CSAPP:ARCHLAB

Part A

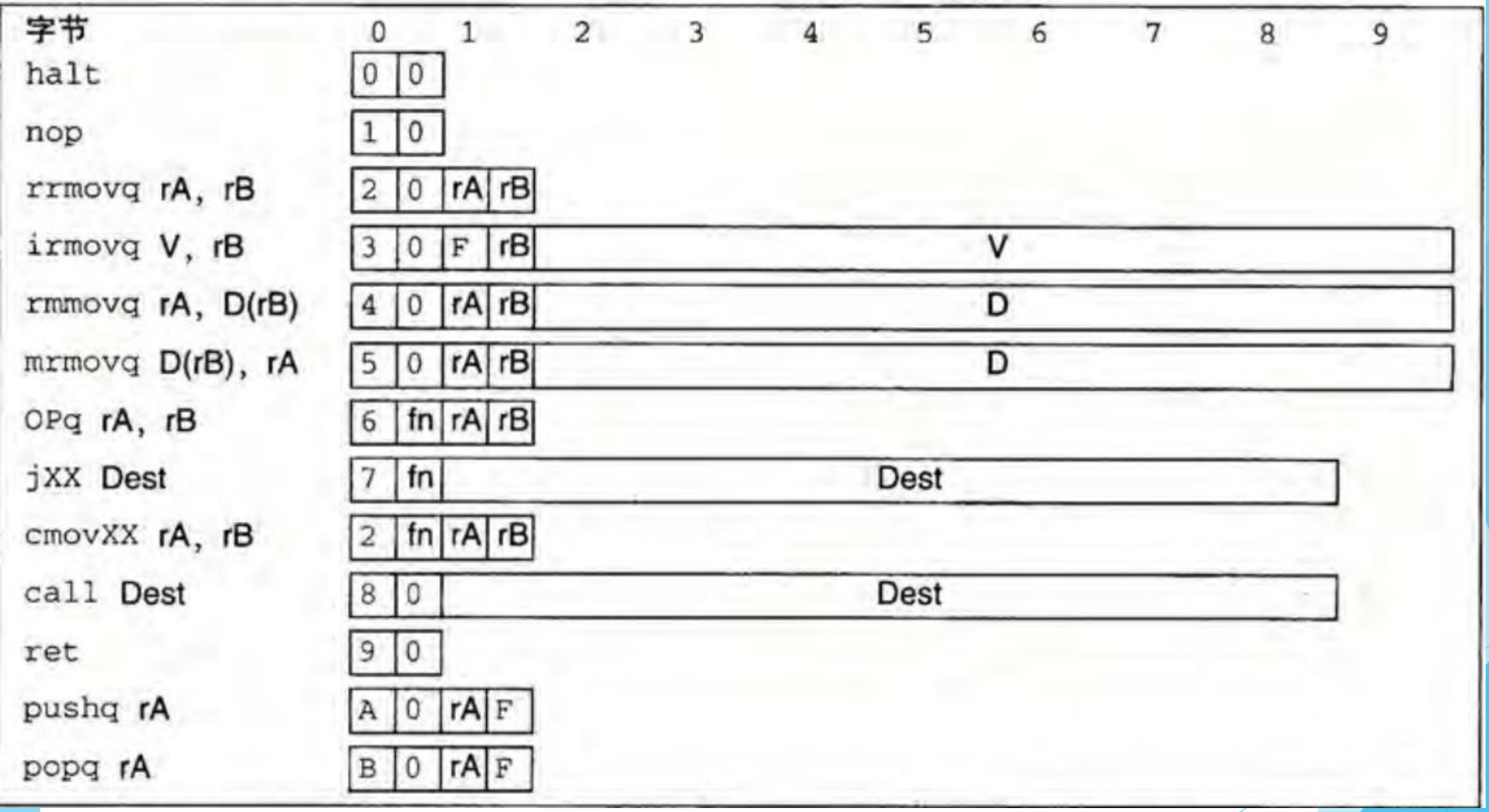
把三个C语言函数翻译成对应的Y86汇编代码

sum_list:
循环累加value

rsum_list:
递归累加value

bubble_sort:
冒泡排序

注：partA中要运行的数据已给出


字节
0 1 2 3 4 5 6 7 8 9
halt
0 0
nop
1 0
rrmovq rA, rB
2 0 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
cmovXX rA, rB
2 fn rA rB
call Dest
8 0 Dest
ret
9 0
pushq rA
A 0 rA F
popq rA
B 0 rA F

- Y86指令集

./y86-code中给出了Y86程序标准示例

```
1   #王清源 2200013187
2   # Execution begins at address 0
3       .pos 0
4       irmovq stack, %rsp      # Set up stack pointer
5       call main       # Execute main program
6       halt            # Terminate program
7
8       .align 8
9       array:
10      ele1:
11              .quad 0x00a
12              .quad ele2
13      ele2:
14              .quad 0x0b0
15              .quad ele3
16      ele3:
17              .quad 0xc00
18              .quad 0
19
```

```
38  # The stack starts here and grows to lower addresses
39      .pos 0x200
40  stack:
41
```

