Parameters: x86-64: Non-optimized

```
matMult:
.LFB0:
  pushq  %rbp                    # save rbp
  movq  %rsp, %rbp               # New rbp
  movq  %rdi, -24(%rbp)          # a
  movq  %rsi, -32(%rbp)          # b
  movq  %rdx, -40(%rbp)          # c
  movl  %ecx, -44(%rbp)          # n
  movl  $0, -12(%rbp)            # i <-- 0
  jmp  .L2                       # goto .L2
```

c[i][j] = 0: x86-64

```
.L5:
  movl  $0, -8(%rbp)        # j <-- 0
  jmp  .L3                  # goto .L3
.L4:
  movl  -12(%rbp), %eax     # eax <-- i
  movslq  %eax, %rdx        # rdx <-- signExt(eax (i))
  movq  %rdx, %rax          # rax <-- rds (i)
  salq  $2, %rax            # rax <-- 4*rax (4i)
  addq  %rdx, %rax          # rax <-- rax + rdx (4i+i=5i)
  salq  $4, %rax            # rax <-- 16*rax (80i)
  movq  %rax, %rdx          # rdx <-- rax (80i)
```

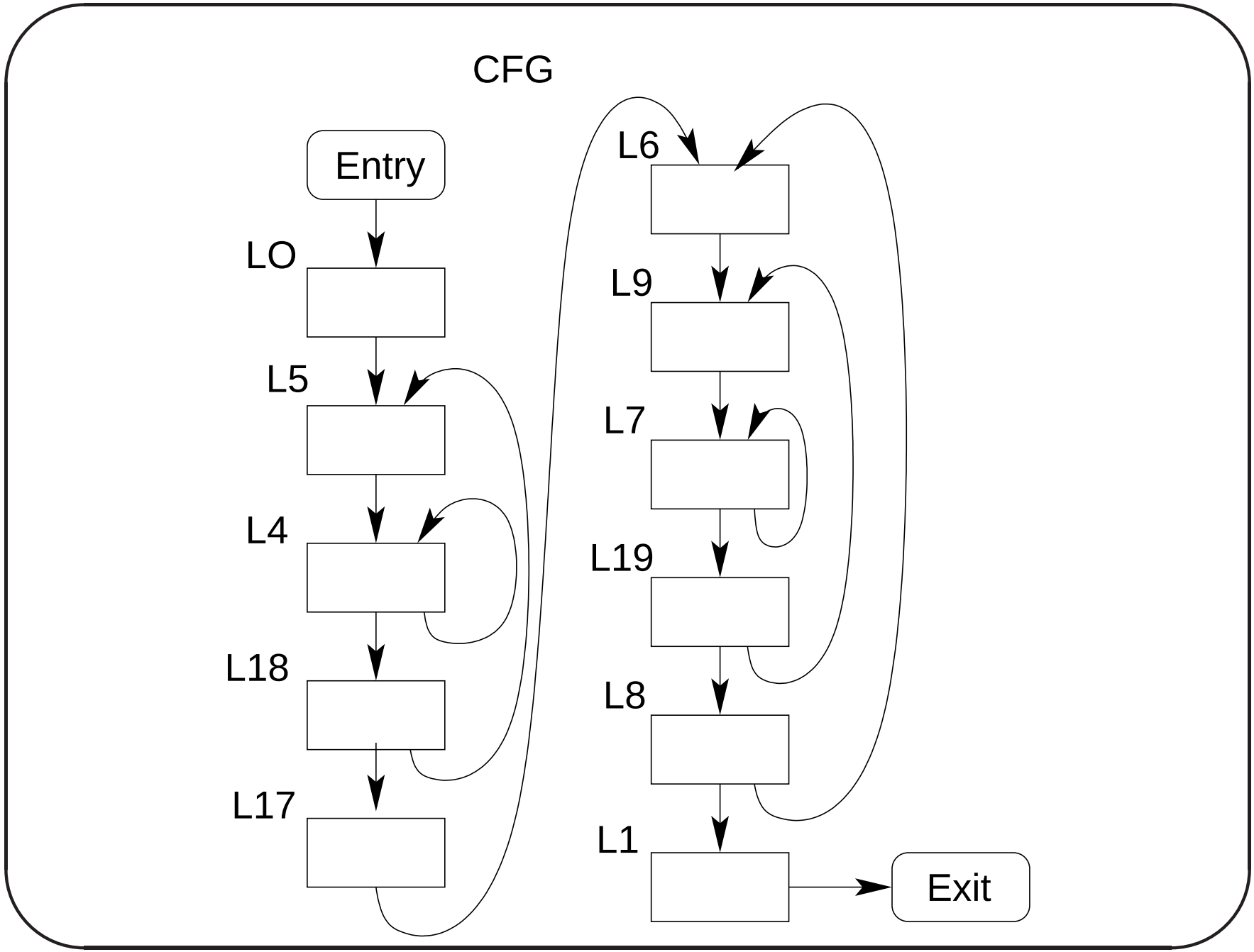
c[i][j] = 0: x86-64

```
  movq  -40(%rbp), %rax     # rax <-- Mem[rbp-40] (c)
  addq  %rax, %rdx          # rdx <-- c + 80i
  movl  -8(%rbp), %eax      # eax <-- j
  cltq                      # rax <-- signExt(eax (j))
  movl  $0, (%rdx,%rax,4)   # Mem[rdx+4*eax] =
                            # Mem[c+80i+4j]=c[i][j] <-- 0
  addl  $1, -8(%rbp)        # j <-- j+1
.L3
  movl  -8(%rbp), %eax      # eax <-- j
  cmpl  -44(%rbp), %eax     # if j < n
  jl  .L4                   #    goto .L4 (inner loop)
```

