第三章 程序的转换与机器级表示

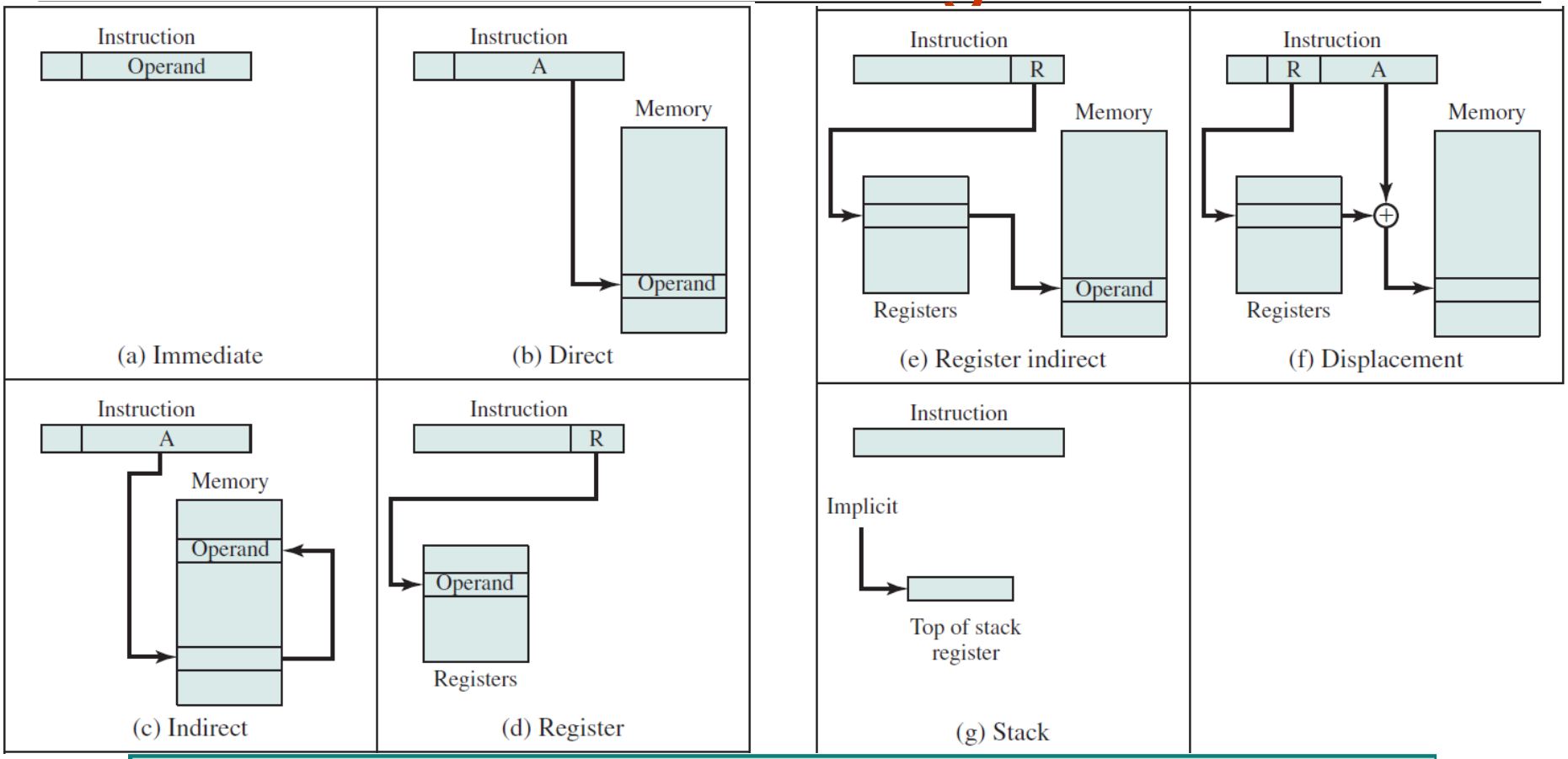
（附加材料）

x86 Addressing Modes

Mode	Algorithm
Immediate	Operand = A
Register Operand	LA = R
Displacement	LA = (SR) + A
Base	LA = (SR) + (B)
Base with Displacement	LA = (SR) + (B) + A
Scaled Index with Displacement	LA = (SR) + (I) × S + A
Base with Index and Displacement	LA = (SR) + (B) + (I) + A
Base with Scaled Index and Displacement	LA = (SR) + (I) × S + (B) + A
Relative	LA = (PC) + A

