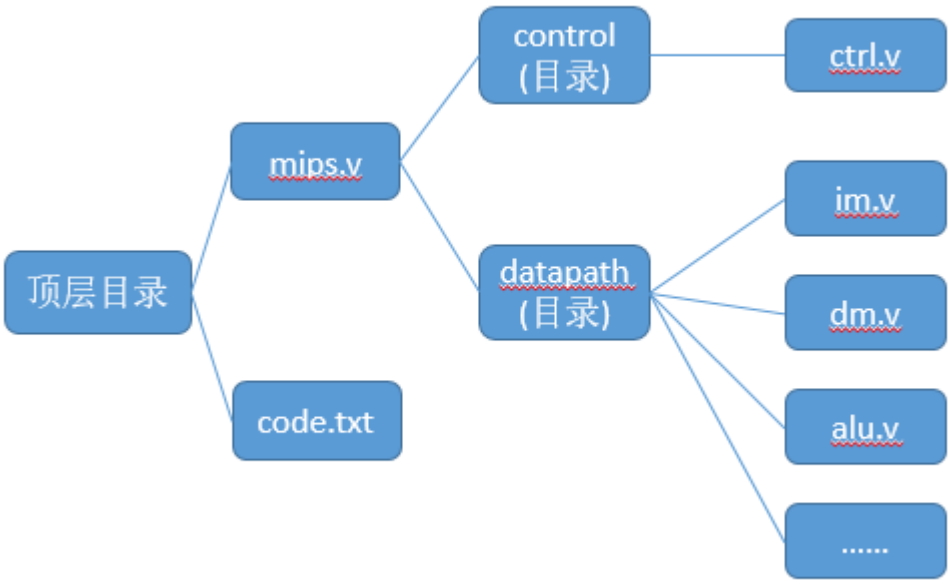
Project4 Verilog 完成单周期处理器开发

一、顶层设计

1、目录结构：



2、支持指令集：

{addu, subu, ori, lw, sw, beq, lui, jal, jr,nop}

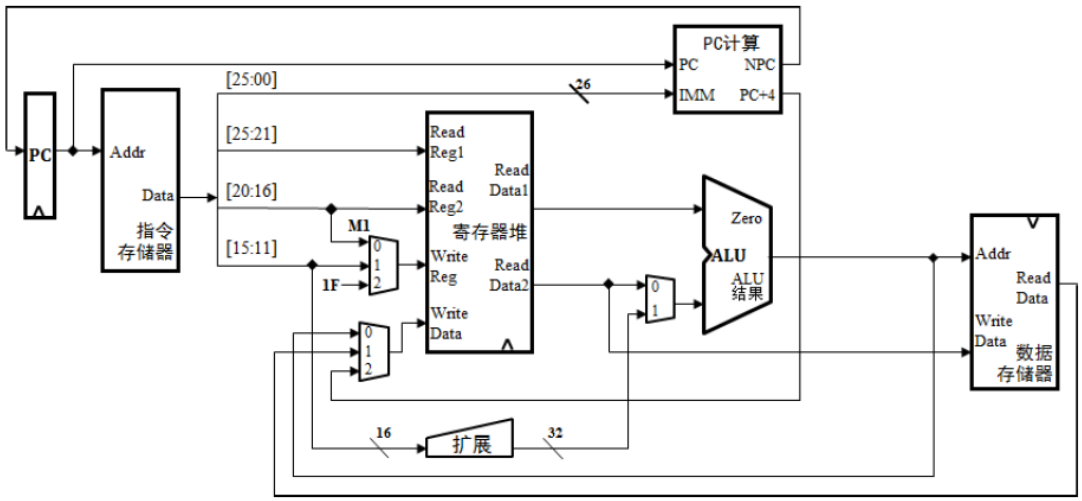
3、顶层文件接口定义：

文件	模块接口定义

mips.v	module mips(clk,reset);
	input clk; //clock
	input reset; //reset

二、数据通路设计

1、数据通路架构图：



2、PC

（1）基本描述：

PC 用寄存器实现，具有复位功能。

起始地址：0x0000_3000。

（2）文件接口定义：

文件	模块接口定义
PC.v	module PC(Clk,Reset,NPC,PC); input Clk, //clock input Reset, //reset input [31:0] NPC, //next PC input output reg[31:0] PC //new PC

