



南京大學  
NANJING UNIVERSITY



# IA-32中的定点算术运算指令

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



## (2) 定点算术运算指令

- 加 / 减运算 ( 影响标志、不区分无/带符号 )

ADD : 加 , 包括add**b**、add**w**、add**l**等

SUB : 减 , 包括sub**b**、sub**w**、sub**l**等

- 增1 / 减1运算 ( 影响除CF以外的标志、不区分无/带符号 )

INC : 加 , 包括inc**b**、inc**w**、inc**l**等

DEC : 减 , 包括dec**b**、dec**w**、dec**l**等

- 取负运算 ( 影响标志、若对0取负 , 则结果为0且CF清0 , 否则CF置1 )

NEG : 取负 , 包括neg**b**、neg**w**、neg**l**等

- 比较运算 ( 做减法得到标志、不区分无/带符号 )

CMP : 比较 , 包括cmp**b**、cmp**w**、cmp**l**等

- 乘 / 除运算 ( 不影响标志、区分无/带符号 )

MUL / IMUL : 无符号乘 / 带符号乘

DIV / IDIV : 带无符号除 / 带符号除

# 整数乘除指令



- 乘法指令：可给出一个、两个或三个操作数
  - 若给出一个操作数SRC，则另一个源操作数隐含在AL/AX/EAX中，将SRC和累加器内容相乘，结果存放在AX（16位）或DX-AX（32位）或EDX-EAX（64位）中。DX-AX表示32位乘积的高、低16位分别在DX和AX中。  $n\text{位} \times n\text{位} = 2n\text{位}$
  - 若指令中给出两个操作数DST和SRC，则将DST和SRC相乘，结果在DST中。  $n\text{位} \times n\text{位} = n\text{位}$
  - 若指令中给出三个操作数REG、SRC和IMM，则将SRC和立即数IMM相乘，结果在REG中。  $n\text{位} \times n\text{位} = n\text{位}$
- 除法指令：只明显指出除数，用EDX-EAX中内容除以指定的除数
  - 若为8位，则16位被除数在AX寄存器中，商送回AL，余数在AH
  - 若为16位，则32位被除数在DX-AX寄存器中，商送回AX，余数在DX
  - 若为32位，则被除数在EDX-EAX寄存器中，商送EAX，余数在EDX

# 定点算术运算指令汇总



指令	显式操作数	影响的常用标志	操作数类型	AT&T 指令助记符	对应 C 运算符
ADD	2 个	OF、ZF、SF、CF	无/带符号整数	addb、addw、addl	+
SUB	2 个	OF、ZF、SF、CF	无/带符号整数	subb、subw、subl	-
INC	1 个	OF、ZF、SF	无/带符号整数	incb、incw、incl	++
DEC	1 个	OF、ZF、SF	无/带符号整数	decb、decw、decl	--
NEG	1 个	OF、ZF、SF、CF	无/带符号整数	negb、negw、negl	-
CMP	2 个	OF、ZF、SF、CF	无/带符号整数	cmpb、cmpw、cmpl	<, <=, >, >=
MUL	1 个	OF、CF	无符号整数	mulb、mulw、mull	*
MUL	2 个		无符号整数	mulb、mulw、mull	*
MUL	3 个		无符号整数	mulb、mulw、mull	*
IMUL	1 个		带符号整数	imulb、imulw、imull	*
IMUL	2 个		带符号整数	imulb、imulw、imull	*
IMUL	3 个		带符号整数	imulb、imulw、imull	*
DIV	1 个	无	无符号整数	divb、divw、divl	/, %
IDIV	1 个	无	带符号整数	idivb、idivw、idivl	/, %

# 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 }
```

若  $i = 2147483647$  ,  $j = 2$  , 则执行结果是什么 ?

```
int main ( ) {
    int t1 = 2147483647;
    int t2 = 2;
    int sum = add (t1, t2);
    printf( "sum=%d" ;sum);
}
```

“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
```

此时 ,  $R[ecx] = 0x2$

$R[edx] = 0x7fffffff$

$\text{add } \%edx, \%eax$

$2147483647 = 2^{31} - 1$   
 $= 011 \dots 1B = 0x7fffffff$

add函数从80483d4开始 ! 执行add时 , 起始EIP=?

