Array

%eax	0x10000000
%ecx	22
$0x10000004	0x10000004
0x10000012	0x50000 (0x0 is also correct)
0xFFFFFF8	NONE
(%eax, %ecx, 8)	44

Jump Table

（答案不唯一）

```
.section        .rodata
.align          4
.L6:
    .long       .L1
    .long       .L2
    .long       .L4
    .long       .L3
    .long       .L3
    .long       .L5
    .long       .L5

movl    [x], %edx
movl    [result], %eax
subl    $24, %edx
cmpl    $6, %edx
ja      .L2
jmp     *.L6(, %edx, 4)
//default
.L2:
movl    $3, %eax
jmp     .L8
//24
.L1:
addl    %eax, %eax
jmp     .L8
//27, 28
.L3:
addl    $10, %eax
jmp     .L8
//26
.L4:
imull   $2, %eax
```

```
//29, 30
.L5:
addl      $5, %eax

.L8:
movl %eax, [result]
```

Stack

(1) Instruction 1: %ebp:0x7FFFFFFF4 %esp:0x7FFFFFC0
(2) Instruction 2: %ebp:0x7FFFFFC0 %esp:0x7FFFFFC0
(3) Instruction 3: %ebp:0x7FFFFFC4 %esp:0x7FsFFFFC4

Conditional Move

(1). (答案不唯一，有必要时需要存储和恢复 caller-saved registers)

```
pushl %ebp
movl %esp, %ebp
movl [x], %ecx              //get x
movl [y], %edx              //get y
movl %ecx, %eax             //copy x
imull %edx, %ecx            //compute x*y and save to %ecx
addl %edx, %eax             // compute x+y
imull %edx, %eax            // compute (x+y)*y and save to %eax
pushl %ebx                  //save %ebx (callee-save)
movl [x], %ebx              //re-get x
cmpl %edx, %ebx             //compare x:y
cmovl %ecx, %eax            //if <, replace return value with x*y
popl %ebx                   //restore %ebx
popl %ebp
ret
```

(2).

GCC only uses conditional moves when the two expressions can be computed very easily (such as a single add instruction). Multiplications are too expensive to compute so there will not be any conditional move operations. (NOTE: you can try it by yourself!)