（3）功能定义：

序号	功能名称	功能描述
1	复位	当 Reset 有效且时钟上升沿来临时，PC 被设置为 0x0000_3000
2	更新 PC 的值	在时钟上升沿来临时，将 NPC 写入到 PC 中

3、IM

（1）基本描述：

IM 容量为 4KB（32bit×1024 字）。

生成相应数量的 32 位寄存器的阵列。

采用$readmemh 指令将指令读入到 IM 中去。

（2）文件接口定义：

文件	模块接口定义
IM.v	module IM(PC,Instr); input [11:2] PC, //Instruction address output [31:0] Instr //Instruction

（3）功能定义：

序号	功能名称	功能描述
1	取指令	根据 PC 从 IM 中取出指令

4、GRF

（1）基本描述：

用具有写使能的寄存器实现，寄存器总数为 32 。

0 号寄存器的值始终保持为 0。其他寄存器初始值均为 0。

（2）文件接口定义：

文件	模块接口定义
IM.v	module GRF(Clk,Reset,WE,A1,A2,A3,WD,WPC,RD1,RD2); input Clk, //clock input Reset, //reset input WE, //write enable input [4:0] A1, //read reg1 input [4:0] A2, //read reg2 input [4:0] A3, //write reg input [31:0] WD, //write data input [31:0] WPC, // PC output [31:0] RD1, //read data1 output [31:0] RD2 //read data2

（3）功能定义：

序号	功能名称	功能描述
1	复位	Reset 信号有效时，所有寄存器存储的数值清零
2	读数据	读出 A1，A2 地址对应寄存器中所存储的数据到 RD1，RD2
3	写数据	当 WE 有效且时钟上升沿来临时，将 WD 写入 A3 所对应的寄存器中

5、EXT

（1）基本描述：

扩展

（2）文件接口定义：

文件	模块接口定义
EXT.v	module EXT(Imm_16,ExtOp,Imm_32); input [15:0] Imm_16, //input 16 bit Immediate input [1:0] ExtOp, //control the extension output [31:0] Imm_32 //output 32 bit Immediate

（3）功能定义：

序号	功能名称	功能描述
1	符号扩展	将 16 位输入数据符号扩展为 32 位
2	零扩展	将 16 位输入数据符号置为低 16 位，高 16 位置 0
3	加载至高位	将 16 位输入数据符号置为高 16 位，低 16 位置 0
4	符号扩展之后，左移两位	将 16 位输入数据符号扩展为 32 位之后，左移两位

6、NPC

（1）基本描述：

计算下一个 PC 的值

（2）文件接口定义：

文件	模块接口定义
nPC.v	module nPC(Instr,pc,rs,Zero,nPCOp,npc,pc_4); input [25:0] Instr, // Instruction

	input [31:0] pc, // PC input [31:0] rs, // $rs input Zero, // ALU Zero input [2:0] nPCOp, // control the operation output [31:0] npc, // next PC output [31:0] pc_4 // PC + 4

（3）功能定义：

序号	功能名称	功能描述
1	计算下一 PC	计算下一 PC 的值

7、ALU

（1）基本描述：

提供 32 位加、减、或运算等功能。

可以不支持溢出（不检测溢出）。

（2）文件接口定义：

文件	模块接口定义
ALU.v	module ALU(A,B,ALUOp,Zero.Result); input [31:0] A, // input data A input [31:0] B, // input data B input [3:0] ALUOp, // control the ALU output Zero, // Result == 0 output [31:0] Result // result

（3）功能定义：

序号	功能名称	功能描述
1	加运算	C = A + B
2	减运算	C = A - B
3	与运算	C = A & B

8、DM

（1）基本描述：

DM 容量为 4KB（32bit×1024 字）。

起始地址：0x00000000

（2）文件接口定义：

文件	模块接口定义
DM.v	module DM(Clk,Reset,WE,A,WD,pc,addr,RD); input Clk, //clock input Reset, //reset input WE, //memory write enable input [11:2] A, // address input [31:0] WD, //write data input [31:0] pc, // PC input [31:0] addr, // 32-bit addr output[31:0] RD //read data