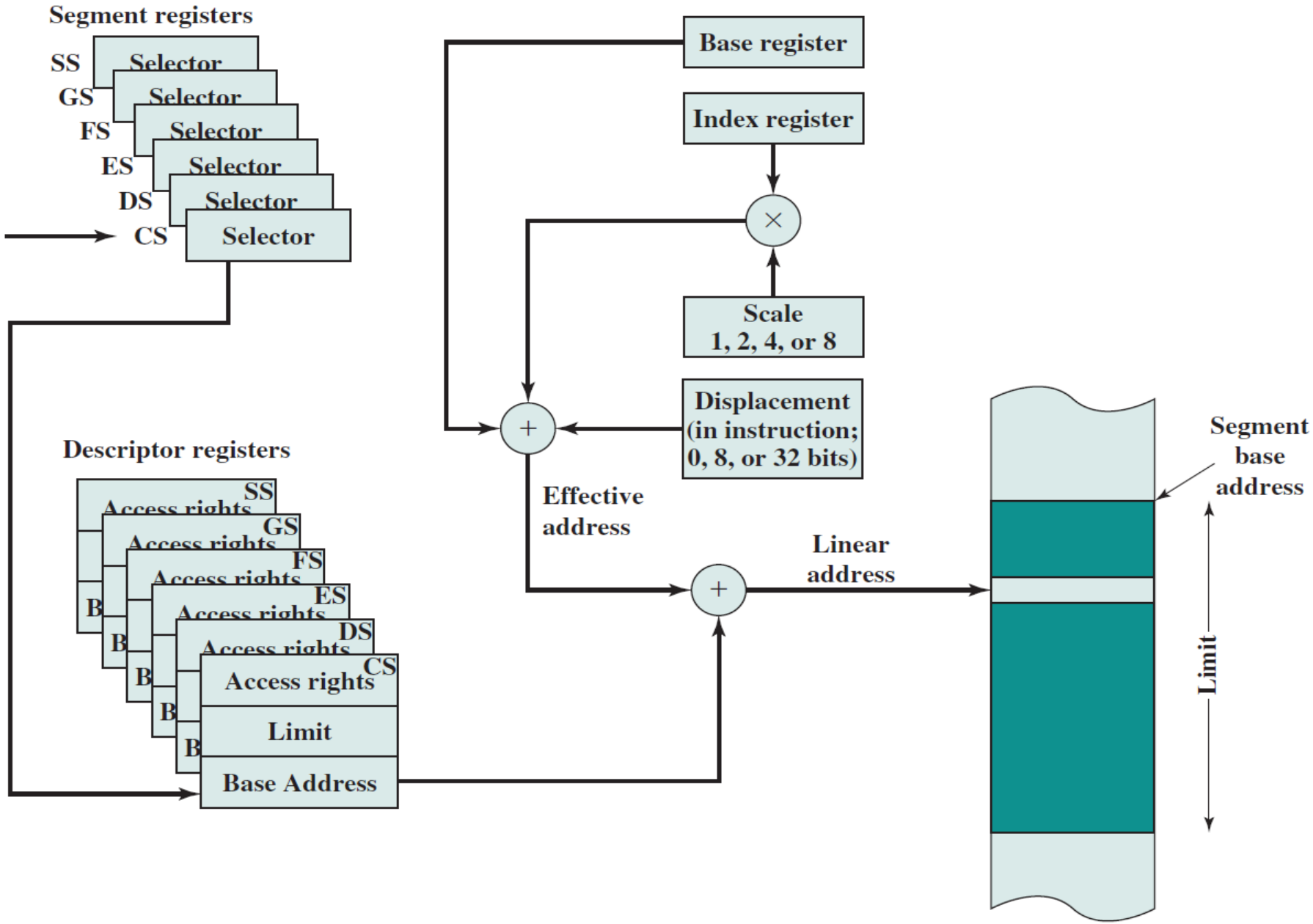
LA = linear address
(X) = contents of X
SR = segment register
PC = program counter
A = contents of an address field in the instruction
R = register
B = base register
I = index register
S = scaling factor

Basic Addressing Modes



Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

x86 Addressing Mode Calculation



变长参数函数调用

```
#include <stdarg.h>

int max_int( int n, ... )  {
  va_list ap;
  int i, current, largest;

  va_start(ap, n);
  largest = va_arg(ap, int);

  for( i = 1; i < n; i++ ) {
    current = va_arg(ap, int);
    if( current > largest )
      largest = current;
  }

  va_end(ap);
  return largest;
}
```

va_start(va_list arg_ptr, prev_param);

- sets **arg_ptr** to the first optional argument in the list of arguments that's passed to the function. The argument **arg_ptr** must have the va_list type. The argument **prev_param** is the name of the required parameter that immediately precedes the first optional argument in the argument list.
- must be used before va_arg is used for the first time.

type va_arg(va_list arg_ptr, type);

- retrieves a value of **type** from the location that's given by **arg_ptr**, and increments **arg_ptr** to point to the next argument in the list by using the size of **type** to determine where the next argument starts.

va_end(va_list arg_ptr);

- resets **arg_ptr** to NULL.
- must be called on each argument list that's initialized with va_start before the function returns.