# IA-32的寄存器组织

编号	8 位寄存器	16 位寄存器	32 位寄存器	64 位寄存器	128 位寄存器
000	AL	AX	EAX	MM0 / ST(0)	XMM0
001	CL	CX	ECX	MM1 / ST(1)	XMM1
010	DL	DX	EDX	MM2 / ST(2)	XMM2
011	BL	BX	EBX	MM3 / ST(3)	XMM3
100	AH	SP	ESP	MM4 / ST(4)	XMM4
101	CH	BP	EBP	MM5 / ST(5)	XMM5
110	DH	SI	ESI	MM6 / ST(6)	XMM6
111	BH	DI	EDI	MM7 / ST(7)	XMM7

此时，R[eax]=0x2，R[edx]=0x7fffffff

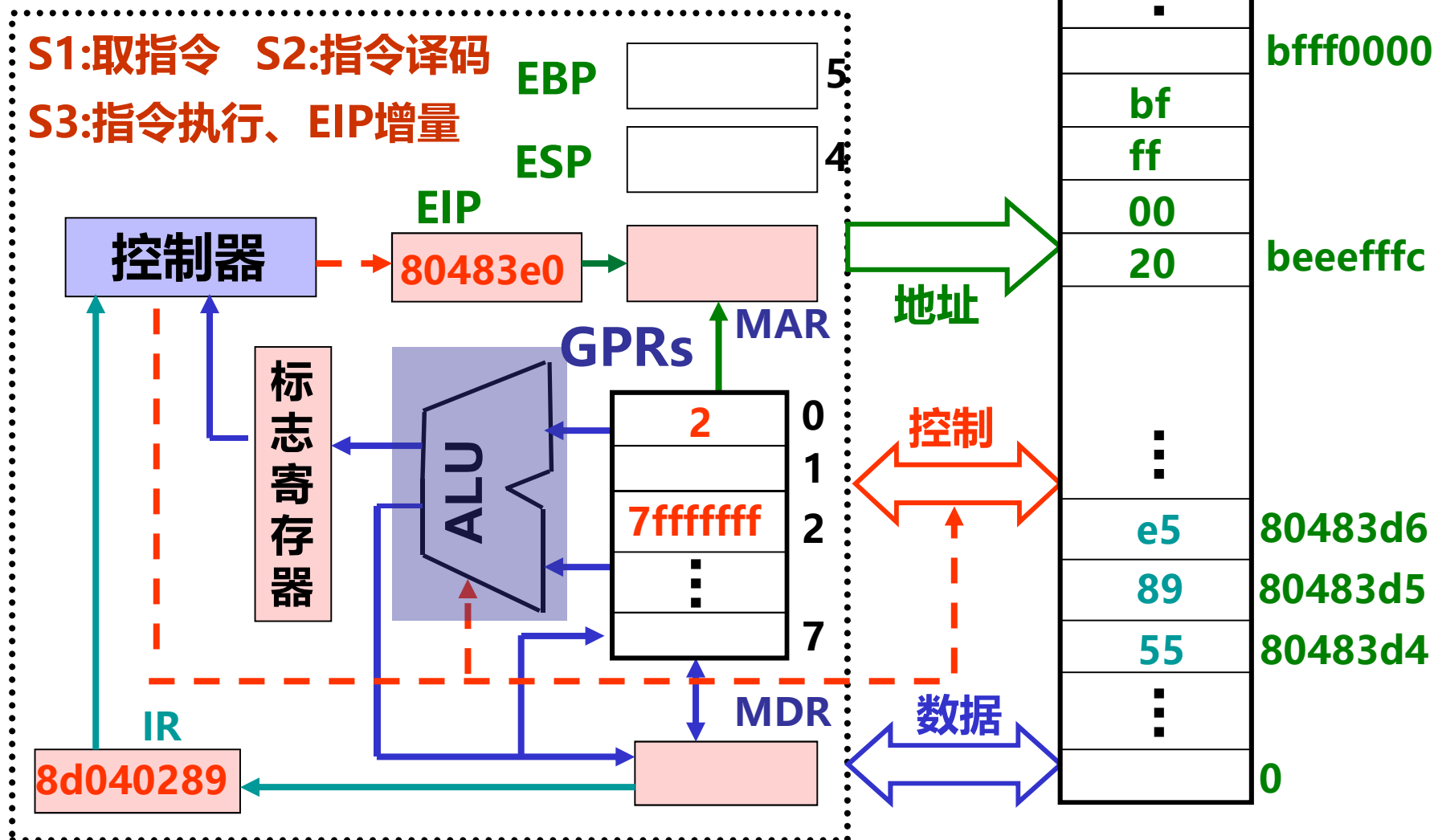
即：0号寄存器中为0x2；2号寄存器中为0x7fffffff

功能 :  $R[edx] \leftarrow R[edx] + R[edx] * 1$

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



# ALU长啥样呢？



- 试想一下ALU中有哪些部件？（想想厨房做菜用什么工具？）

- 补码加/减器（可以干什么？）

- 带符号整数加、减
- 无符号整数加、减

ALU如何实现呢？

- ~~乘法器~~？（为什么可以没有？）

- 可用加/减+移位实现，也可有独立乘法器
- 带符号乘和无符号乘是独立的部件

- ~~除法器~~？（为什么可以没有？）

- 可用加/减+移位实现，也可有独立除法器
- 带符号除和无符号除是独立的部件

- 各种逻辑运算部件（可以干什么？）

- 非、与、或、非、前置0个数、前置1个数.....



