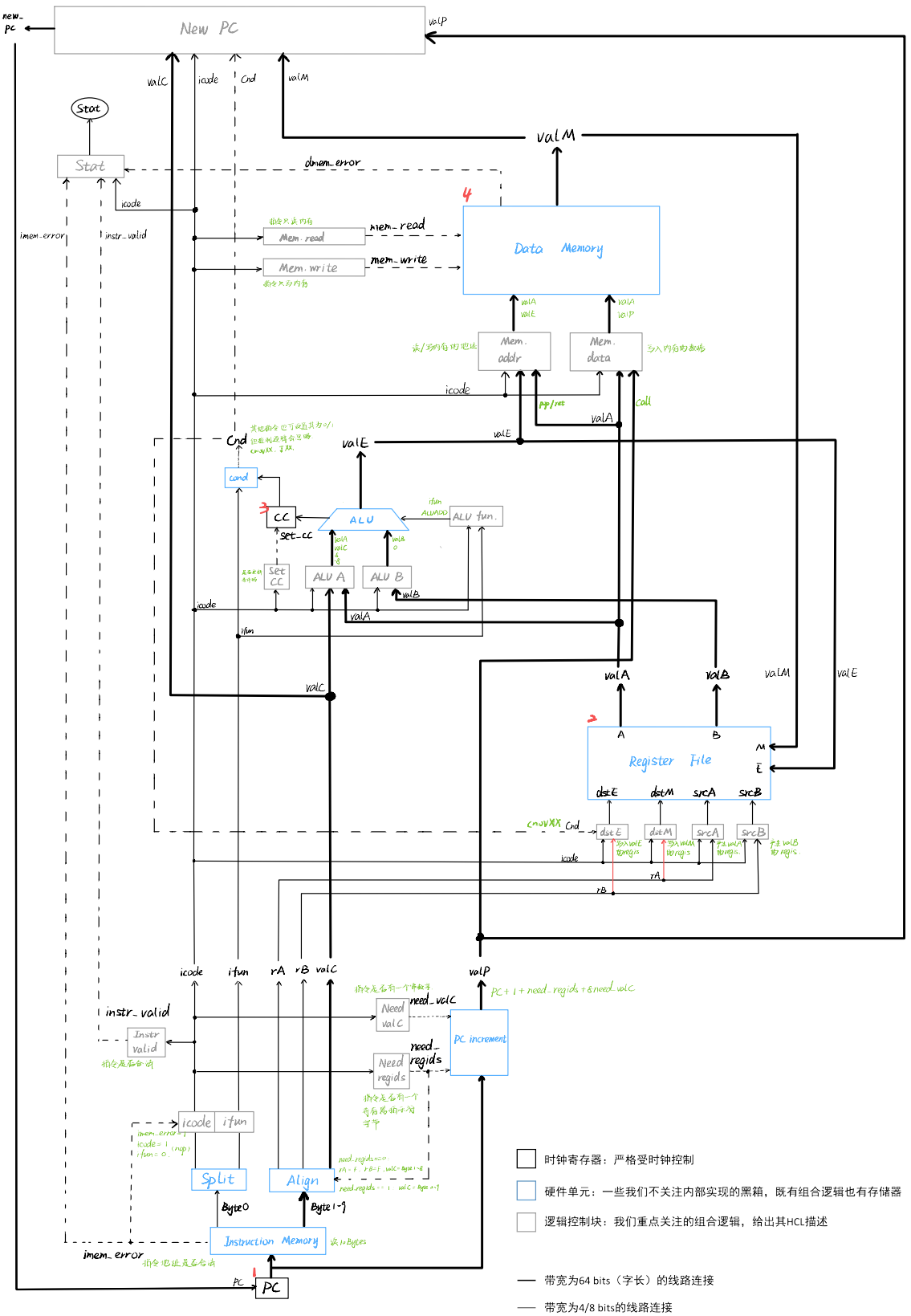
		rrmovq	irmovq	rmmovq	mrmovq	OPq	cmovXX	jXX	call	ret	pushq	popq
Fetch	icode: ifun rA: rB valC valP	icode: ifun ← M1[PC] rA: rB ← M1[PC+1] valP ← PC+2	icode: ifun ← M1[PC] rA: rB ← M1[PC+1] valC ← M8[PC+2] valP ← PC+10	icode: ifun ← M1[PC] rA: rB ← M1[PC+1] valC ← M8[PC+2] valP ← PC+10	icode: ifun ← M1[PC] rA: rB ← M1[PC+1] valC ← M8[PC+2] valP ← PC+10	icode: ifun ← M1[PC] rA: rB ← M1[PC+1] valP ← PC+2	icode: ifun ← M1[PC] rA: rB ← M1[PC+1] valP ← PC+2	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	icode: ifun ← M1[PC] rA: rB ← M1[PC+1] valP ← PC+2	icode: ifun ← M1[PC] rA: rB ← M1[PC+1] valP ← PC+2
Decode	valA, srcA valB, srcB	valA ← R[rA]		valA ← R[rA] valB ← R[rB]	valB ← R[rB]	valA ← R[rA] valB ← R[rB]	valA ← R[rA]		valB ← R[%rsp]	valA ← R[%rsp] valB ← R[%rsp]	valA ← R[rA] valB ← R[%rsp]	valA ← R[%rsp] valB ← R[%rsp]
Execute	valE Cond. codes	valE ← 0+valA	valE ← 0+valC	valE ← valB+valC	valE ← valB+valC	valE ← valB OP valA Set CC	valE ← 0+valA Cnd ← Cond(CC, ifun)	Cnd ← Cond(CC, ifun)	valE ← valB-8	valE ← valB+8	valE ← valB-8	valE ← valB+8
Memory	Read / Write			M8[valE] ← valA	valM ← M8[valE]				M8[valE] ← valP	valM ← M8[valA]	M8[valE] ← valA	valM ← M8[valA]
Write back	E.port, dstE M.port, dstM	R[rB] ← valE	R[rB] ← valE		R[rA] ← valM	R[rB] ← valE	if Cnd R[rB] ← valE		R[%rsp] ← valE	R[%rsp] ← valE	R[%rsp] ← valE	R[%rsp] ← valE R[rA] ← valM
PC update	PC	PC ← valP	PC ← valP	PC ← valP	PC ← valP	PC ← valP	PC ← valP	PC ← Cnd ? valC : valP	PC ← valC	PC ← valM	PC ← valP	PC ← valP