变长参数函数调用

```
#include <stdarg.h>

int max_int( int n, ... )  {
  va_list ap;
  int i, current, largest;

  va_start(ap, n);
  largest = va_arg(ap, int);

  for( i = 1; i < n; i++ ) {
    current = va_arg(ap, int);
    if( current > largest )
      largest = current;
  }

  va_end(ap);
  return largest;
}
```

```
va_start(va_list arg_ptr,  prev_param);

type va_arg(va_list arg_ptr,  type);

va_end(va_list arg_ptr);
```

```
00000000 <max_int>:
... ...
  10:   8d 45 0c        lea    0xc(%ebp),%eax
  13:   89 45 f0        mov    %eax,-0x10(%ebp)
  16:   8b 45 f0        mov    -0x10(%ebp),%eax
  19:   8d 50 04        lea    0x4(%eax),%edx
  1c:   89 55 f0        mov    %edx,-0x10(%ebp)
  1f:   8b 00           mov    (%eax),%eax
  21:   89 45 f8        mov    %eax,-0x8(%ebp)
  24:   c7 45 fc 01 00 00 00   movl $0x1,-
0x4(%ebp)
  2b:   eb 20           jmp    4d <max_int+0x4d>
  2d:   8b 45 f0        mov    -0x10(%ebp),%eax
  30:   8d 50 04        lea    0x4(%eax),%edx
  33:   89 55 f0        mov    %edx,-0x10(%ebp)
  36:   8b 00           mov    (%eax),%eax
  38:   89 45 f4        mov    %eax,-0xc(%ebp)
  3b:   8b 45 f4        mov    -0xc(%ebp),%eax
  3e:   3b 45 f8        cmp    -0x8(%ebp),%eax
  41:   7e 06           jle    49 <max_int+0x49>
  43:   8b 45 f4        mov    -0xc(%ebp),%eax
  46:   89 45 f8        mov    %eax,-0x8(%ebp)
  49:   83 45 fc 01     addl   $0x1,-0x4(%ebp)
  4d:   8b 45 fc        mov    -0x4(%ebp),%eax
  50:   3b 45 08        cmp    0x8(%ebp),%eax
  53:   7c d8           jl     2d <max_int+0x2d>
  55:   8b 45 f8        mov    -0x8(%ebp),%eax
  58:   c9              leave
  59:   c3              ret
```

变长参数函数调用（示例）

```
#include <stdio.h>

char str[1024] = "";

void test( const char* fmt_str )
{
  int x = 0xAABBCCDD;
  printf(fmt_str, x);
  printf("\n");
}

main()
{
  scanf("%s", str);
  test(str);
}
```

```
0804847b <test>:
 804847b:   push   %ebp
 804847c:   mov    %esp,%ebp
 804847e:   sub    $0x18,%esp
 8048481:   movl   $0xaabbccdd,-
0xc(%ebp)
 8048488:   sub    $0x8,%esp
 804848b:   pushl  -0xc(%ebp)
 804848e:   pushl  0x8(%ebp)
 8048491:   call   8048330<printf>
 8048496:   add    $0x10,%esp
 8048499:   sub    $0xc,%esp
 804849c:   push   $0xa
 804849e:   call 8048360<putchar>
 80484a3:   add    $0x10,%esp
 80484a6:   leave
 80484a7:   ret
```

```
linuxer@debian:~/temp$ ./printbomb
%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x
aabbccdd,bf9f86a4,0,bf9f86b8,b77db020,b76710a9,aabbccdd,0,0,bf9f86b8,80484d
b,8049800,8049800,bf9f876c,b764139d,b77b63c4
```