注意：stack：后要换行

- YAS将所写的.ys文件编译成.yo文件
- YIS运行.yo文件

```
u2200013187@icsdancer:~/archlab-handout/sim/misc$ ./yas ans-sum.ys
u2200013187@icsdancer:~/archlab-handout/sim/misc$ ./yis ans-sum.yo
Stopped in 27 steps at PC = 0x13.  Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%rax:   0x0000000000000000      0x0000000000000cba
%rdx:   0x0000000000000000      0x0000000000000c00
%rsp:   0x0000000000000000      0x0000000000000200

Changes to memory:
0x01f0: 0x0000000000000000      0x000000000000005b
0x01f8: 0x0000000000000000      0x0000000000000013
```

Sum_list()

```c
long sum_list(list_ptr ls)
{
    long val = 0;
    while (ls) {
    val += ls->val;
    ls = ls->next;
    }
    return val;
}
```

```
array:
ele1:
        .quad 0x00a
        .quad ele2
ele2:
        .quad 0x0b0
        .quad ele3
ele3:
        .quad 0xc00
        .quad 0
```

```
20  main:
21      irmovq array,%rdi
22      call sum
23      ret
24
25  sum:
26      irmovq $0,%rax
27      irmovq $8,%r8
28      jmp test
29  loop:
30      mrmovq (%rdi),%rdx
31      addq %rdx,%rax
32      mrmovq $8(%rdi),%rdi
33  test:
34      subq %r8,%rdi
35      jne loop
36      ret
37
```

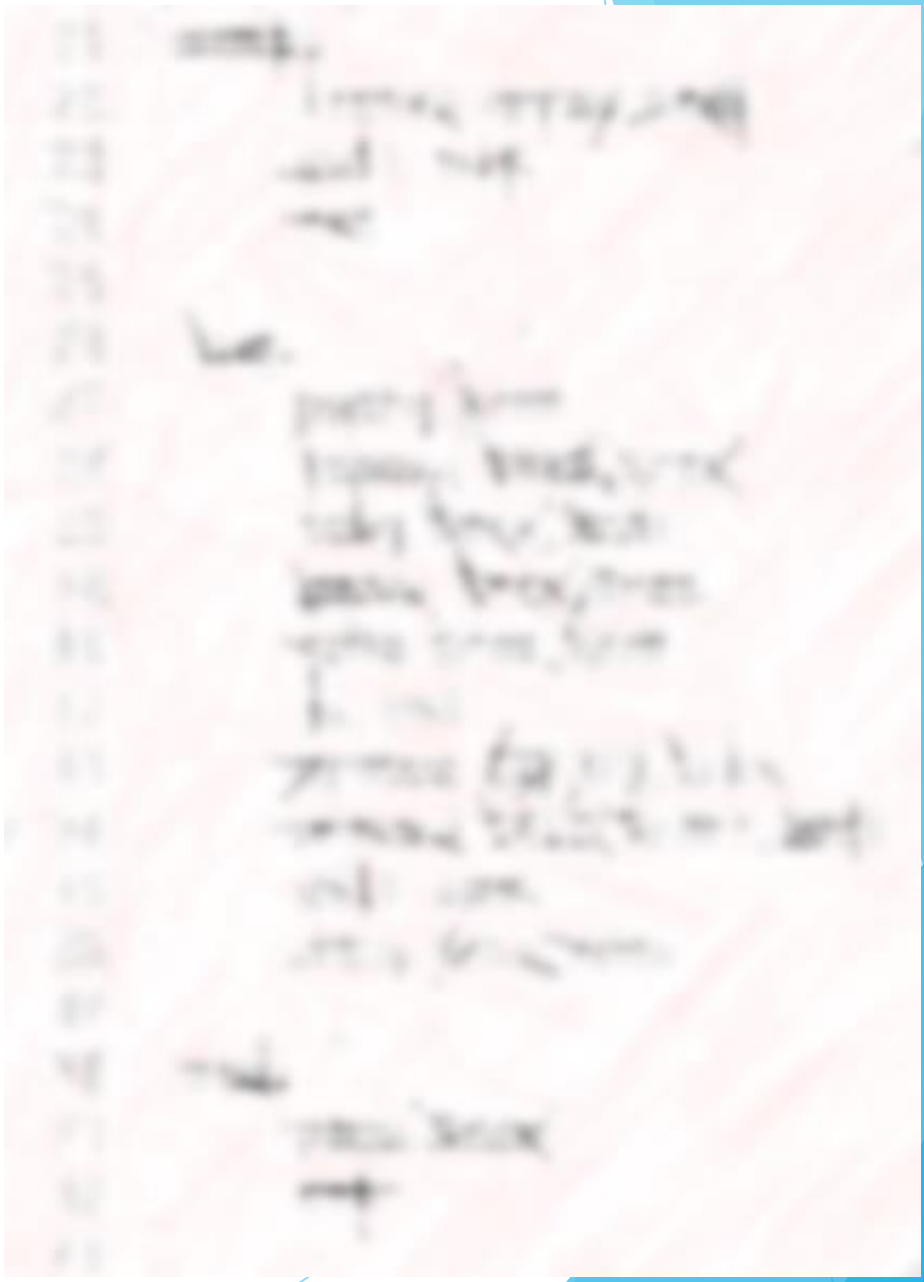
注意：返回0xcba即算正确

Rsum_list()

```c
/* rsum_list - Recursive version of sum_list */
long rsum_list(list_ptr ls)
{
    if (!ls)
    return 0;
    else {
    long val = ls->val;
    long rest = rsum_list(ls->next);
    return val + rest;
    }
}
```