（3）功能定义：

序号	功能名称	功能描述
1	读数据	读出地址 A 中所存储的数据到 RD
2	写数据	当 WE 有效且时钟上升沿来临时，将 WD 写入地址 A

三、控制器设计

1、基本描述：

控制单元基于指令的 opcode 字段（$Instr_{31:26}$）和 funct 字段（$Instr_{5:0}$）计算控制信号。

2、模块定义：

信号名	方向	描述
Op [5:0]	I	用于识别指令的功能
Func [5:0]	I	用于辅助 op 来识别指令
RegDst [1:0]	O	控制写入端地址选择 00：寄存器堆写入端地址选择 Rt 字段 01：寄存器堆写入端地址选择 Rd 字段 10：寄存器堆写入端地址选择 31 号寄存器
RegWrite	O	GRF 的写使能信号 0：无效 1：把数据写入寄存器堆中对应寄存器
ALUSrc	O	控制 ALU 的操作 0：ALU 输入端 B 选择寄存器堆输出 R[rt] 1：ALU 输入端 B 选择 extend(immediate)
MemWrite	O	DM 的写使能信号 0：无效 1：数据存储器 DM 写数据（输入）
MemtoReg [1:0]	O	控制数据从 ALU 读出还是从 DM 读出 00：寄存器堆写入端数据来自 ALU 输出 01：寄存器堆写入端数据来自 DM 输出 10：寄存器堆写入端数据来自 PC+4
ExtOp [1:0]	O	EXT 功能的选择信号 00：符号扩展 01：零扩展 10：加载至高位 11：符号扩展之后，左移两位
nPCOp [2:0]	O	下一 PC 的选择信号 000：PC = PC+4 001：执行 beq 分支指令 010：执行 jal 跳转指令 011：执行 jr 跳转指令
ALUControl [3:0]	O	ALU 功能的选择信号 0000：加法运算 0001：减法运算 0010：或运算

3、文件接口定义：

文件	模块接口定义
ctrl.v	module ctrl(Op,Func,RegDst,RegWrite,ALUSrc,MemWrite,MemReg, ExtOp,ALUOp,nPCOp); input [5:0] Op, input [5:0] Func, output [1:0] RegDst, output RegWrite, output ALUSrc, output MemWrite, output [1:0]MemtoReg, output [1:0] ExtOp, output [3:0] ALUOp, output [2:0] nPCOp

4 支持指令集：

(1)addu 指令：

① 功能：无符号加法，不考虑溢出

② 操作：GPR[rd] ← GPR[rs] + GPR[rt]

③ 编码： 000000[31:26] rs[25:11] rt[20:16] rd[15:11] 00000[10:6] 100001[5:0]

④ 控制信号：

RegDst	RegWrite	ALUSrc	MemWrite	MemtoReg	ExtOp	ALUControl	nPCOp
01	1	0	0	00	xx	0000	000

(2)subu 指令：

① 功能：无符号减法，不考虑溢出

② 操作：GPR[rd] ← GPR[rs] - GPR[rt]

③ 编码： 000000[31:26] rs[25:11] rt[20:16] rd[15:11] 00000[10:6]

100010[5:0]

④ 控制信号：

RegDst	RegWrite	ALUSrc	MemWrite	MemtoReg	ExtOp	ALUControl	nPCOp
01	1	0	0	00	xx	0001	000

(3)ori 指令：

① 功能：或立即数

② 操作：GPR[rt] ← GPR[rs] OR zero_extend(immediate)

③ 编码：001101[31:26] rs[25:11] rt[20:16] immediate[15:0]

④ 控制信号：

RegDst	RegWrite	ALUSrc	MemWrite	MemtoReg	ExtOp	ALUControl	nPCOp
00	1	1	0	00	01	0010	000

(4)lw 指令：

① 功能：加载字，从内存中读取 4 个字节

② 操作：GPR[rt] ← GPR[base] + sign_extend(immediate)

③ 编码：100011[31:26] base[25:11] rt[20:16] offset[15:0]

④ 控制信号：

RegDst	RegWrite	ALUSrc	MemWrite	MemtoReg	ExtOp	ALUControl	nPCOp
00	1	1	0	01	00	0000	000

