COL 729 COMPILER OPTIMISATIONS

LAB 1
UNDERSTANDING LLVM IR AND CLANG OPTIMISATIONS

SUBMITTED BY-
NAMRATA JAIN
2018MCS2840

1. Optimisation level -O0

No optimisation

2. Optimization level -O2

At optimization level -O2, the compiler performs comprehensive optimization, which includes the following techniques:.

1. Global assignment of user variables to registers (register allocation).
2. Strength reduction and effective use of addressing modes.
3. Elimination of redundant instructions, known as common subexpression elimination
4. Elimination of instructions whose results are unused or that cannot be reached by a specified control flow, known as dead code elimination.
5.Algebraic simplification.
6. Movement of invariant code out of loops.
7. Compile-time evaluation of constant expressions, known as constant propagation.
8. Control flow simplification.
9. Instruction scheduling (reordering) for the target machine.
10. Loop unrolling and software pipelining -
If the loop does thingA and then thingB, we can move thingA out above the loop, then rotate the loop so it looks like thingB and then thingA.
11. Branch prediction

Specific to LLVM IR:

1. Repetitive Allocations were removed. Some of the alloc get replaced with phi nodes.

2. Loop-Closed SSA Form
It adds phi nodes for every live variable at the end of the basic block because this might expose optimizations done by other passes.

X86 ASSEMBLY:
Generates highly optimized code but has slow compilation time.

1. Move 0 replaced by xor of the register with itself.

2. More registers are used instead of storing values on stack.

1. LLVM IR

a. emptyloop O0

```
define i32 @emptyloop(i32, i8**) #0 {                ;defining function emptyloop with 2 arg. of type i32 and **i8 and return i32 (int 32bit value)
  %3 = alloca i32, align 4                            ;%3 address contains address of allocated space for an int32 variable
  %4 = alloca i8**, align 8                           ;%4=**char8
  %5 = alloca i64, align 8                            ;%5=int64 for i
  %6 = alloca i64, align 8                            ;%6=int64 for numiter
  store i32 %0, i32* %3, align 4                      ;%0 contains first arguement argc, %3=argc
  store i8** %1, i8*** %4, align 8                    ;%1 contains second arguement argv, %4=argv
  store i64 2147483646, i64* %6, align 8              ;numiter=int_max-1
  %7 = load i32, i32* %3, align 4                     ;%7=i
  %8 = icmp sge i32 %7, 2                             ;%8=result of comparison of argc and 2 (argc>=2)
  br i1 %8, label %9, label %15                       ;if %8=true then jump to label 9 else label 15

; <label>:9:                                        ; preds = %2
  %10 = load i8**, i8*** %4, align 8                      ;%10=argv
  %11 = getelementptr inbounds i8*, i8** %10, i64 1       ;get element ptr argv+1,
  %12 = load i8*, i8** %11, align 8                       ;%12=argv[1]
  %13 = call i32 @atoi(i8* %12) #2                        ;call atoi with arg as argv[1]
  %14 = sext i32 %13 to i64                               ;extend the result to 64bit,%14 typecast
  store i64 %14, i64* %6, align 8                         ;numiter=%14
  br label %15                                            ;jump to label 15

; <label>:15:                                       ; preds = %9, %2
  store i64 0, i64* %5, align 8                           ;i=0
  br label %16                                            ;jump to loop start 16

; <label>:16:                                       ; preds = %27, %15
  %17 = load i64, i64* %5, align 8                        ;%17=i
  %18 = load i64, i64* %6, align 8                        ;%18=numiter
  %19 = icmp ult i64 344865, %18                          ;%19=(magic_number<numiter)
  br i1 %19, label %20, label %22                         ;jump to label 20 if true else label 22

; <label>:20:                                       ; preds = %16
  %21 = load i64, i64* %6, align 8                        ;%21=numiter
  br label %23                                            ;branch to label 23

; <label>:22:                                       ; preds = %16
  br label %23

; <label>:23:                                       ; preds = %22, %20
  %24 = phi i64 [ %21, %20 ], [ 344865, %22 ]             ;if reached here from label 20 then %24=20 else 344865
  %25 = icmp ult i64 %17, %24                             ;comparing magic_number and numiter
  br i1 %25, label %26, label %30

; <label>:26:                                       ; preds = %23
  br label %27

; <label>:27:                                       ; preds = %26
  %28 = load i64, i64* %5, align 8                        ;%28=i
  %29 = add i64 %28, 1                                    ;i incremented by 1
  store i64 %29, i64* %5, align 8                         ;%29=i
  br label %16                                            ;jump to start of loop

; <label>:30:                                       ; preds = %23
  ret i32 0                                               ;return 0
}
```

b. emptyloop O2

```
define i32 @emptyloop(i32, i8** nocapture readonly) local_unnamed_addr #0 {
  %3 = icmp sgt i32 %0, 1                                                    ;%3=(argc>1)
  br i1 %3, label %4, label %8                                               ;if true then jump to label 4 else label 8 to return

; <label>:4:                                        ; preds = %2
  %5 = getelementptr inbounds i8*, i8** %1, i64 1                            ;%5=get element pointer (argv)
  %6 = load i8*, i8** %5, align 8, !tbaa !2                                  ;%6=argv[1]
  %7 = tail call i64 @strtol(i8* nocapture nonnull %6, i8** null, i32 10) #2 ;call strtol on argv[1]
  br label %8                                                                ;jump to return

; <label>:8:                                        ; preds = %4, %2
  ret i32 0                                                                  ;return 0
}

; Function Attrs: nounwind
declare i64 @strtol(i8* readonly, i8** nocapture, i32) local_unnamed_addr #1
```

