



南京大學  
NANJING UNIVERSITY



# IA-32中的传送指令

南京大学

计算机科学与技术系

袁春风

email: [cfyuan@nju.edu.cn](mailto:cfyuan@nju.edu.cn)

2015.6

# IA-32常用指令类型

---

## (1) 传送指令

### — 通用数据传送指令

MOV: 一般传送, 包括mov**b**、mov**w**和mov**l**等

MOVS: 符号扩展传送, 如mov**sbw**、mov**swl**等

MOVZ: 零扩展传送, 如mov**zwl**、mov**zbl**等

XCHG: 数据交换

PUSH/POP: **入栈/出栈**, 如push**l**, push**w**, pop**l**, pop**w**等

### — 地址传送指令

LEA: 加载有效地址, 如lea**l** (%edx,%eax), %eax” 的功能为

$R[edx] \leftarrow R[edx] + R[eax]$ , 执行前, 若 $R[edx] = i$ ,

$R[eax] = j$ , 则指令执行后,  $R[eax] = i + j$

### — 输入输出指令

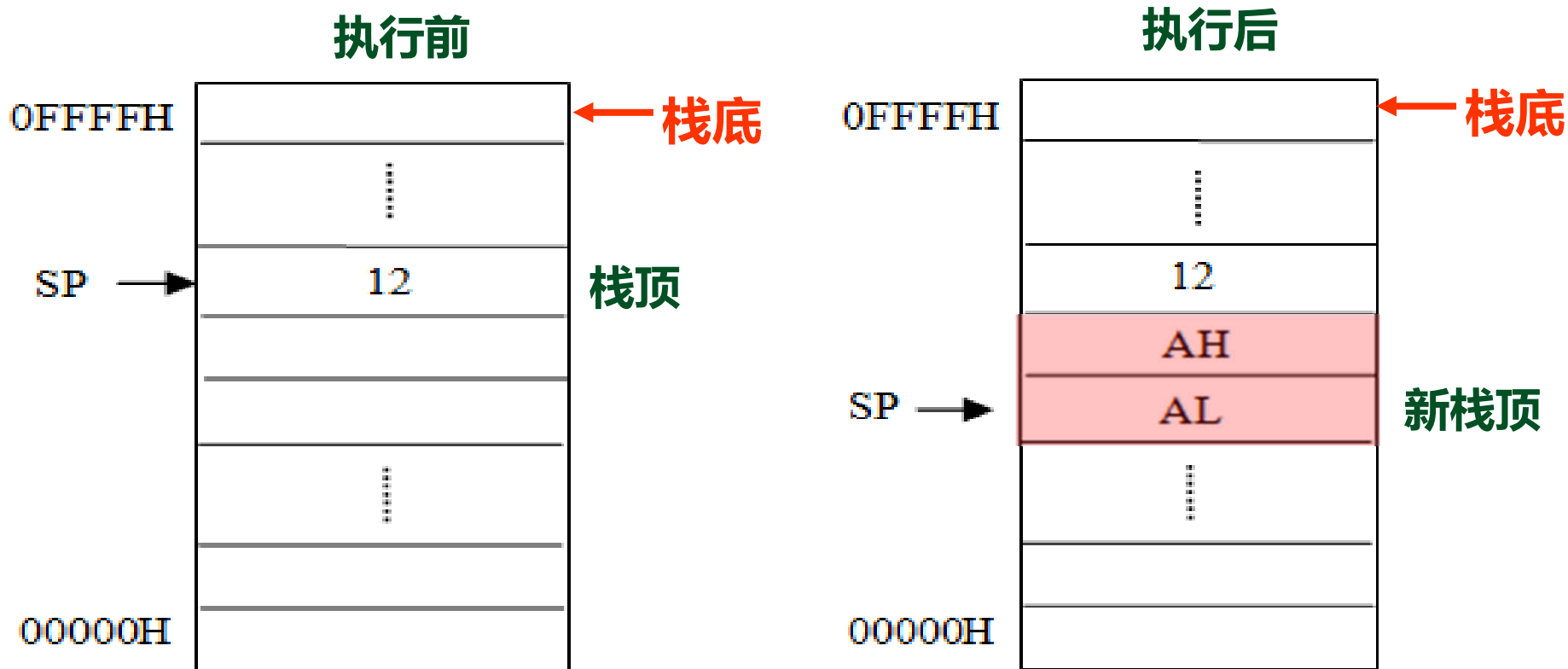
IN和OUT: I/O端口与寄存器之间的交换

### — 标志传送指令

PUSHF、POPF: 将EFLAG压栈, 或将栈顶内容送EFLAG

# “入栈” (pushw %ax)

- 栈 (Stack) 是一种采用 “先进后出” 方式进行访问的一块存储区，用于嵌套过程调用。从高地址向低地址增长
- “栈” 不等于 “堆栈” (由 “堆” 和 “栈” 组成)