# ALU结构原理

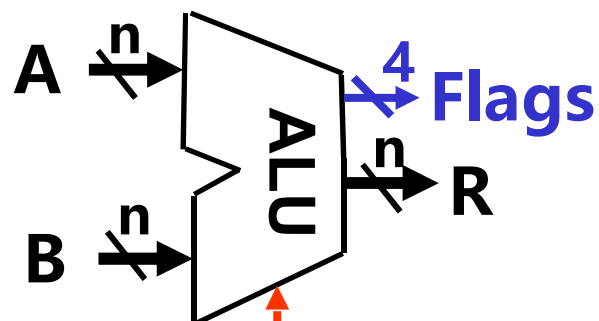
在ALU中执行：

$R[edx] = 0x2$

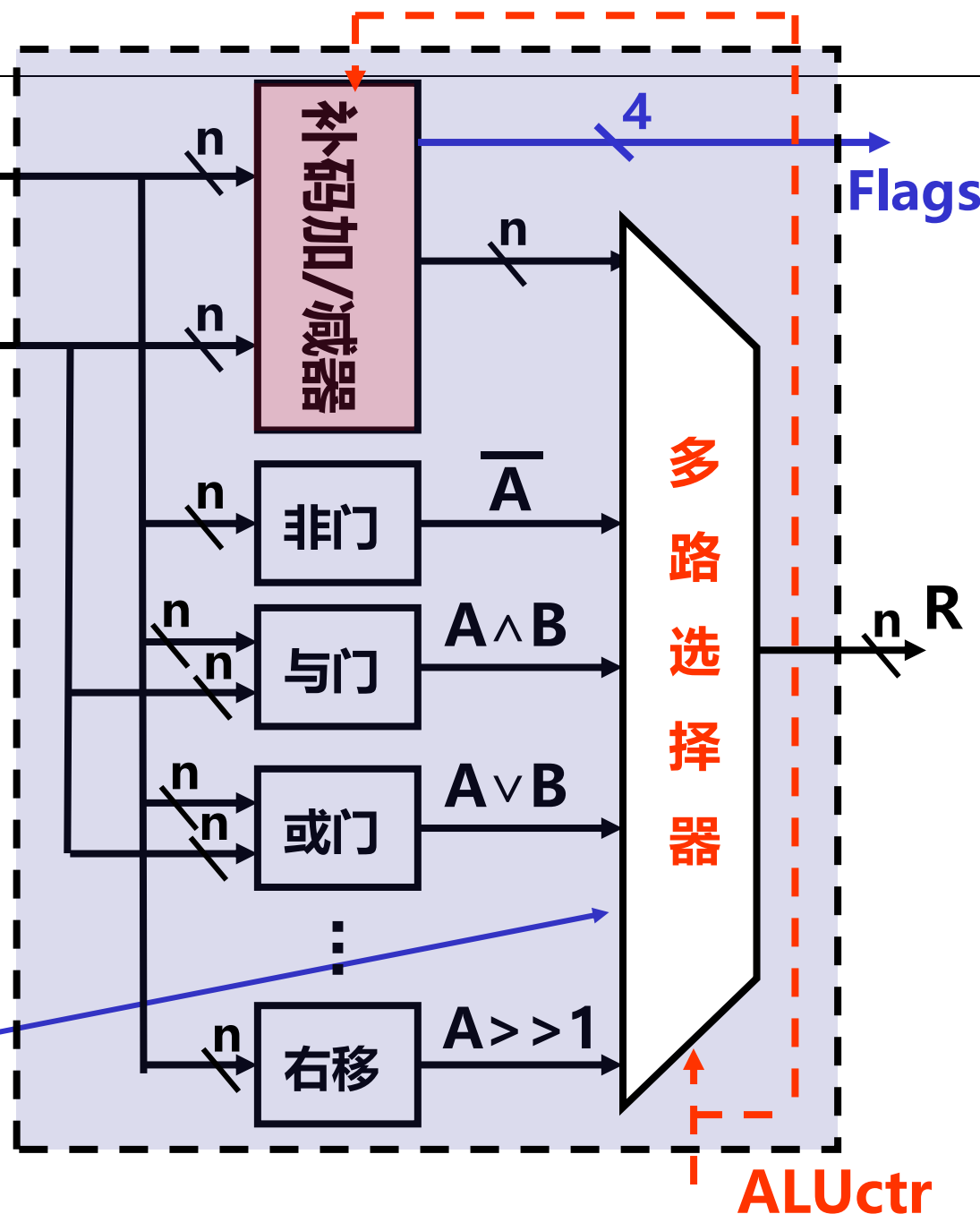
$R[edx] = 0x7fffffff$

$R[edx] + R[edx] = ?$

ALU的符号是  
