指	icode:ifun ← M_1[PC] valP ← PC + 1
译码	（每行操作2分，一行全对才得分。共2行，合计4分） **valA ← R[%rbp]** **valB ← R[%rbp]**
执行	（4分，一行全对才得分） **valE ← valB + 8**
访存	（4分，一行全对才得分） **valM ← M_8[valA]**
写回	（每行操作2分，一行全对才得分。共2行，合计4分） **R[%rsp] ← ValE** **R[%rbp] ← ValM**
更新PC	PC ← valP

Thank you!