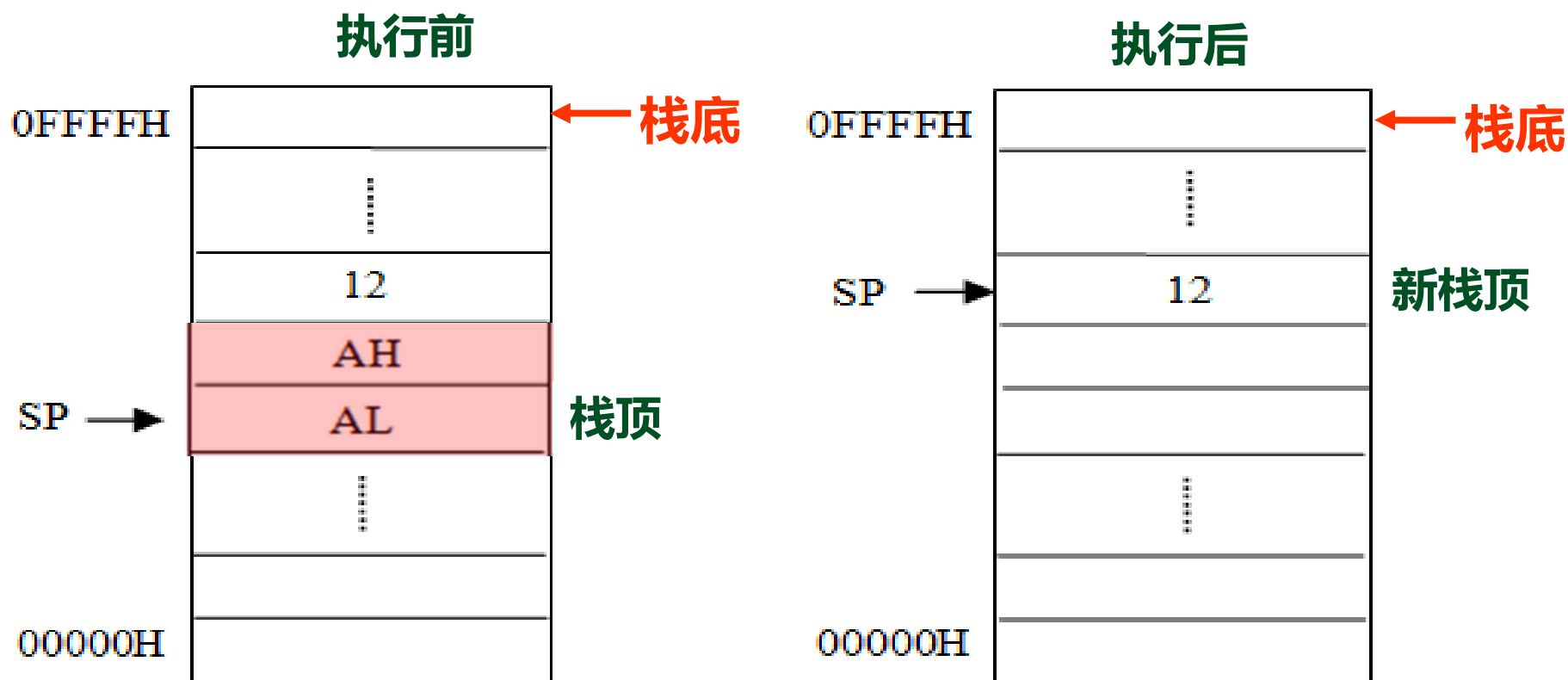
$R[sp] \leftarrow R[sp] - 2$ 、  
 $M[R[sp]] \leftarrow R[ax]$

为什么AL在栈顶？

小端方式！

# “出栈” (popw %ax)

- 栈 (Stack) 是一种采用“先进后出”方式进行访问的一块存储区，用于嵌套过程调用。从高地址向低地址增长



$R[ax] \leftarrow M[R[sp]]$ ,  
 $R[sp] \leftarrow R[sp] + 2$

原栈顶处的数据送AX

# 程序由指令序列组成

```
1 // test.c
2 #include <stdio.h>
3 int add(int i, int j)
4 {
5     int x = i + j;
6     return x;
7 }
```

add函数中有哪些传送指令?

每一条传送指令的功能是什么?

“objdump -d test” 结果

指令的功能用RTL描述

080483d4 <add>:

80483d4:	55	<u>push %ebp</u>	$R[esp] \leftarrow R[esp] - 4; M[R[esp]] \leftarrow R[ebp]$
80483d5:	89 e5	<u>mov %esp, %ebp</u>	$R[ebp] \leftarrow R[esp]$
80483d7:	83 ec 10	sub \$0x10, %esp	
80483da:	8b 45 0c	<u>mov 0xc(%ebp), %eax</u>	$R[eax] \leftarrow M[R[ebp] + 12]$
80483dd:	8b 55 08	<u>mov 0x8(%ebp), %edx</u>	$R[edx] \leftarrow M[R[ebp] + 8]$
80483e0:	8d 04 02	<u>lea (%edx,%eax,1), %eax</u>	$R[eax] \leftarrow R[edx] + R[eax]$
80483e3:	89 45 fc	<u>mov %eax, -0x4(%ebp)</u>	$M[R[ebp] - 4] \leftarrow R[eax]$
80483e6:	8b 45 fc	<u>mov -0x4(%ebp), %eax</u>	$R[eax] \leftarrow M[R[ebp] - 4]$
80483e9:	c9	leave	
80483ea:	c3	ret	

# 程序由指令序列组成

```
1 // test.c
2 #include <stdio.h>
3 int add(int i, int j )
4 {
5     int x = i +j;
6     return x;
7 }
```

程序的执行过程如何?