Memory allocations removed for those variables which are not required.
Empty loop removed from the code.

c. fib O0

```
; Function Attrs: noinline nounwind optnone uwtable
define i32 @fib(i32) #0 {                           ;function fib with arg n and return int32
  %2 = alloca i32, align 4                          ;space allocated for arguements and local variables
  %3 = alloca i32, align 4
  store i32 %0, i32* %3, align 4                    ;%3=n storing n
  %4 = load i32, i32* %3, align 4                   ;%4=n
  %5 = icmp slt i32 %4, 2                           ;%5=(n<2)
  br i1 %5, label %6, label %7                      ;if true then label 6 to return value 1 else label 7

; <label>:6:                                        ; preds = %1
  store i32 1, i32* %2, align 4                     ;%2=1
  br label %15

; <label>:7:                                        ; preds = %1
  %8 = load i32, i32* %3, align 4                   ;%8=n
  %9 = sub nsw i32 %8, 1                            ;%9=n-1
  %10 = call i32 @fib(i32 %9)                       ;%10=fib(n-1)
  %11 = load i32, i32* %3, align 4                  ;%11=n
  %12 = sub nsw i32 %11, 2                          ;%12=n-2
  %13 = call i32 @fib(i32 %12)                      ;%13=fib(n-2)
  %14 = add nsw i32 %10, %13                        ;%14=fib(n-1)+fib(n-2)
  store i32 %14, i32* %2, align 4                   ;storing return value at %2
  br label %15                                      ;jump to end of function

; <label>:15:                                       ; preds = %7, %6
  %16 = load i32, i32* %2, align 4                  ;%16=%2
  ret i32 %16                                       ;return value 1
}
```

d. fib O2

```
define i32 @fib(i32) local_unnamed_addr #0 {
  %2 = icmp slt i32 %0, 2                          ;%2=(n<2)
  br i1 %2, label %12, label %3                    ;if true then jump to 12 to return 1 else label 3

; <label>:3:                                       ; preds = %1
  br label %4

; <label>:4:                                       ; preds = %3, %4
  %5 = phi i32 [ %9, %4 ], [ %0, %3 ]              ;if pred=4 then %9(n-2) else n
  %6 = phi i32 [ %10, %4 ], [ 1, %3 ]              ;if pred=4 then %10(added value) else 1
  %7 = add nsw i32 %5, -1                          ;%7=n-1
  %8 = tail call i32 @fib(i32 %7)                  ;%8=fib(n-1)
  %9 = add nsw i32 %5, -2                          ;%9=n-2
  %10 = add nsw i32 %8, %6                         ;%10=fib(n-1)+fib(n-2)
  %11 = icmp slt i32 %5, 4                         ;%11=compare 4 and value from phi node
  br i1 %11, label %12, label %4

; <label>:12:                                      ; preds = %4, %1
  %13 = phi i32 [ 1, %1 ], [ %10, %4 ]
  ret i32 %13
}
```

Explicit memory allocations removed for local variables and function arguements. Two call statements replaced by 1 using phi nodes determining the entry point for this node of the CFG.