(5)sw 指令：

① 功能：存储字，向内存中写入 4 个字节

② 操作：Addr ← GPR[base] + sign_extend(immediate)

Memory[Addr] ← GPR[rt]

③ 编码：101011[31:26] base[25:11] rt[20:16] offset[15:0]

④ 控制信号：

RegDst	RegWrite	ALUSrc	MemWrite	MemtoReg	ExtOp	ALUControl	nPCOp
xx	0	1	1	xx	00	0000	000

(6)beq 指令：

① 功能：当两个待比较寄存器相等时，跳转到分支地址

② 操作：if (GPR[rs] == GPR[rt])

PC ← PC + 1 + sign_extend(offset || 0^2)

else

PC ← PC + 1

③ 编码：000100[31:26] rs[25:11] rt[20:16] offset[15:0]

④ 控制信号：

RegDst	RegWrite	ALUSrc	MemWrite	MemtoReg	ExtOp	ALUControl	nPCOp
xx	0	0	0	xx	11	0001	001

(7)lui 指令：

① 功能：立即数加载至最高位

② 操作：GPR[rt] ← GPR[zero] + immediate || 0^{16}

③ 编码：001111[31:26] 00000[25:11] rt[20:16] immediate[15:0]

④ 控制信号：

RegDst	RegWrite	ALUSrc	MemWrite	MemtoReg	ExtOp	ALUControl	nPCOp
00	1	1	0	00	10	0000	000

（8）jal 指令：

①功能：跳转到地址，并将 PC+4 保存在 GPR[31]中

②操作：PC←PC31…28||instr_index||0^2

GPR[31]←PC+4

③编码：000011[31:26] insrt_index[25:0]

④控制信号：

RegDst	RegWrite	ALUSrc	MemWrite	MemtoReg	ExtOp	ALUControl	nPCOp
10	1	x	0	10	11	xxxx	010

（9）jr 指令：

①功能：跳转到寄存器

②操作：PC←GPR[rs]

③编码：000000[31:26] rs[25:11] 0[20:11] 00000[10:6] 001000[5:0]

④控制信号：

RegDst	RegWrite	ALUSrc	MemWrite	MemtoReg	ExtOp	ALUControl	nPCOp

xx	0	x	0	xx	00	xxxx	011

（10）nop 指令：

①功能：不执行任何操作

②操作：无

③编码：00000000000000000000000000000000 [31:0]

④控制信号：

RegDst	RegWrite	ALUSrc	MemWrite	MemtoReg	ExtOp	ALUControl	nPCOp
00	0	0	0	00	00	xxxx	000