数组参数

```
typedef int Data[4];

int Process( Data d ) {
  return d[0]+d[1]+d[2]+d[3];
}

void main() {
  Data d = {1,2,3,4};
  Process(d);
}
```

```
00000032 <main>:
  … …
  5e:   8d 45 f0          lea    -0x10(%ebp),%eax
  61:   50                push   %eax
  62:   e8 fc ff ff ff    call   63 <main+0x31>
```

```
00000000 <Process>:
  … …
  d:   8b 45 08          mov    0x8(%ebp),%eax
  10:   8b 10            mov    (%eax),%edx
  12:   8b 45 08          mov    0x8(%ebp),%eax
  15:   83 c0 04          add    $0x4,%eax
  18:   8b 00            mov    (%eax),%eax
  1a:   01 c2            add    %eax,%edx
  1c:   8b 45 08          mov    0x8(%ebp),%eax
  1f:   83 c0 08          add    $0x8,%eax
  22:   8b 00            mov    (%eax),%eax
  24:   01 c2            add    %eax,%edx
  26:   8b 45 08          mov    0x8(%ebp),%eax
  29:   83 c0 0c          add    $0xc,%eax
  2c:   8b 00            mov    (%eax),%eax
  2e:   01 d0            add    %edx,%eax
```

- 数组作为参数时，以指向数组首元素的指针值（而不是数组元素的值）进行传递
- 函数中对数组参数指向的数组元素的修改，将作用于实参数组上

结构参数

如何传递结构类型参数？

```
typedef struct {
  int id;   char name[6];   float age;
  char gender;
} Info;

int Process( Info inf ) {
  return (inf.id<1000 && inf.gender=='M')?
1 : 0;
}

Info Create() {
  Info f = {123456, "Jason", 12.5, 'M'};
  return f;
}

void main() {
  Info f = Create();
  Process(f);
}
```

```
00000000 <Process>:
  0:   55                   push   %ebp
  1:   89 e5                mov    %esp,%ebp
  3:   e8 fc ff ff ff       call   4 <Process+0x4>
  8:   05 01 00 00 00       add    $0x1,%eax
  d:   8b 45 08             mov    0x8(%ebp),%eax
 10:   3d e7 03 00 00       cmp    $0x3e7,%eax
 15:   7f 0f                jg     26 <Process+0x26>
 17:   0f b6 45 18          movzbl 0x18(%ebp),%eax
 1b:   3c 4d                cmp    $0x4d,%al
 1d:   75 07                jne    26 <Process+0x26>
 1f:   b8 01 00 00 00       mov    $0x1,%eax
 24:   eb 05                jmp    2b <Process+0x2b>
 … …

000005e5 <main>:
 5fe:   ff 75 fc            pushl  -0x4(%ebp)
 601:   ff 75 f8            pushl  -0x8(%ebp)
 604:   ff 75 f4            pushl  -0xc(%ebp)
 607:   ff 75 f0            pushl  -0x10(%ebp)
 60a:   ff 75 ec            pushl  -0x14(%ebp)
 60d:   e8 4e ff ff ff      call   560 <Process>
```

结构返回值

```
0000002d <Create>:
 30:   83 ec 20             sub    $0x20,%esp
 33:   e8 fc ff ff ff       call   34 <Create+0x7>
 38:   05 01 00 00 00       add    $0x1,%eax
 3d:   c7 45 ec 40 e2 01 00 movl   $0x1e240,-0x14(%ebp)
 44:   c7 45 f0 4a 61 73 6f movl   $0x6f73614a,-0x10(%ebp)
 4b:   66 c7 45 f4 6e 00    movw   $0x6e,-0xc(%ebp)
 … …
 5a:   c6 45 fc 4d          movb   $0x4d,-0x4(%ebp)
 5e:   8b 45 08             mov    0x8(%ebp),%eax
 61:   8b 55 ec             mov    -0x14(%ebp),%edx
 64:   89 10                mov    %edx,(%eax)
 66:   8b 55 f0             mov    -0x10(%ebp),%edx
 69:   89 50 04             mov    %edx,0x4(%eax)
 6c:   8b 55 f4             mov    -0xc(%ebp),%edx
 6f:   89 50 08             mov    %edx,0x8(%eax)
 72:   8b 55 f8             mov    -0x8(%ebp),%edx
 75:   89 50 0c             mov    %edx,0xc(%eax)
 78:   8b 55 fc             mov    -0x4(%ebp),%edx
 7b:   89 50 10             mov    %edx,0x10(%eax)
 7e:   8b 45 08             mov    0x8(%ebp),%eax
```

- 如何返回结构数据类型？

```
000005e5 <main>:
 5e8:   83 ec 20        sub    $0x20,%esp
 … …
 5f5:   8d 45 ec        lea    -0x14(%ebp),%eax
 5f8:   50              push   %eax
 5f9:   e8 8f ff ff ff  call   58d <Create>
```

```
Info Create() {
  Info f = {123456, "Jason", 12.5, 'M'};
  return f;
}
```