周而复始执行指令!

指令如何执行?

“objdump -d test” 结果

080483d4 <add>: EIP←0x80483d4

80483d4:	55	push %ebp
80483d5:	89 e5	mov %esp, %ebp
80483d7:	83 ec 10	sub \$0x10, %esp
80483da:	8b 45 0c	mov 0xc(%ebp), %eax
80483dd:	8b 55 08	mov 0x8(%ebp), %edx
80483e0:	8d 04 02	lea (%edx,%eax,1), %eax
80483e3:	89 45 fc	mov %eax, -0x4(%ebp)
80483e6:	8b 45 fc	mov -0x4(%ebp), %eax
80483e9:	c9	leave
80483ea:	c3	ret

取并  
执行  
指令

根据EIP取指令  
指令译码  
取操作数  
指令执行  
回写结果  
修改EIP的值

举例

add函数从80483d4开始!

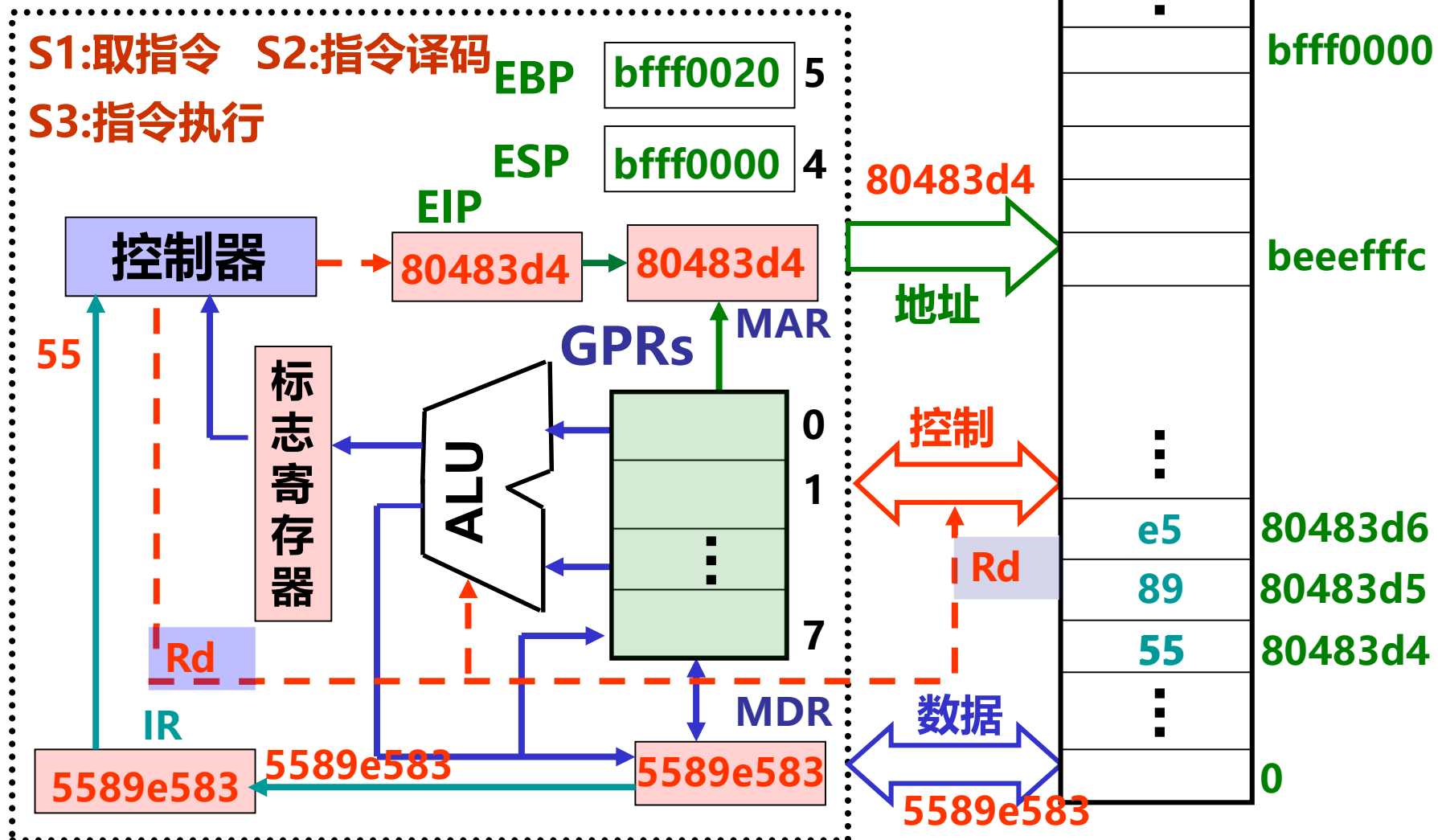
OP

执行add时, 起始EIP=?