四、测试

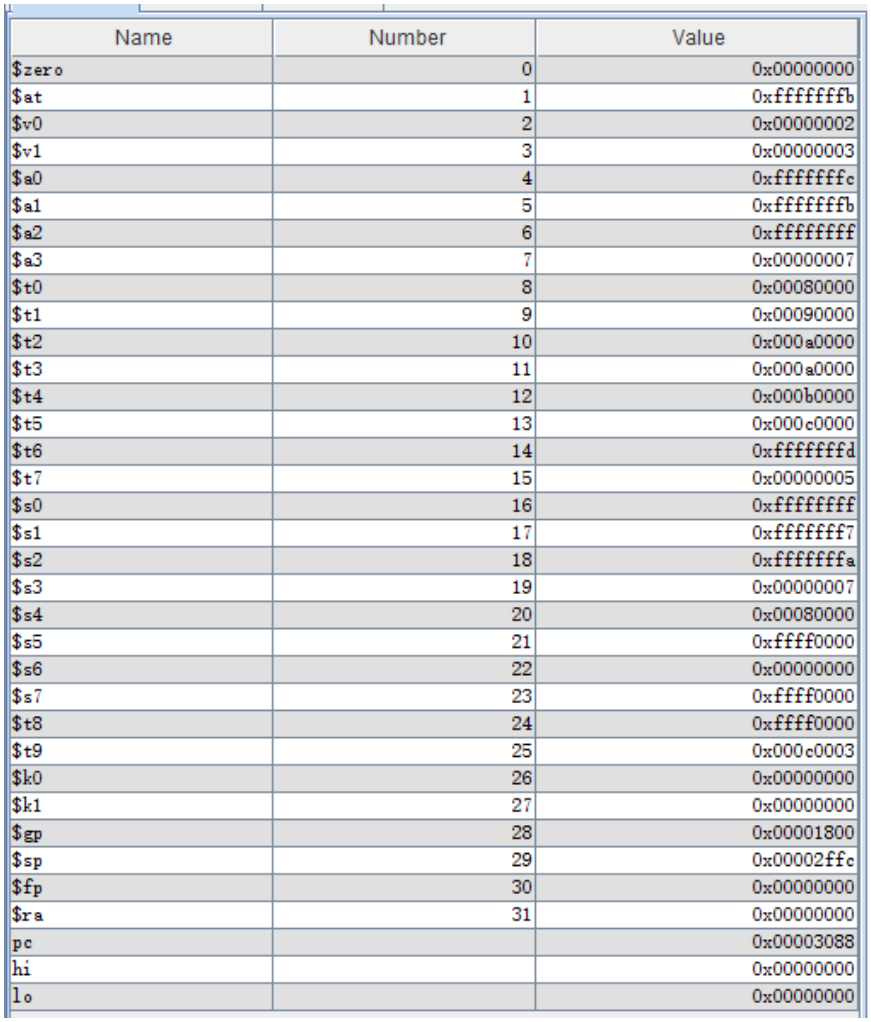
1、首先测试 ori、lui、addu、subu、nop

```
ori $0,$0,100
ori $1,$0,1
ori $2,$0,2
ori $3,$0,3
ori $4,$0,-4
ori $5,$0,-5
ori $6,$1,6
ori $7,$2,7
nop

lui $8,8
lui $9,9
lui $10,10
lui $11,0xa
lui $12,0xb
lui $13,0xc
nop

addu   $14,$1,$2
addu   $15,$2,$3
addu   $16,$3,$4
addu   $17,$4,$5
addu   $18,$5,$6
addu   $19,$19,$7
addu   $20,$20,$8
nop

subu   $21,$9,$10
subu   $22,$10,$11
subu   $23,$11,$12
subu   $24,$12,$13
subu   $25,$13,$14
nop
34000064
34010001
34020002
34030003
3c01ffff
3421fffc
00012025
3c01ffff
```



Name	Number	Value
$zero	0	0x00000000
$at	1	0xfffffffb
$v0	2	0x00000002
$v1	3	0x00000003
$a0	4	0xfffffffc
$a1	5	0xfffffffb
$a2	6	0xffffffff
$a3	7	0x00000007
$t0	8	0x00080000
$t1	9	0x00090000
$t2	10	0x000a0000
$t3	11	0x000a0000
$t4	12	0x000b0000
$t5	13	0x000c0000
$t6	14	0xfffffffd
$t7	15	0x00000005
$s0	16	0xffffffff
$s1	17	0xfffffff7
$s2	18	0xfffffffa
$s3	19	0x00000007
$s4	20	0x00080000
$s5	21	0xffff0000
$s6	22	0x00000000
$s7	23	0xffff0000
$t8	24	0xffff0000
$t9	25	0x000c0003
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00003088
hi		0x00000000
lo		0x00000000

```
3421fffb
00012825
34260006
34470007
00000000
3c080008
3c090009
3c0a000a
3c0b000a
3c0c000b
3c0d000c
00000000
00227021
```

2、然后测试全部指令

```
.data
arr:   .space 40
.text
ori    $10,$0,4
ori    $11,$0,0
ori $12,$0,3
jal sssww

lui $13,100
jal llww

ori $20,$0,1
ori $21,$0,2
ori $22,$0,2

beq $20,$21,end
ori $24,$0,111
beq $21,$22,end
ori $25,$0,111

sssww:  sw  $11,arr($11)
    addu   $11,$11,$10
    sw  $11,arr($11)
    addu   $11,$11,$10
    sw  $11,arr($11)
    addu   $11,$11,$10
    jr  $31
    nop
```

```
llww:  subu   $11,$11,$10
    lw  $14,arr($11)
    subu   $11,$11,$10
    lw  $15,arr($11)
    subu   $11,$11,$10
    lw  $16,arr($11)
    jr  $31
    nop

end:   ori    $30,111
340a0004
340b0000
340c0003
0c000c0d
3c0d0064
0c000c15
34140001
34150002
34160002
12950013
3418006f
12b60011
3419006f
ad6b0000
016a5821
ad6b0000
016a5821
ad6b0000
016a5821
03e00008
00000000
016a5823
8d6e0000
016a5823
8d6f0000
016a5823
8d700000
03e00008
00000000
37de006f
```