什么样的？



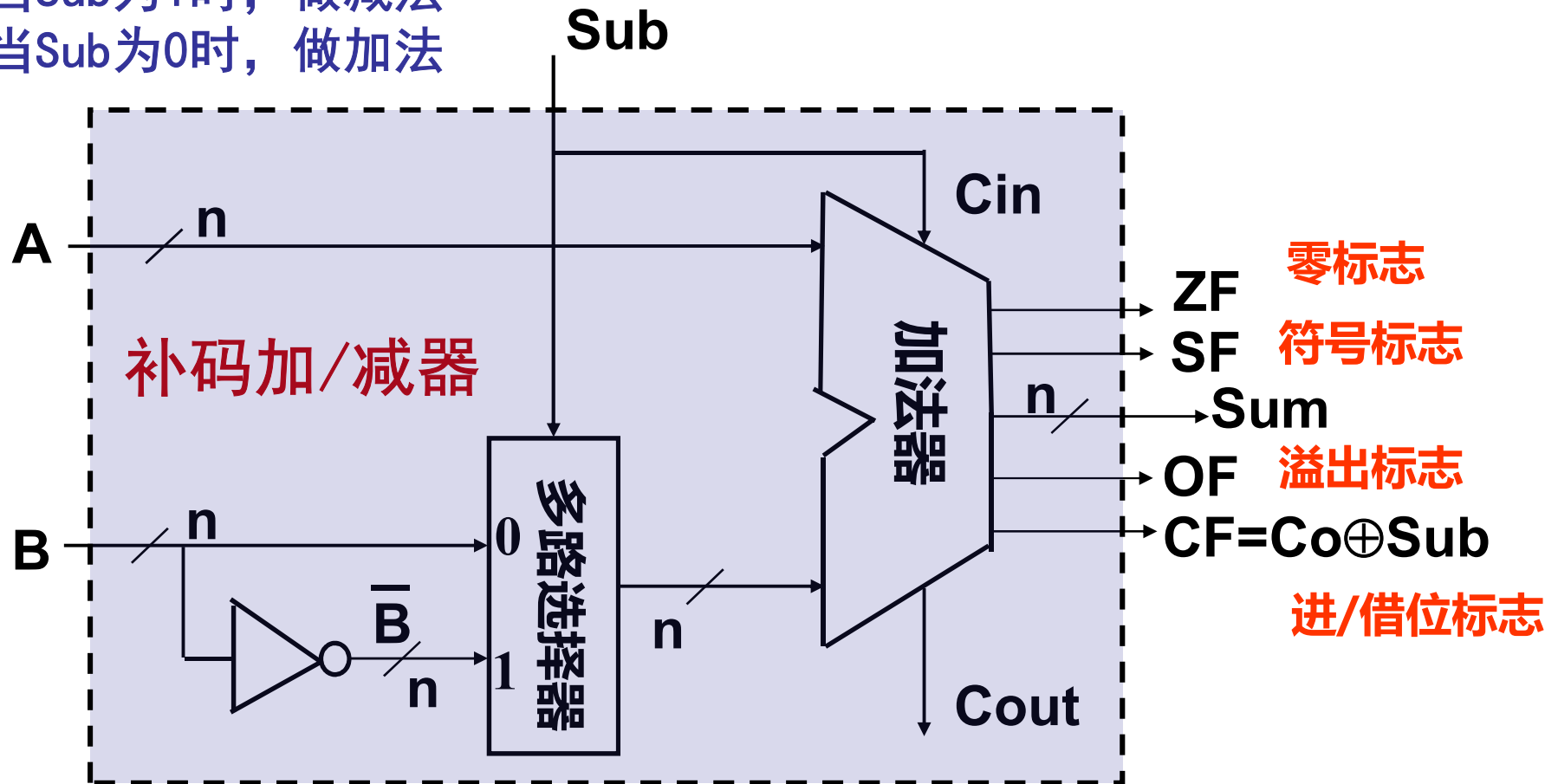
猜猜这是什么？



# 回顾：补码加/减器



当Sub为1时，做减法  
当Sub为0时，做加法



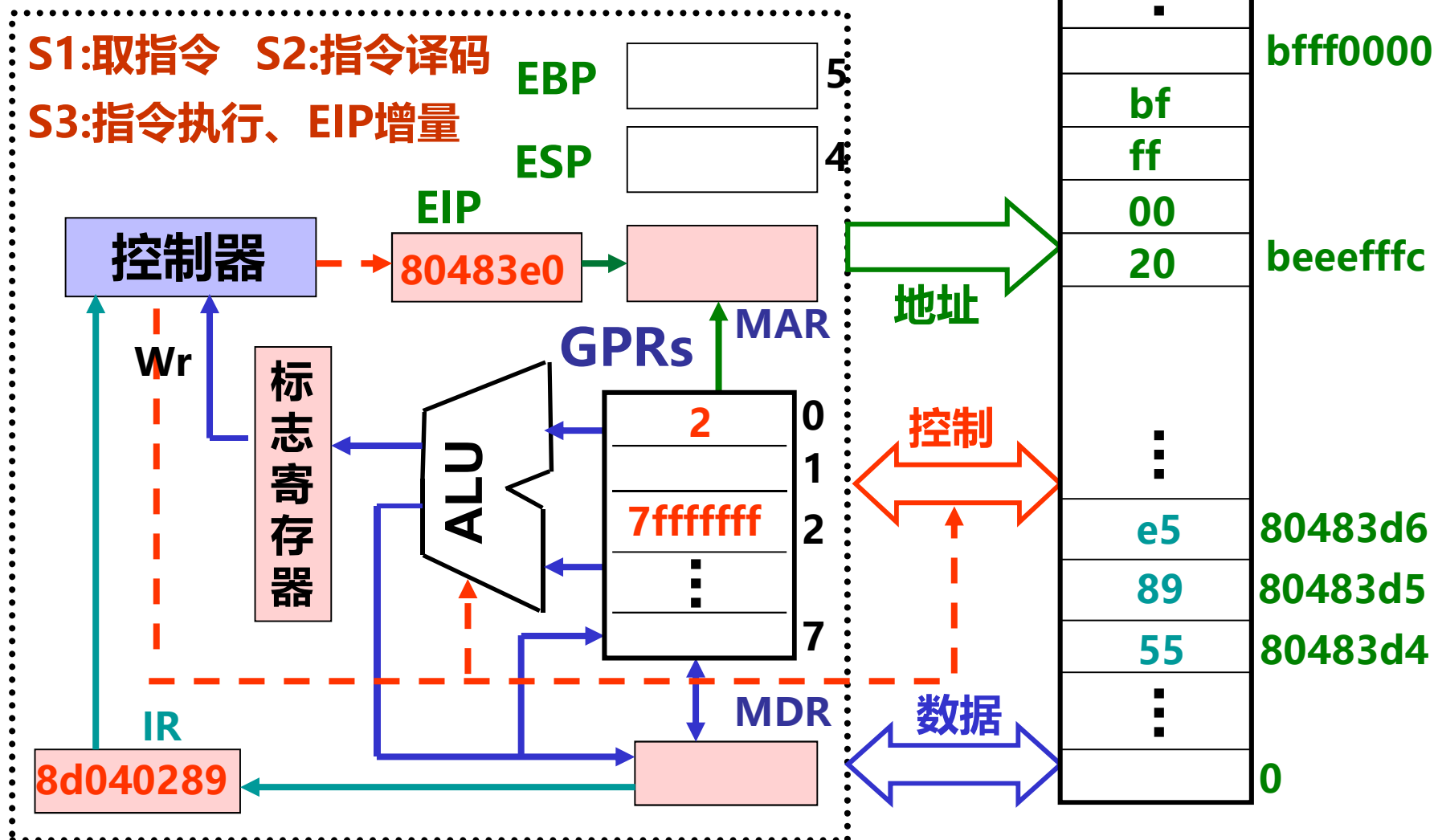
A : 0000 0000 0000 0000 0000 0000 0000 0010  
B : 0111 1111 1111 1111 1111 1111 1111 1111  
sum : 1000 0000 0000 0000 0000 0000 0000 0001

功能：  $R[edx] + R[edx] * 1 \leftarrow R[edx]$  (执行前)