□ 时钟寄存器：严格受时钟控制

□ 硬件单元：一些我们不关注内部实现的黑箱，既有组合逻辑也有存储器

□ 逻辑控制块：我们重点关注的组合逻辑，给出其HCL描述

—— 带宽为64 bits（字长）的线路连接

—— 带宽为4/8 bits的线路连接

---- 带宽为1 bit的线路连接

1 2 3 4：受时钟控制

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

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

word aluA = [
    icode in { IRRMOVQ, IOPQ } : valA;
    icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ } : valC;
    icode in { ICALL, IPUSHQ } : -8;
    icode in { IRET, IPOPQ } : 8;
    # ...
];
word aluB = [
    icode in { IRMMOVQ, IMRMOVQ, IOPQ, ICALL, IRET,
               IPUSHQ, IPOPQ } : valB;
    icode in { IRRMOVQ, IIRMOVQ } : 0;
    #
];
word alufun = [
    icode == IOPQ : ifun;
    1 : ALUADD;
];
bool set_cc = icode == IOPQ;

word mem_addr = [
    icode in { IRMMOVQ, IMRMOVQ, ICALL, IPUSHQ } : valE;
    icode in { IRET, IPOPQ } : valA;
    #
];
word mem_data = [
    icode in { IRMMOVQ, IPUSHQ } : valA;
    icode == ICALL : valP;
    #
];
bool mem_read = icode in { IMRMOVQ, IRET, IPOPQ };
bool mem_write = icode in { IRMMOVQ, ICALL, IPUSHQ };

word stat = [
    imem_error || dmem_error : SADR;
    !instr_valid : SINS;
    icode == IHALT : SHLT;
    1 : SAOK;
];

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