使用%rdi - 0来设置条件码

返回0xcba即算正确



Bubble_sort()

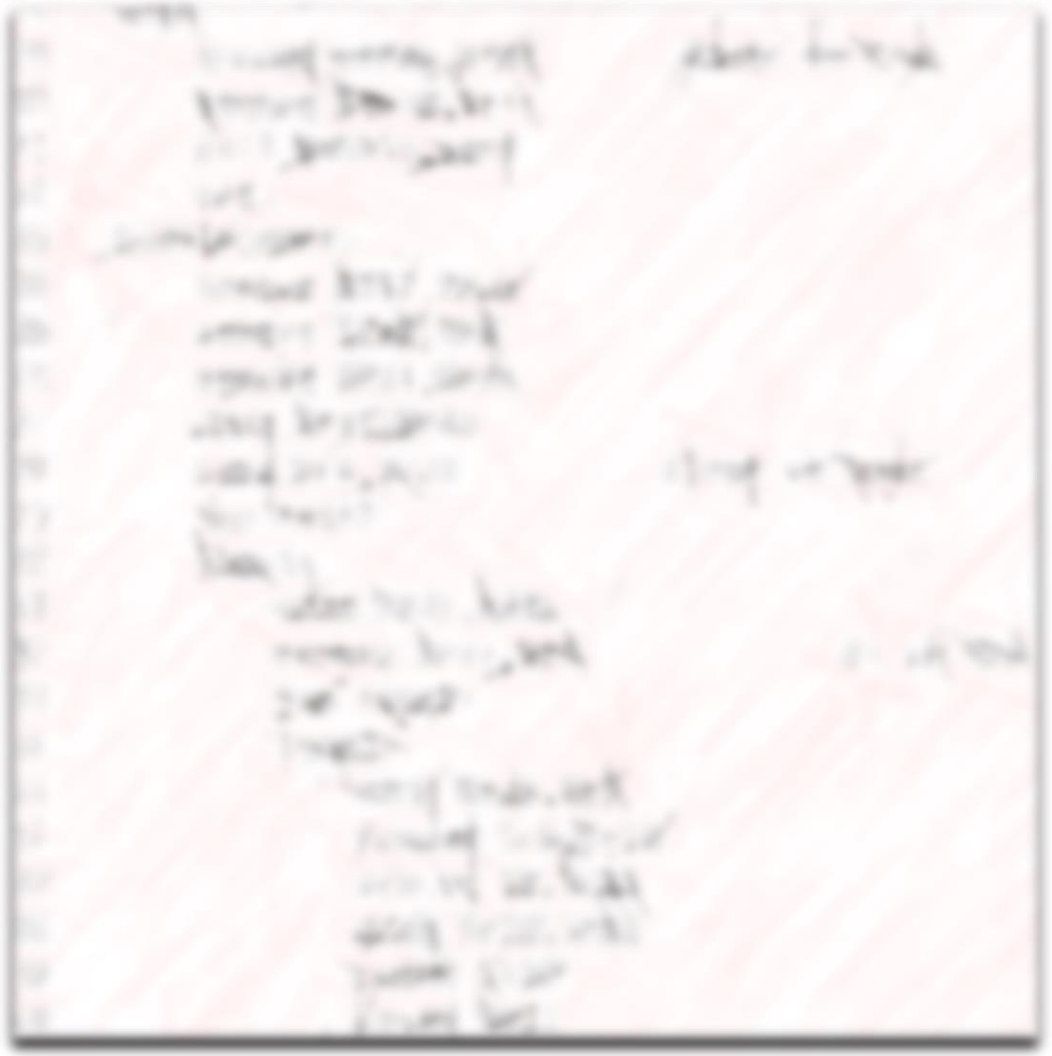
```c
/* bubble_sort - Sort long numbers at data in ascending order */
void bubble_sort(long *data, long count)
{
    long *i, *last;
    for(last = data + count - 1; last > data; last--) {
        for(i = data; i < last; i++) {
            if(*(i + 1) < *i) {
                long t = *(i + 1);
                *(i + 1) = *i;
                *i = t;
            }
        }
    }
}
/* $end examples */
```