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

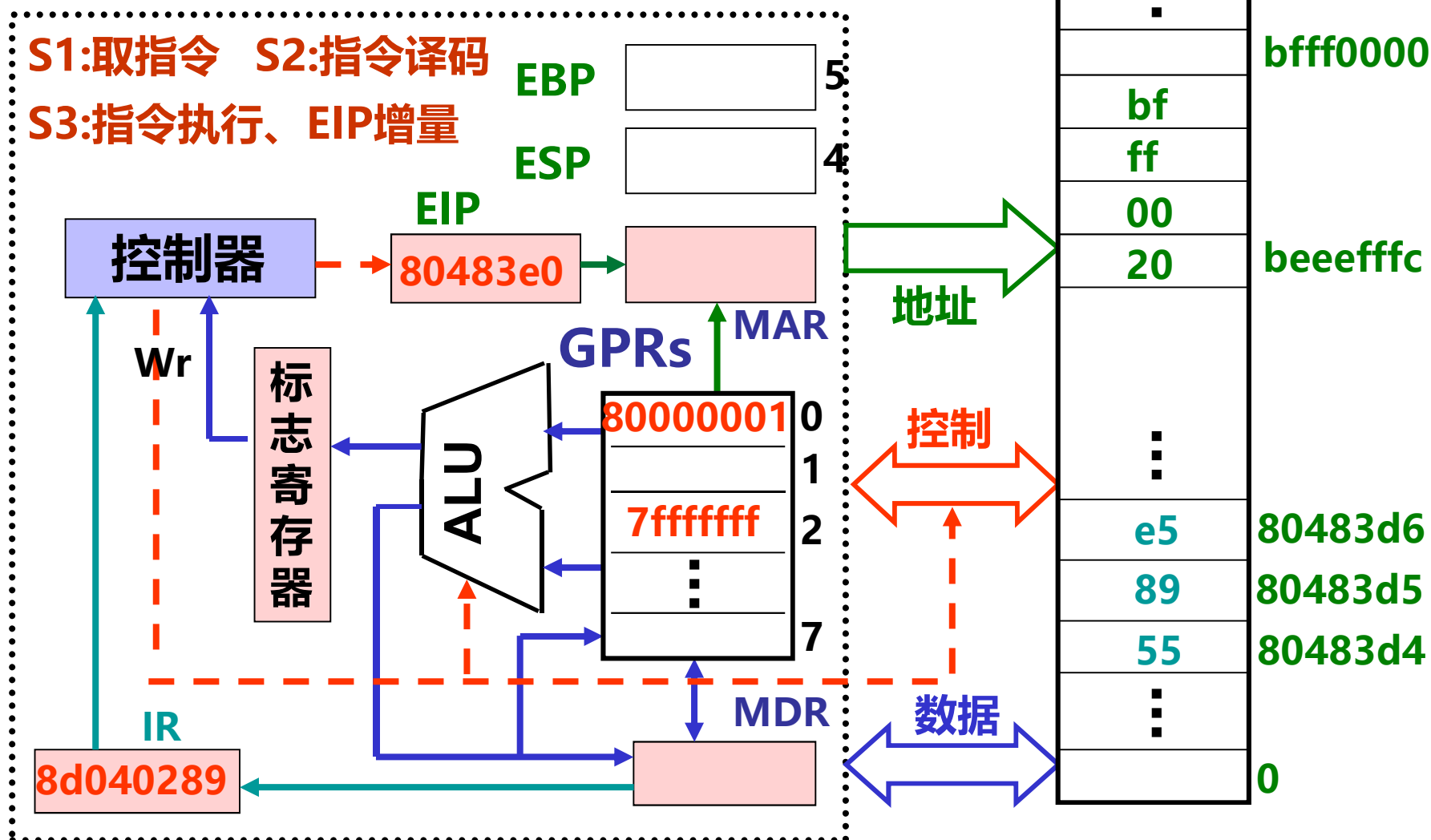


**功能：R[edx] ← R[edx] + R[edx] \* 1 (执行后)**

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



# 程序执行的结果

---

```
int add ( int i, int j ) {  
    return i+j;  
}  
  
int main ( ) {  
    int t1 = 2147483647;  
    int t2 = 2;  
    int sum = add (t1, t2);  
    printf(" sum=%d" , sum);  
}
```