功能:  $R[esp] \leftarrow R[esp] - 4$ ,  $M[R[esp]] \leftarrow R[ebp]$

080483d4 <add>:

→ 80483d4: 55      push %ebp  
80483d5: 89 e5    mov %esp, %ebp



功能:  $R[esp] \leftarrow R[esp] - 4$ ,  $M[R[esp]] \leftarrow R[ebp]$

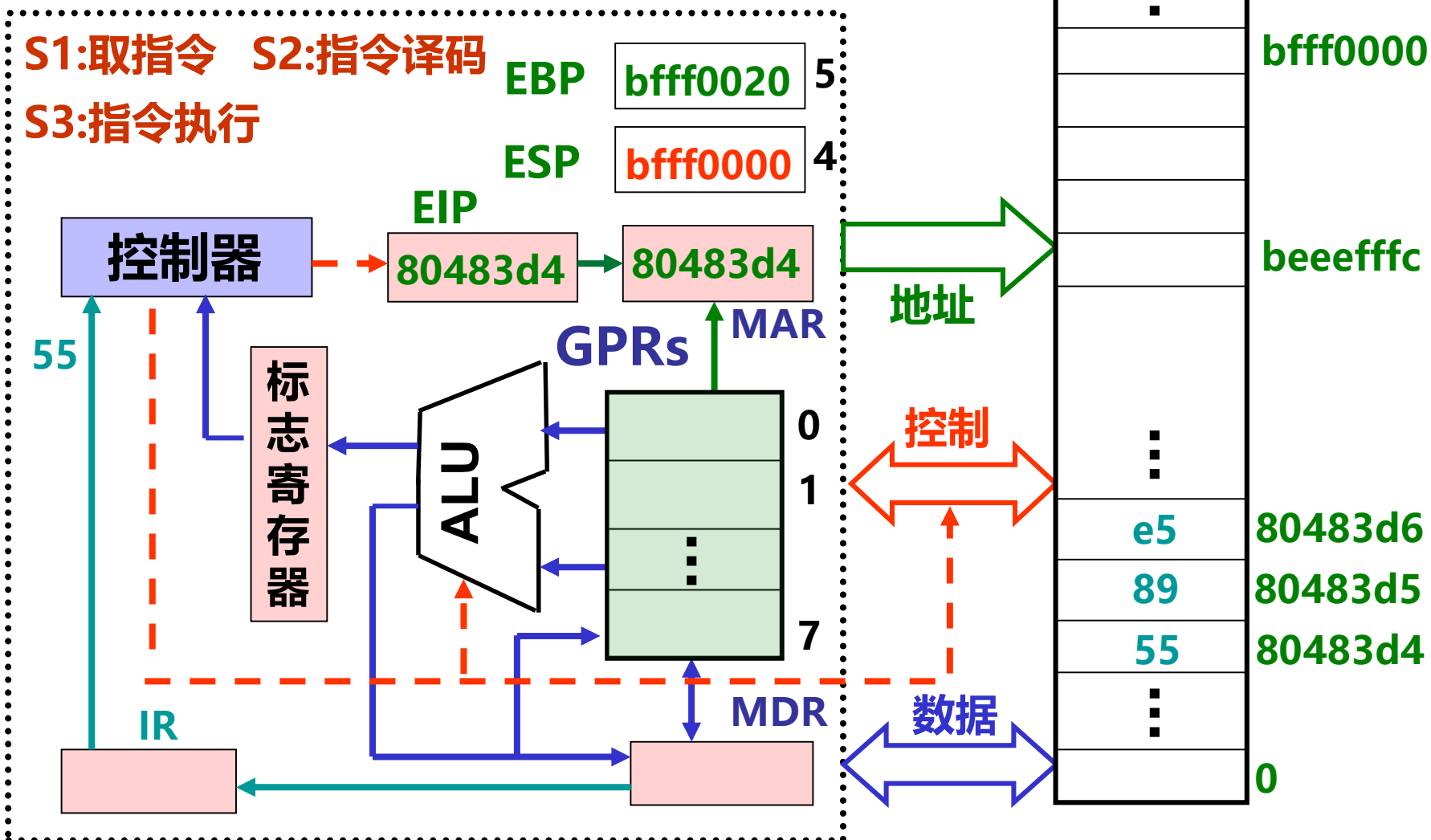
080483d4 <add>:

80483d4: 55 push %ebp

80483d5: 89 e5 mov %esp, %ebp

S1:取指令 S2:指令译码

S3:指令执行



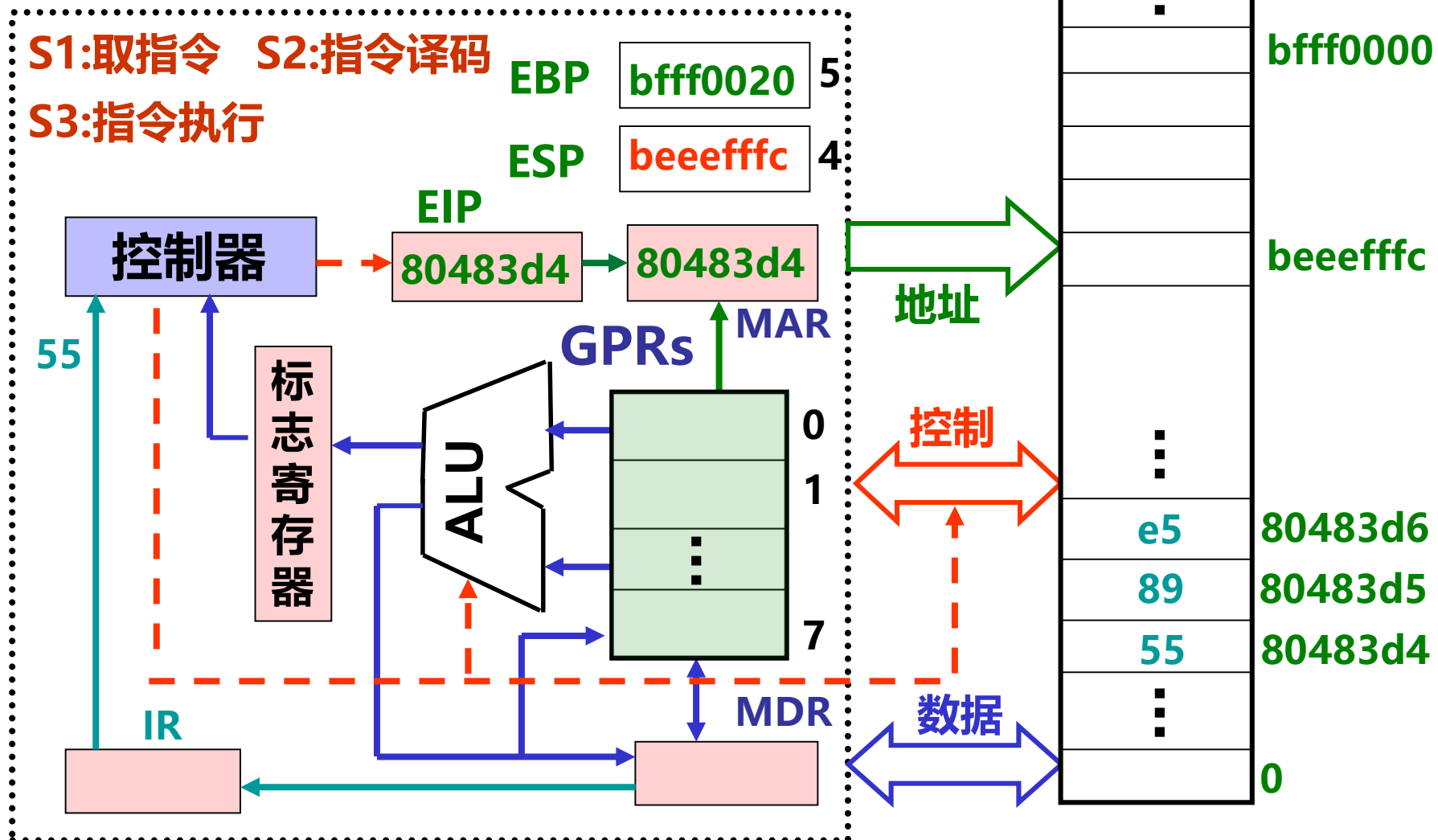


**功能:  $R[esp] \leftarrow R[esp]-4, M[R[esp]] \leftarrow R[ebp]$**

**080483d4 <add>:**

```
80483d4: 55      push %ebp
```

80483d5: 89 e5 mov %esp, %ebp