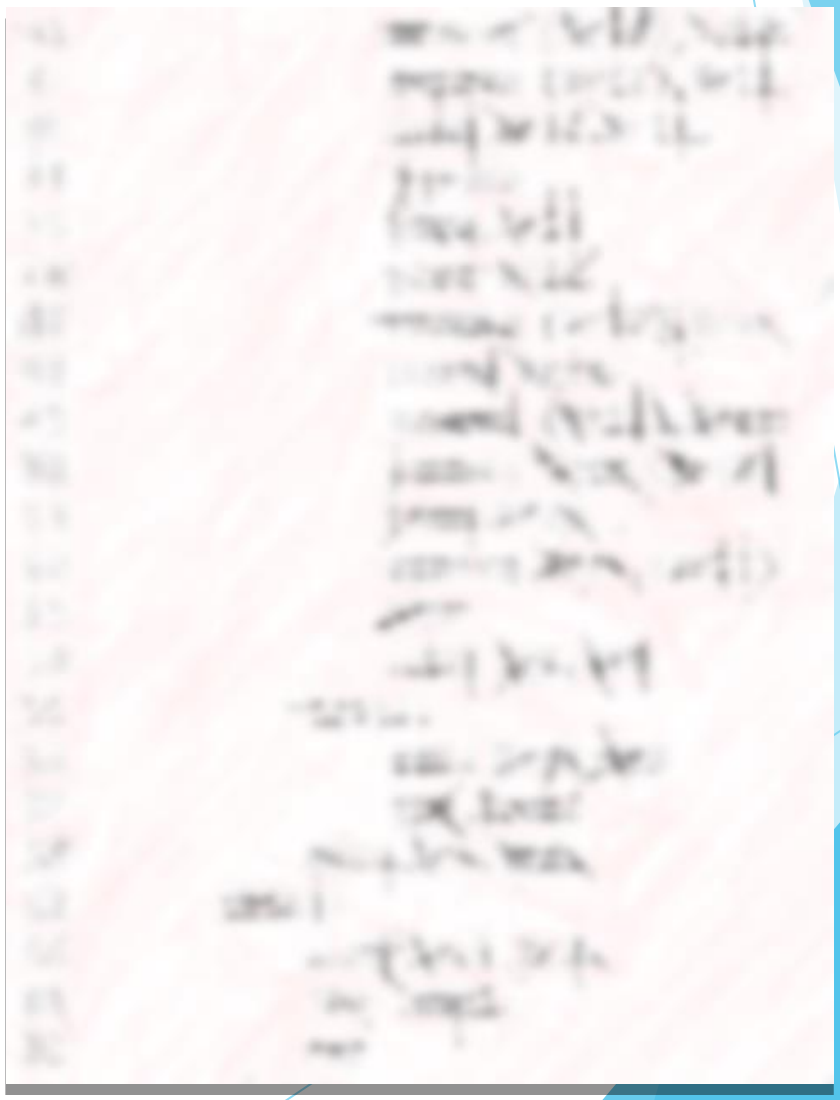
```
u2200013187@icsdancer:~/archlab-handout/sim/misc$ ./yis ans-bubble.yo
Stopped in 331 steps at PC = 0x2a.  Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%rcx:   0x0000000000000000      0x0000000000000acb
%rsp:   0x0000000000000000      0x00000000000001a8
%rsi:   0x0000000000000000      0x0000000000000030
%rdi:   0x0000000000000000      0x0000000000000018
%r8:    0x0000000000000000      0x0000000000000008
%r10:   0x0000000000000000      0x0bac000000000abc
%r11:   0x0000000000000000      0x000000000000000f

Changes to memory:
0x0018: 0x0000000000000bca      0x0000000000000abc
0x0020: 0x0000000000000cba      0x0000000000000acb
0x0028: 0x0000000000000acb      0x0000000000000bac
0x0030: 0x0000000000000cab      0x0000000000000bca
0x0038: 0x0000000000000abc      0x0000000000000cab
0x0040: 0x0000000000000bac      0x0000000000000cba
0x01a0: 0x0000000000000000      0x0000000000000020
0x01a8: 0x0000000000000000      0x0000000000000018
0x01b0: 0x0000000000000000      0x0000000000000028
```

- 含有两个循环

- 只需要按照C代码逐句翻译成汇编语言，过程中注意每个寄存器里存的是什么变量
- Y86中没有mmmovq这种指令，所以需要先将内存中的值取到寄存器中，再放入内存中
- addq和subq也必须是两个寄存器相减，所以需要将8和0x30放入寄存器中使用





PartB:

- sim/seq
- 任务要求：在seq-full.hcl文件中添加iaddq和jm指令
- laddq指令可以参考irmovq和opq的实现
- Jm指令可以参考popq的实现