c[i][j] = 0: x86-64

```
  addl  $1, -12(%rbp)       # i <-- i+1
.L2:
  movl  -12(%rbp), %eax     # eax <-- i
  cmpl  -44(%rbp), %eax     # if i<n
  jl  .L5                   #    goto .L5
```

CFG: x86-64: Optimized Code


CFG
Entry
LO
L5
L4
L18
L17
L6
L9
L7
L19
L8
L1
Exit

Initial Block (L_0): x86-64: Optimized

```
.LFB0:
  testl  %ecx, %ecx          # ecx has the parameter n.
  jle  .L16                  # If n <= 0 , nothing to
                             # compute, so return.
  pushq  %r12                # save r12
  pushq  %rbp                # save base pointer
  pushq  %rbx                # save rbx
  movq  %rdi, %rbx           # rbx <-- a
  movq  %rsi, %r10           # r10 <-- b
  movq  %rdx, %rax           # rax <-- c
                             # ecx has the parameter n
  leal  -1(%rcx), %r8d       # r8d <-- rcx - 1 (n-1), in
```

```
                                #    lower 32-bis of r8
leaq  0(,%r8,4), %r12           # r12 <-- 4*r8 + 0 = 4(n-1)
movq  %rdx, %r11                # r11 <-- c
leaq  4(%rdx,%r12), %rdx        # rdx <-- 4+rdx+r12
                                #     rdx <-- 4 + c + 4(n-1)
                                #     rdx <-- c+4n
leaq  (%r12,%r8), %rcx          # rcx <-- r12+r8
                                #    rcx <-- 4(n-1)+(n-1)
                                #    rcx <-- 5(n-1)
leaq  (%r8,%rcx,4), %rcx        # rcx <-- r8+4*rcx
                                #    rcx <-- (n-1) + 20(n-1)
                                  #    rcx <-- 21(n-1)
leaq  84(%rax,%rcx,4), %rsi     # rsi <-- 84+rax+4*rcx
                                #     rsi <-- 84+c+84(n-1)
```

```
                                #     rsi <-- c + 84n
  movq  %r8, %rcx               # rcx <-- r8 (n-1)
  notq  %rcx                    # rcx <-- not of rcx bits
                                #     rcx <-- -n
  salq  $2, %rcx                # rcx <-- -4n
                                # Note: 2's complement of n
                                # 2^k - n = (2^k-1) - n + 1 =
                                # (2^k-1) - (n-1) =
                                # (1's complement of n)+1 or
                                # 1's complement of (n-1).
  jmp   .L5                     # goto .L5
.L16:
  ret                           # return as n <= 0
```