功能:  $R[esp] \leftarrow R[esp] - 4$ ,  $M[R[esp]] \leftarrow R[ebp]$

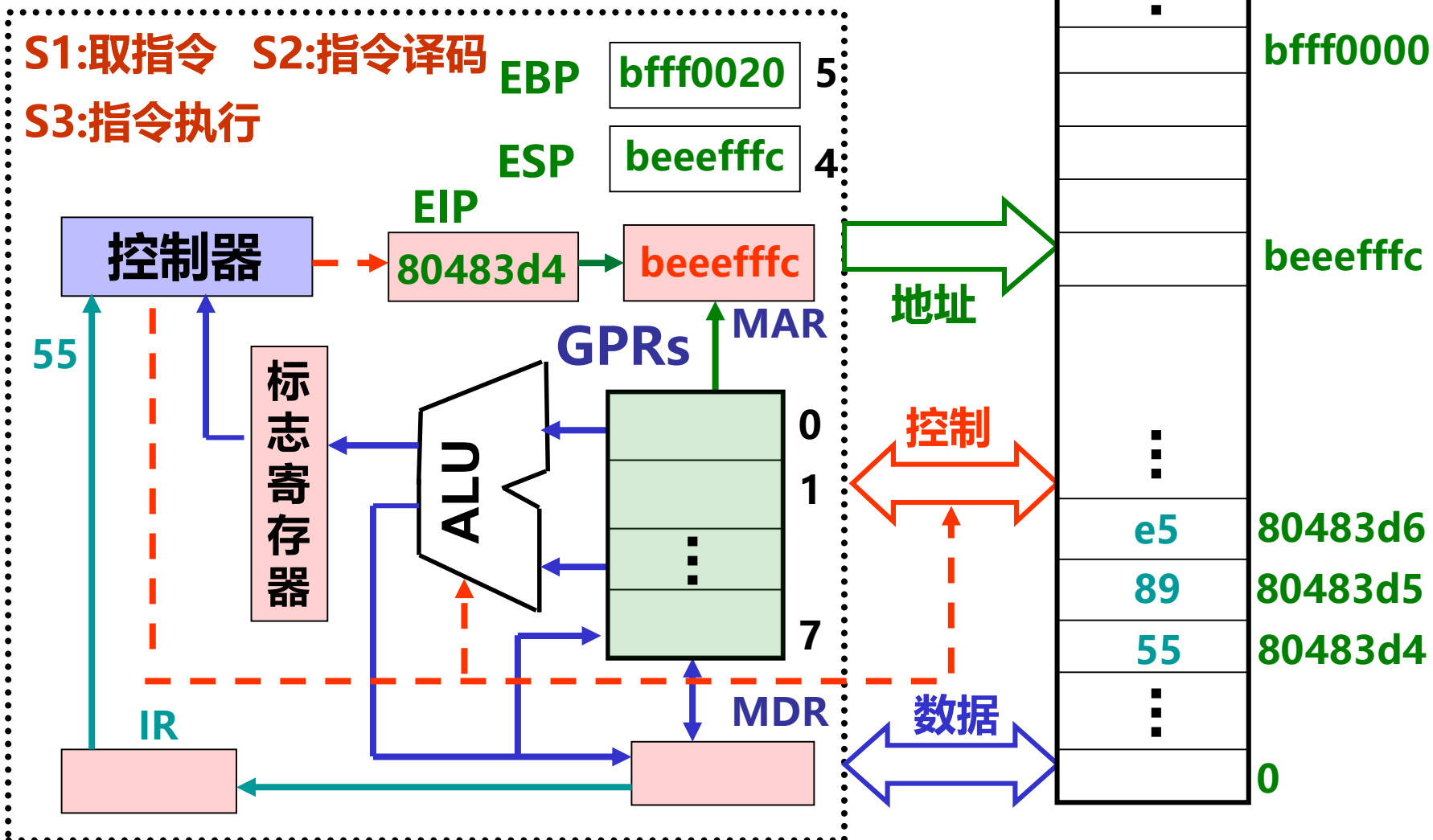
080483d4 <add>:

80483d4: 55 push %ebp

80483d5: 89 e5 mov %esp, %ebp

S1:取指令 S2:指令译码

S3:指令执行



功能:  $R[esp] \leftarrow R[esp] - 4$ ,  $M[R[esp]] \leftarrow R[ebp]$

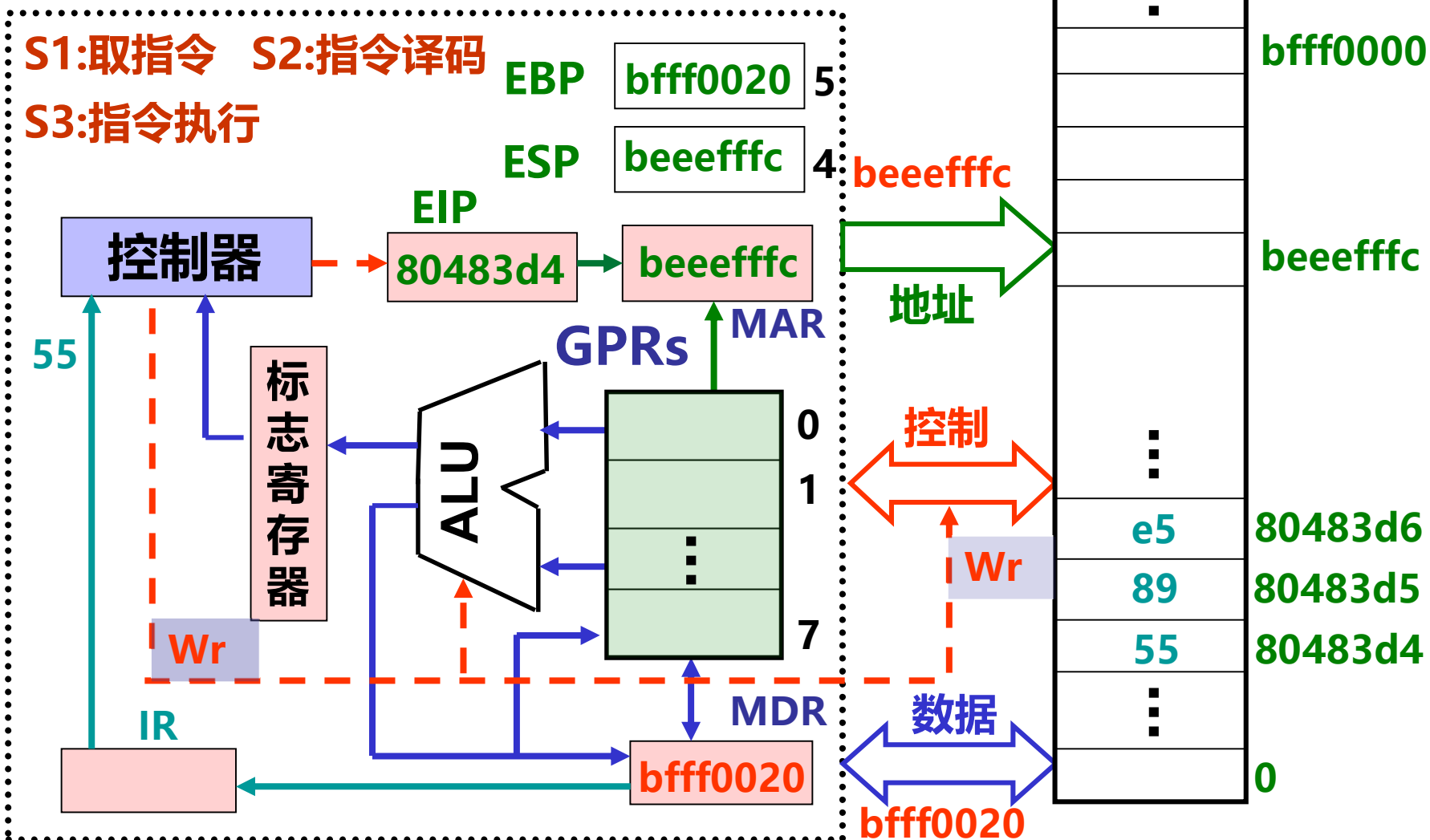
080483d4 <add>:

80483d4: 55 push %ebp

80483d5: 89 e5 mov %esp, %ebp

S1:取指令 S2:指令译码

S3:指令执行



功能:  $R[esp] \leftarrow R[esp] - 4$ ,  $M[R[esp]] \leftarrow R[ebp]$

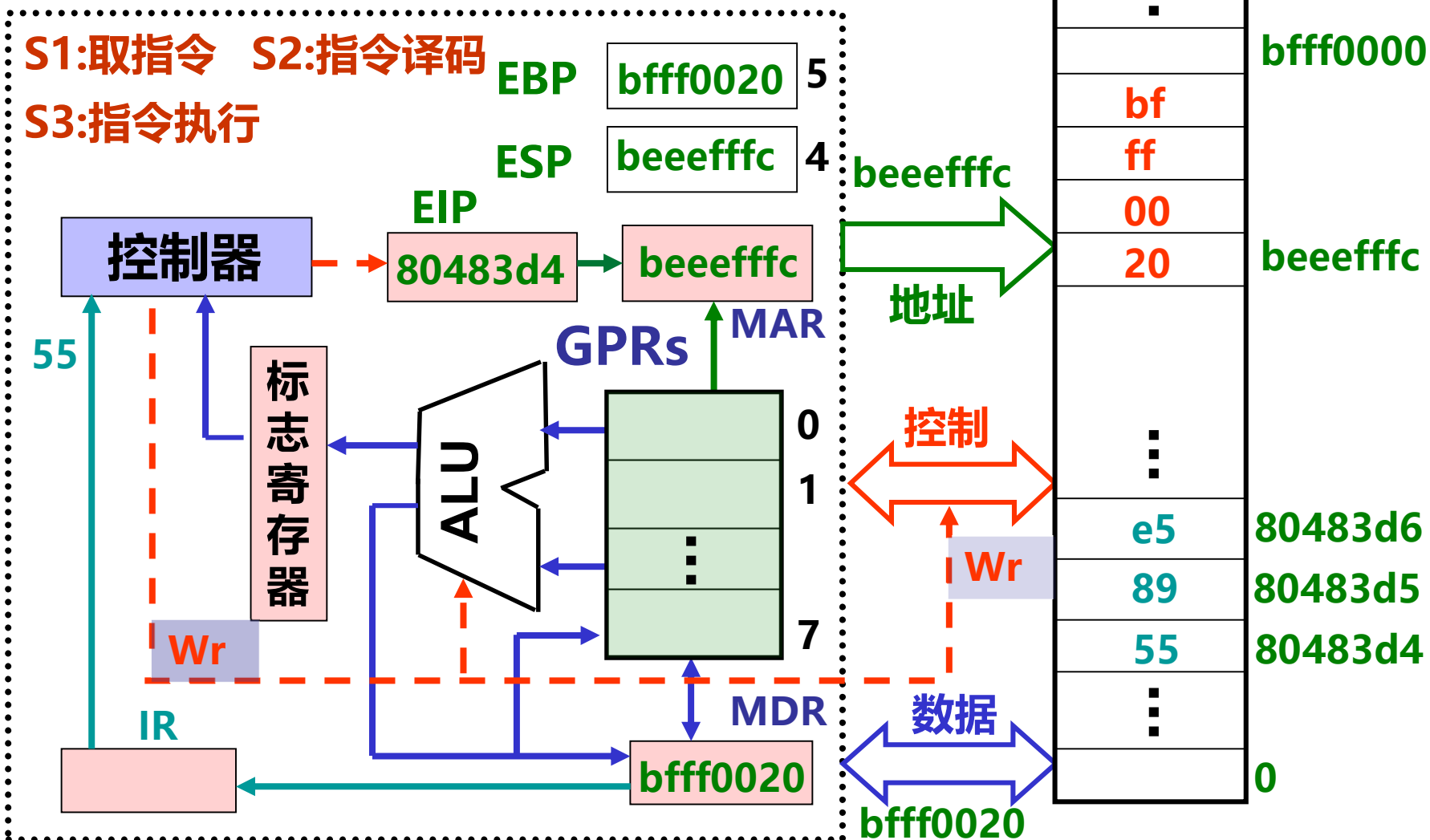
080483d4 <add>:

80483d4: 55 push %ebp

80483d5: 89 e5 mov %esp, %ebp

S1:取指令 S2:指令译码

S3:指令执行



# 开始执行下一条指令

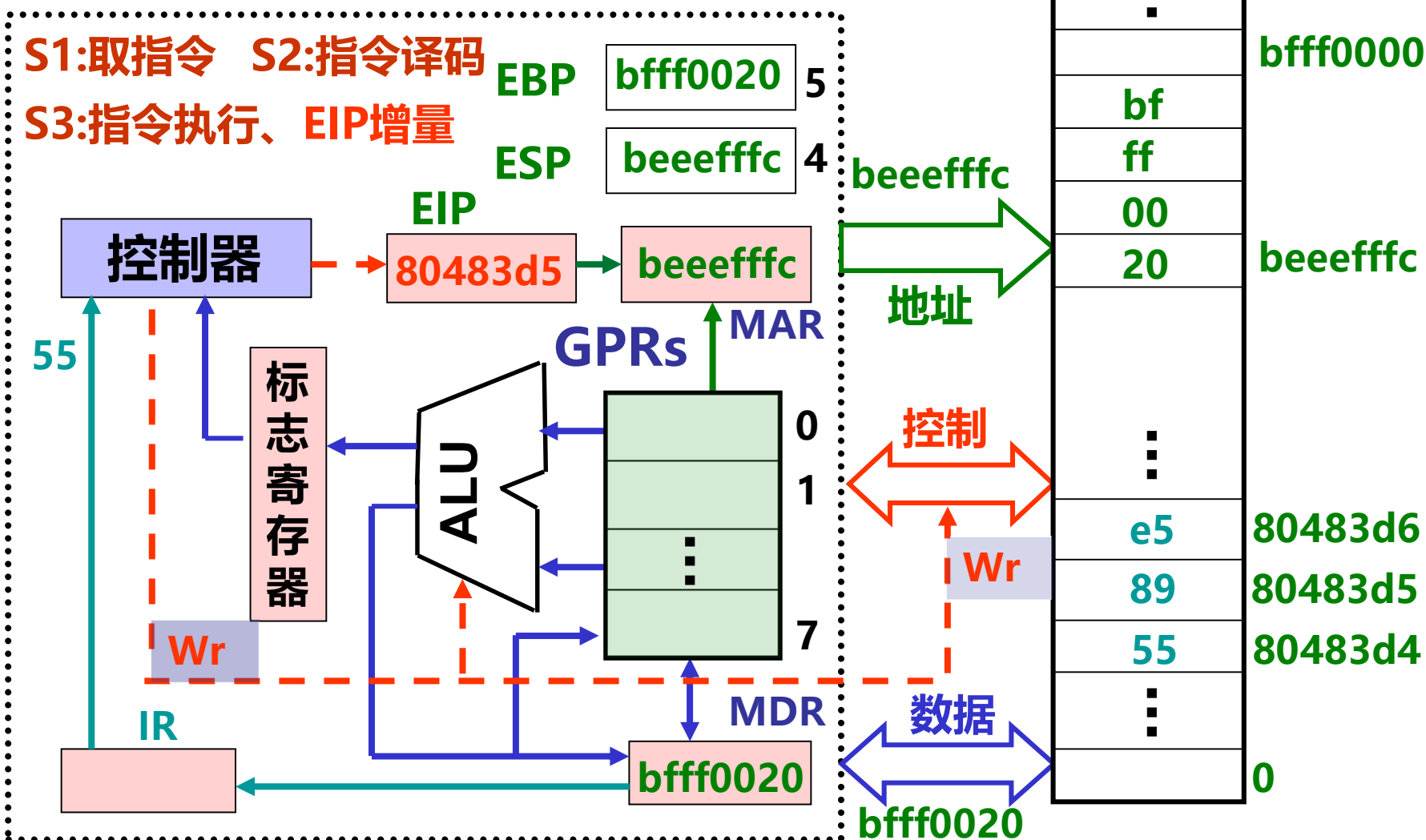
080483d4 <add>:

80483d4: 55 push %ebp

→80483d5: 89 e5 mov %esp, %ebp

S1:取指令 S2:指令译码

S3:指令执行、EIP增量



# 传送指令举例

---

将以下 Intel格式 指令转换为 AT&T格式 指令，并说明功能。

```
push    ebp
mov     ebp, esp
mov     edx, DWORD PTR [ebp+8]
mov     bl, 255
mov     ax, WORD PTR [ebp+edx*4+8]
mov     WORD PTR [ebp+20], dx
lea     eax, [ecx+edx*4+8]
```

```
pushl    %ebp                //R[esp]←R[esp]-4, M[R[esp]] ←R[ebp], 双字
movl     %esp, %ebp          //R[ebp] ←R[esp], 双字
movl     8(%ebp), %edx        //R[edx] ←M[R[ebp]+8], 双字
movb     $255, %bl           //R[bl]←255, 字节
movw     8(%ebp,%edx,4), %ax  //R[ax]←M[R[ebp]+R[edx]×4+8], 字
movw     %dx, 20(%ebp)        //M[R[ebp]+20]←R[dx], 字
leal     8(%ecx,%edx,4), %eax //R[eax]←R[ecx]+R[edx]×4+8, 双字
```