- 掌握seq各个阶段执行的具体操作

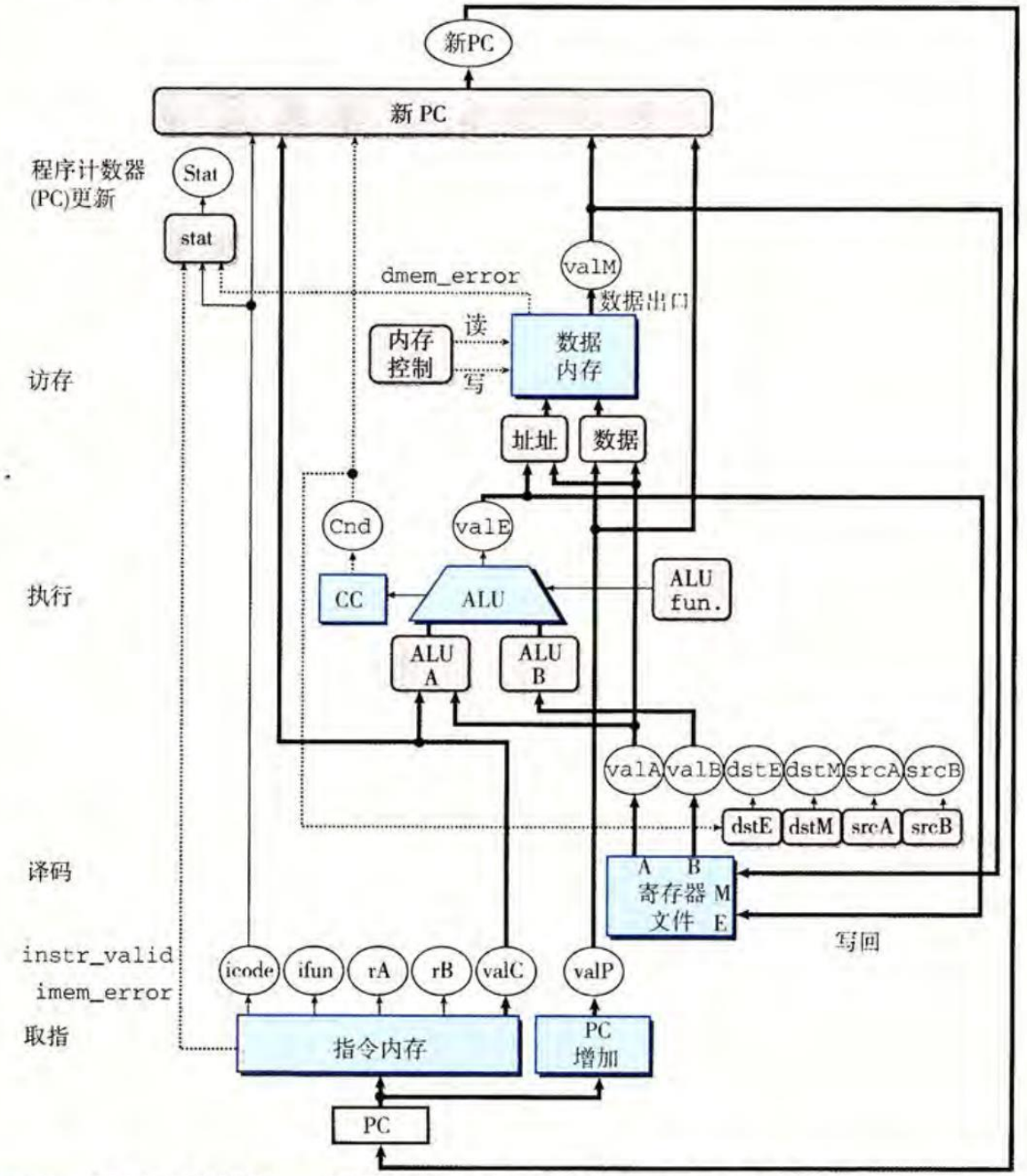