**sum的机器数和  
值分别是什么？**

**sum=0x80000001  
sum=-2147483647**

**两个正数相加结果怎么为负数呢？**

**因为计算机是一种模运算系统！**

**高位有效数字被丢失，即发生了溢出**

# 定点加法指令举例



- 假设  $R[ax]=FFFAH$  ,  $R[bx]=FFF0H$  , 则执行以下指令后

**“addw %bx, %ax”**

AX、BX中的内容各是什么？标志CF、OF、ZF、SF各是什么？要求分别将操作数作为**无符号数**和**带符号整数**解释并验证指令执行结果。

**解：功能：** $R[ax] \leftarrow R[ax] + R[bx]$ ，指令执行后的结果如下

**$R[ax]=FFFAH+FFF0H=FFEAH$ ，BX中内容不变**

**$CF=1$ ， $OF=0$ ， $ZF=0$ ， $SF=1$**

**若是无符号整数运算，则 $CF=1$ 说明结果溢出**

**验证：**FFFA的真值为 $65535-5=65530$ ，FFF0的真值为65515

**FFEA的真值为 $65535-21=65514 \neq 65530+65515$ ，即溢出**

**若是带符号整数运算，则 $OF=0$ 说明结果没有溢出**

**验证：**FFFA的真值为-6，FFF0的真值为-16

**FFEA的真值为 $-22=-6+(-16)$ ，结果正确，无溢出**

# 定点乘法指令举例



- 假设  $R[ eax ] = 000000B4H$  ,  $R[ ebx ] = 00000011H$  ,  
 $M[ 000000F8H ] = 000000A0H$  , 请问 :

(1) 执行指令 “**mulb %bl**” 后 , 哪些寄存器的内容会发生变化 ? 是否与执行 “**imulb %bl**” 指令所发生的变化一样 ? 为什么 ? 请用该例给出的数据验证你的结论。

解 : “**mulb %bl**” 功能为  $R[ ax ] \leftarrow R[ al ] \times R[ bl ]$  , 执行结果如下

$R[ ax ] = B4H \times 11H$  ( 无符号整数180和17相乘 )

$R[ ax ] = 0BF4H$  , 真值为  $3060 = 180 \times 17$

“**imulb %bl**” 功能为  $R[ ax ] \leftarrow R[ al ] \times R[ bl ]$

$R[ ax ] = B4H \times 11H$  ( 带符号整数-76和17相乘 )

若  $R[ ax ] = 0BF4H$  , 则真值为  $3060 \neq -76 \times 17$

$R[ al ] = F4H$  ,  $R[ ah ] = ?$  **AH中的变化不一样 !**

$R[ ax ] = FAF4H$  , 真值为  $-1292 = -76 \times 17$

无符号乘 :

$$\begin{array}{r} 1011\ 0100 \\ \times\ 0001\ 0001 \\ \hline 1011\ 0100 \\ 1011\ 0100 \\ \hline 0000\ 1011\ 1111\ 0100 \\ \hline \end{array}$$

**AH = ?      AL = ?**

对于带符号乘 , 若积只取低n位 , 则和无符号相同 ; 若取2n位 , 则采用 “**布斯**” 乘法





# 定点乘法指令举例

---

- 假设  $R[ecx]=000000B4H$  ,  $R[ebx]=00000011H$  ,  
 $M[000000F8H]=000000A0H$  , 请问 :

(2) 执行指令 “`imull $-16, (%eax,%ebx,4), %eax`” 后哪些寄存器和存储单元发生了变化? 乘积的机器数和真值各是多少?

解: “`imull -16, (%eax,%ebx,4),%eax`”

功能为  $R[ecx] \leftarrow (-16) \times M[R[ecx] + R[ebx] \times 4]$  , 执行结果如下

$$R[ecx] + R[ebx] \times 4 = 000000B4H + 00000011H \ll 2 = 000000F8H$$

$$R[ecx] = (-16) \times M[000000F8H]$$

$$= (-16) \times 000000A0H \text{ (带符号整数乘)}$$

$$= 16 \times (-000000A0H)$$

$$= FFFFFFF60H \ll 4$$

$$= FFFFFFF600H$$

EAX中的真值为-2560

# 整数乘指令

---

- 乘法指令：可给出一个、两个或三个操作数
  - 若给出一个操作数SRC，则另一个源操作数隐含在AL/AX/EAX中，将SRC和累加器内容相乘，结果存放在AX（16位）或DX-AX（32位）或EDX-EAX（64位）中。DX-AX表示32位乘积的高、低16位分别在DX和AX中。[BACK](#)
  - 若指令中给出两个操作数DST和SRC，则将DST和SRC相乘，结果在DST中。
  - 若指令中给出三个操作数REG、SRC和IMM，则将SRC和立即数IMM相乘，结果在REG中。[BACK](#)