`c[i][j]=0` (L_5, L_4, L_{18}): x86-64: Optimized

Initial values: `rcx`:$-4n$, `rdx`:$c+4n$,
`rsi`:$c+(80+4)n$, $i=0$

```
.L18:
  addq  $80, %rdx     # rdx <-- rdx + 80 (next row (i))
  cmpq  %rsi, %rdx    # rsi = c + (80+4)n
  je  .L17
.L5:
  leaq  (%rcx,%rdx), %rax  # rax <-- rcx + rdx (c+4n+80i)
                           #  rax <-- -4n+(c+4n+80i) = c+80i
                           #  Initially i=0, ..., (n-1)
.L4:
```

```
movl   $0, (%rax)     # Loop for
addq   $4, %rax       # c[i][j] = 0. Starting from c+80i in
cmpq   %rdx, %rax     # rax, every 4 bytes is initialized
                      # to zero for j=0,1,..., n-1
jne    .L4            # if rax != rdx goto .L4 (loop)
jmp    .L18           # else goto .L18
```

An Example of `c[i][j]=0`

All `int` arrays are of size 20×20. So each row is of size 80 bytes. Let n=5.

`rcx`: -20, `rdx`: $c + 20$, `rsi`: $c + 5(80 + 4)$

An Example of `c[i][j]=0`

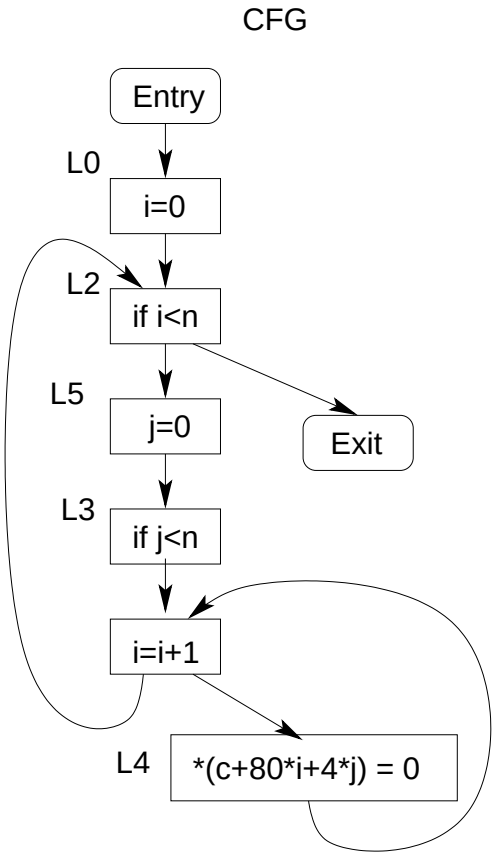
L_5 :	`rax:`	c	L_4 :	`c[0][3]:`	0
L_4 :	`c[0][0]:`	0		`rax:`	$c+16$
	`rax:`	$c+4$		`c[0][4]:`	0
	`c[0][1]:`	0		`rax:`	$c+20$
	`rax:`	$c+8$	L_{18} :	`rdx:`	100
	`c[0][2]:`	0	L_5 :	`rax:`	$c+80$
	`rax:`	$c+12$	L_4 :	`c[1][0]:`	0