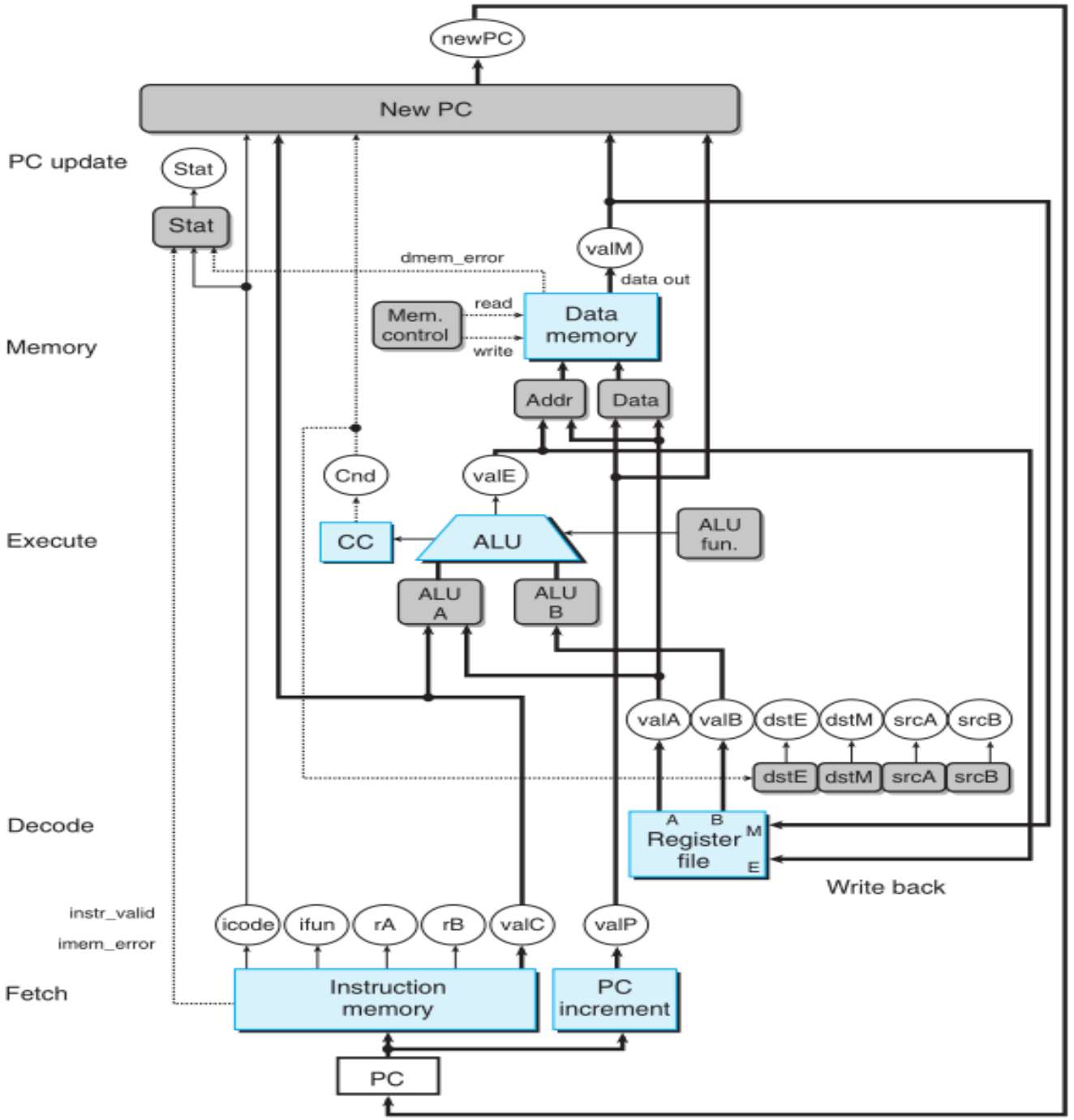