0	0x00000000
1	0x00000000
2	0x00000000
3	0x00000000
4	0x00000000
5	0x00000000
6	0x00000000
7	0x00000000
8	0x00000000
9	0x00000000
10	0x00000004
11	0x00000000
12	0x00000003
13	0x00640000
14	0x00000008
15	0x00000004
16	0x00000000
17	0x00000000
18	0x00000000
19	0x00000000
20	0x00000001
21	0x00000002
22	0x00000002
23	0x00000000
24	0x0000006f
25	0x00000000
26	0x00000000
27	0x00000000
28	0x00001800
29	0x00002ffc
30	0x0000006f
31	0x00003018

五、思考题

1、根据你的理解，在下面给出的 DM 的输入示例中，地址信号 addr 位数为什么是[11:2]而不是[9:0]？这个 addr 信号又是从哪里来的？

文件	模块接口定义

dm.v	dm(clk,reset,MemWrite,addr,din,dout);
	input clk; //clock
	input reset; //reset
	input MemWrite; //memory write enable
	input [11:2] addr; //memory's address for write
	input [31:0] din; //write data
	output [31:0] dout; //read data

DM 的存储是按字存储，而传入的地址是以字节为单位，相差 4 倍，所以取 2—11 位信号作为地址，addr 来自 ALU

2、在相应的部件中，reset 的优先级比其他控制信号（不包括 clk 信号）都要高，且相应的设计都是同步复位。清零信号 reset 是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

PC：恢复初始地址 0x0000_3000，重新开始执行程序
GRF：清空寄存器，避免上一次运行程序对本程序的影响
DM：清空存储单元，避免上一次运行程序对本程序的影响

3、列举出用 Verilog 语言设计控制器的几种编码方式（至少三种），并给出代码示例。

1、case：

```
parameter addu_f = 6'b100001;
case(Func)
    addu_f:begin
                RegDst    <= 2'b01;
                RegWrite  <= 1;
                ALUSrc    <= 0;
                MemWrite  <= 0;
                MemtoReg  <= 2'b00;
                ExtOp     <= 2'bxx;
                ALUOp     <= 4'b0000;
                nPCOp     <= 3'b000;
           end
```

2、assign：

```
assign  memwrite = (Op == 6'b101011) ?  1:0;
```

3、宏定义：

```
`define sw 6'b101011
memwrite =sw
```

4、根据你所列举的编码方式，说明他们的优缺点。

1、case：

优点：方便维护

缺点：不方便扩展控制信号维数

2、assign：

```
assign  memwrite = (Op == 6'b101011) ?  1:0;
```

优点：便于添加和修改指令

缺点：容易出错

3、宏定义：

```
`define sw 6'b101011
memwrite =sw
```

优点：便于维护

缺点：宏定义较多，设计 Funct 字段更复杂

5、C 语言是一种弱类型程序设计语言。C 语言中不对计算结果溢出进行处理，这意味着 C 语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持 C 语言，MIPS 指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi 与 addiu 是等价的，add 与 addu 是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的 Operation 部分 。

有符号溢出时，符号位会被加法进位（减法借位）影响而出现错误。

不考虑溢出时，因为二进制补码对有无符号的加减法的处理规则是一样的。

6、根据自己的设计说明单周期处理器的优缺点。

优点：

控制简单，没有延迟槽，没有数据冲突；

缺点：

①性能较低：时钟周期对所有指令等长，由计算机中可能的最长路径决定；

②成本较高：一个功能单元在一个时钟周期内只能使用一次，所需的硬件数量多

7、简要说明 jal、jr 和堆栈的关系。

jal：跳转并链接，相当于调用函数，将必要变量保存在栈中

jr：跳转到寄存器，相当于函数返回，将保存在栈中的变量从栈中取出恢复