An Example of `c[i][j]=0`

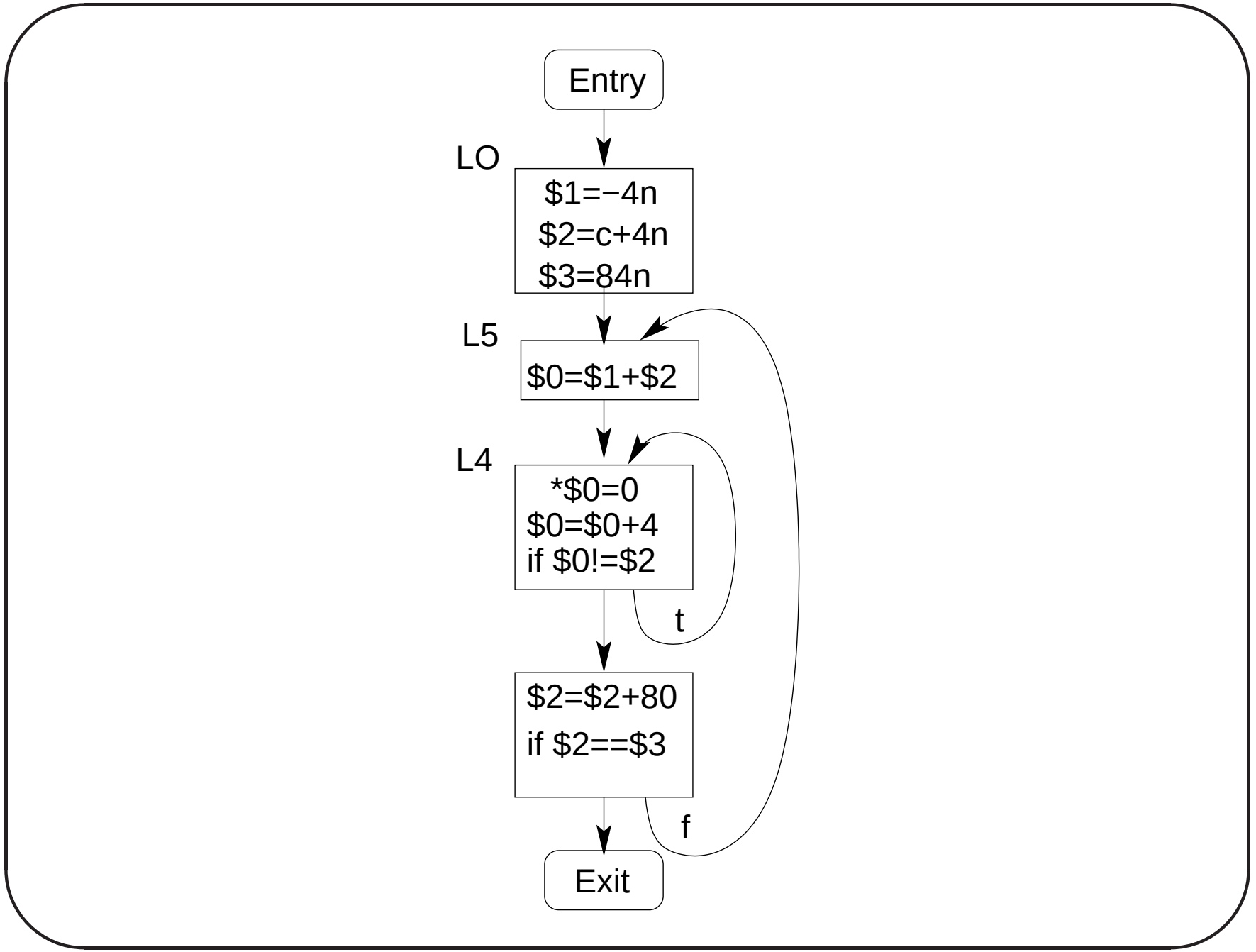
...	...
L_{18} :	`rdx:` 340
L_5 :	`rax:` $c + 320$
L_4:	`c[4][0]:0`
	`rax:` $c + 324$
	`c[4][1]:0`
	`rax:` $c + 328$

	`c[4][2]:0`
	`rax:` $c + 332$
	`c[4][3]:0`
	`rax:` $c + 336$
	`c[4][4]:0`
	`rax:` $c + 340$
L_{18}:	`rdx:` 420
	`goto` L_{17}

CFG: `c[i][j]=0` Optimized Code



CFG: `c[i][j]=0` Non-Optimized Code


Entry
LO
$1=−4n
$2=c+4n
$3=84n
L5
$0=$1+$2
L4
*$0=0
$0=$0+4
if $0!=$2
t
$2=$2+80
if $2==$3
f
Exit