图 4-23 SEQ 的硬件结构，一种顺序实现。有些控制信号以及寄存器和控制字连接没有画出来

阶段	计算	OPq rA, rB
取指	icode:ifun rA, rB valC valP	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2
译码	valA, srcA valB, srcB	valA ← R[rA] valB ← R[rB]
执行	valE Cond. codes	valE ← valB OP valA Set CC
访存	Read/write	
写回	E port, dstE M port, dstM	R[rB] ← valE
更新 PC	PC	PC ← valP



irmovqV, rB
icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC+10
valE ← 0+valC
R[rB] ← valE
PC ← valP

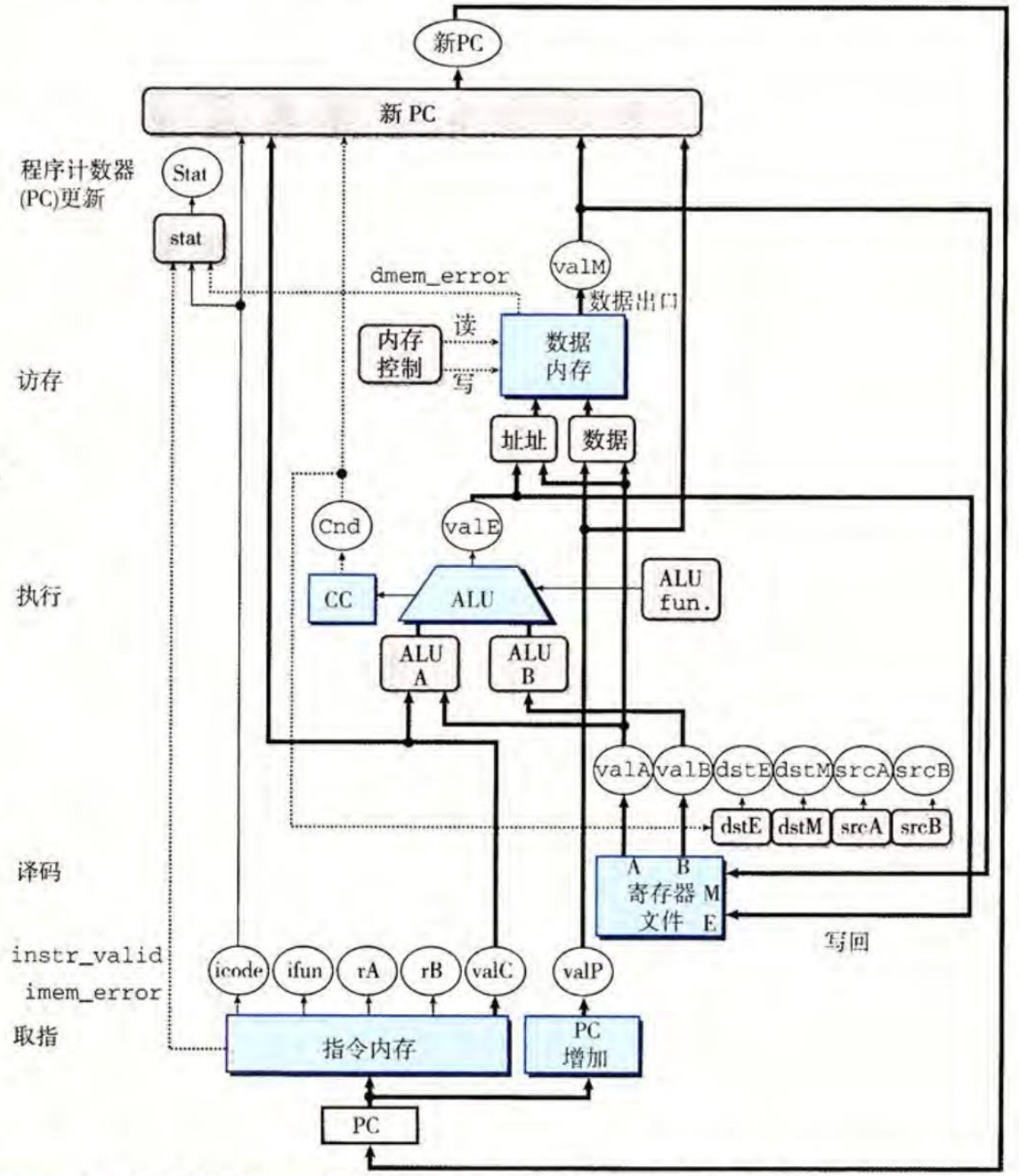


图 4-23 SEQ 的硬件结构，一种顺序实现。有些控制信号以及寄存器和控制字连接没有画出来

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

iaddq

依次看看seq-full.hcl需要更改的地方：
fetch stage：
bool instr_valid：判断指令是否有效，添上IIANDQ
bool need_regids：判断是否需要寄存器指示字节，这个指令需要用到寄存器，因此需要寄存器指示字节指定是哪个寄存器，添上
bool need_valC：判断是否需要常数，添上
decode stage：
word srcA：是否需要源寄存器rA，不需要
word srcB：是否需要源寄存器，我们需要一个源寄存器rB，这个rB提供一个操作数，添上
word dstE：是否需要一个目标寄存器，我们需要一个目的寄存器存计算后的值，还是选rB存，添上
word dstM：是否需要存入内存，不需要
execute stage：首先要知道，ALU执行阶段是aluB OP aluA，由于我们设置的dstE是rB，因此结果存入aluB。aluA与aluB操作数的来源是valA，valB，valC，其中valC存常数。因此aluA中valC添加，aluB中valB添加
bool set_cc：设置标志位，需要设置，添上
memory stage：不需要访存更改PC：不涉及转移指令，不需要修改

jm

- 与添加iaddq指令原理相同
- Fetch stage:

  instr_valid

  need_registers

  need_valC
- Decode stage:

  srcB: rB

  aluA: valC

  aluB: valB
- Memory stage:

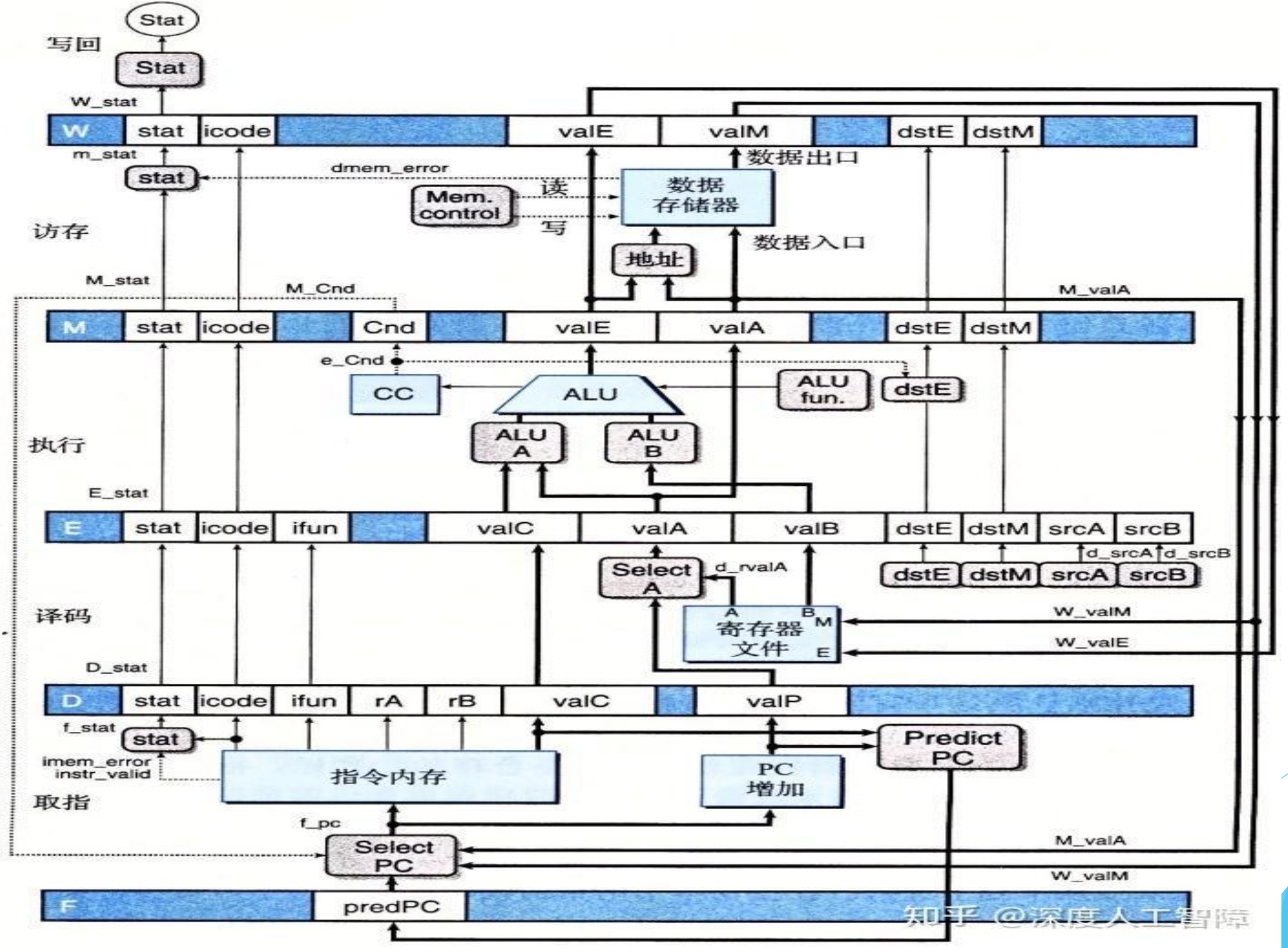
  mem_read: true

  mem_addr: valE
- PC update:

  new_pc: 添加　　icode == IJM : valM;

PartC：

- sim/pipe
- 任务要求：更改hcl文件和ncopy.ys文件加速程序运行（减小CPE值）
- 涉及pipeline和第五章优化内容


Stat
写回
Stat
W_stat
W
stat
icode
valE
valM
dstE
dstM
m_stat
stat
dmem_error
数据出口
Mem.
control
读
写
数据
存储器
访存
地址
数据入口
M_stat
M_Cnd
M_valA
M
stat
icode
Cnd
valE
valA
dstE
dstM
e_Cnd
CC
ALU
ALU
fun.
dstE
执行
ALU
A
ALU
B
E_stat
E
stat
icode
ifun
valC
valA
valB
dstE
dstM
srcA
srcB
d_srcA
d_srcB
Select
A
d_rvalA
dstE
dstM
srcA
srcB
译码
A
B
M
寄存器
文件
E
W_valM
W_valE
D_stat
D
stat
icode
ifun
rA
rB
valC
valP
f_stat
stat
Predict
PC
imem_error
instr_valid
指令内存
PC
增加
取指
f_pc
Select
PC
M_valA
W_valM
F
predPC

- 整体优化思路：
- 必须添加iaddq指令（重要）
- 循环展开（重要）
- mrmovq和rmmovq之间bubble的处理
- 注：通过correctness.pl检查正确性，benchmarkpl检查CPE

函数	方法	整数		浮点数	
		+	*	+	*
combine4	无展开	1.27	3.01	3.01	5.01
combine5	2×1 展开	1.01	3.01	3.01	5.01
	3×1 展开	1.01	3.01	3.01	5.01
延迟界限		1.00	3.00	3.00	5.00
吞吐量界限		0.50	1.00	1.00	0.50

通过修改环境变量方便查看各个case的CPE：
export LC_ALL=C
export LANGUAGE=C

通过看哪个部分CPE较大对代码进行进一步优化

	ncopy	
0	22	
1	20	20.00
2	30	15.00
3	32	10.67
4	51	12.75
5	58	11.60
6	68	11.33
7	62	8.86
8	68	8.50
9	74	8.22
10	85	8.50
11	83	7.55
12	93	7.75
13	95	7.31
14	114	8.14
15	121	8.07
16	131	8.19
17	125	7.35
18	131	7.28
19	137	7.21
20	144	7.20
21	142	6.76
22	152	6.91
23	154	6.70
24	173	7.21
25	180	7.20
26	190	7.31
27	184	6.81
28	190	6.79
29	196	6.76
30	203	6.77
31	201	6.48
32	211	6.59
33	213	6.45
34	232	6.82
35	239	6.83
36	249	6.92
37	243	6.57
38	249	6.55
39	255	6.54
40	262	6.55
41	260	6.34
42	270	6.43
43	272	6.33
44	291	6.61
45	298	6.62
46	308	6.70
47	302	6.43
48	308	6.42
49	314	6.41
50	321	6.42
51	319	6.25
52	329	6.33
53	331	6.25
54	350	6.48
55	357	6.49
56	367	6.55
57	361	6.33
58	367	6.33
59	373	6.32
60	380	6.33
61	378	6.20
62	388	6.26
63	390	6.19
64	409	6.39
Average	CPE	7.51
Score	59.9/60.0	

- 进一步优化思路：
- 加载转发：在不可避免的邻接的mrmovq和rmmovq之间加入转发
- 余项处理：希望对数组中元素个数极少的情况进行加速