e. fibo_iter O0

```llvm
define i64 @fibo_iter(i32) #0 {                  ;func fibo_iter with arg n and return int64
  %2 = alloca i64, align 8                       ;allocating space for arg. and local variables %2=value to be returned
  %3 = alloca i32, align 4                       ;for n
  %4 = alloca i64, align 8                       ;for fibo_cur
  %5 = alloca i64, align 8                       ;for fibo_prev
  %6 = alloca i32, align 4                       ;for i
  %7 = alloca i32, align 4                       ;for tmp
  store i32 %0, i32* %3, align 4                 ;%3=n store n
  %8 = load i32, i32* %3, align 4                ;%8=n
  %9 = icmp ult i32 %8, 3                        ;%9=(n<3)
  br i1 %9, label %10, label %11                 ;if true then jump to return 1 else label 11

; <label>:10:                                     ; preds = %1
  store i64 1, i64* %2, align 8                  ;%2=1 value to be returned
  br label %29

; <label>:11:                                     ; preds = %1
  store i64 1, i64* %4, align 8                  ;fibo_cur=1
  store i64 1, i64* %5, align 8                  ;fibo_prev=1
  store i32 3, i32* %6, align 4                  ;i=3
  br label %12

; <label>:12:                                     ; preds = %24, %11
  %13 = load i32, i32* %6, align 4               ;%13=i
  %14 = load i32, i32* %3, align 4               ;%14=n
  %15 = icmp ule i32 %13, %14                    ;%15=(i<=n)
  br i1 %15, label %16, label %27                ;if true to loop else return from function

; <label>:16:                                     ; preds = %12
  %17 = load i64, i64* %4, align 8               ;%17=fibo_cur
  %18 = trunc i64 %17 to i32                     ;%18=(int32)fibo_cur
  store i32 %18, i32* %7, align 4                ;store fibo_cur at %7 tmp=%18
  %19 = load i64, i64* %5, align 8               ;%19=fibo_prev
  %20 = load i64, i64* %4, align 8               ;%20=fibo_cur
  %21 = add i64 %20, %19                         ;%21=fibo_cur+fibo_prev
  store i64 %21, i64* %4, align 8                ;%4=%21, fibo_cur=%21
  %22 = load i32, i32* %7, align 4               ;%22=tmp
  %23 = zext i32 %22 to i64                      ;%23=(int64)tmp
  store i64 %23, i64* %5, align 8                ;%5=tmp, fibo_prev=tmp
  br label %24

; <label>:24:                                     ; preds = %16
  %25 = load i32, i32* %6, align 4               ;%25=i
  %26 = add i32 %25, 1                           ;%26=i+1
```

LLVM

```llvm
; <label>:24:                                     ; preds = %16
  %25 = load i32, i32* %6, align 4               ;%25=i
  %26 = add i32 %25, 1                           ;%26=i+1
  store i32 %26, i32* %6, align 4                ;storing incremented i
  br label %12                                   ;jump to start of loop

; <label>:27:                                     ; preds = %12
  %28 = load i64, i64* %4, align 8               ;%28=fibo_cur
  store i64 %28, i64* %2, align 8                ;storing fibo_cur at %2
  br label %29

; <label>:29:                                     ; preds = %27, %10
  %30 = load i64, i64* %2, align 8               ;%30=return value at %2
  ret i64 %30
}
```

f. fibo_iter O2

```
; Function Attrs: norecurse nounwind readnone uwtable
define i64 @fibo_iter(i32) local_unnamed_addr #0 {
  %2 = icmp ult i32 %0, 3                         ;%2=(n<3)
  br i1 %2, label %12, label %3                   ;if true then return with 1 else loop

; <label>:3:                                      ; preds = %1
  br label %4

; <label>:4:                                      ; preds = %3, %4
  %5 = phi i32 [ %10, %4 ], [ 3, %3 ]             ;if pred=4 then i+1 else 3
  %6 = phi i64 [ %9, %4 ], [ 1, %3 ]              ;if pred=4 then %6=fibo_cur else 1
  %7 = phi i64 [ %8, %4 ], [ 1, %3 ]              ;if pred=4 then %7=fibo_prev else 1
  %8 = add i64 %6, %7                             ;%8=fibo_cur+fibo_prev
  %9 = and i64 %7, 4294967295
  %10 = add i32 %5, 1                             ;%10=i+1
  %11 = icmp ugt i32 %10, %0                      ;%11=(i+1>n)
  br i1 %11, label %12, label %4                  ;if true then return from func, else loop

; <label>:12:                                     ; preds = %4, %1
  %13 = phi i64 [ 1, %1 ], [ %8, %4 ]             ;%13=value to be reurned
  ret i64 %13
}
```

Explicit memory allocations removed for local variables and function arguements. Loop operations simplified using phi node instruction which determines the live variables and thus specufying the values of fibo_cur and fibo_prev to be used.

g. gcd O0

```
; Function Attrs: noinline nounwind optnone uwtable
define i32 @gcd1(i32, i32) #0 {                          ;func. gcd1 with arg a and b
  %3 = alloca i32, align 4                               ;allocating space for arg and local variables
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  store i32 %0, i32* %4, align 4                         ;%4=a
  store i32 %1, i32* %5, align 4                         ;%5=b
  %6 = load i32, i32* %5, align 4                        ;%6=b
  %7 = icmp ne i32 %6, 0                                 ;%7=(b1!=0)
  br i1 %7, label %10, label %8                          ;if true then label 10 else 8

; <label>:8:                                             ; preds = %2
  %9 = load i32, i32* %4, align 4                        ;%9=a
  store i32 %9, i32* %3, align 4                         ;%3=a
  br label %16                                           ;jump to label 16 to return a

; <label>:10:                                            ; preds = %2
  %11 = load i32, i32* %5, align 4                       ;%11=b
  %12 = load i32, i32* %4, align 4                       ;%12=a
  %13 = load i32, i32* %5, align 4                       ;%13=b
  %14 = srem i32 %12, %13                                ;%14=a%b
  %15 = call i32 @gcd1(i32 %11, i32 %14)                 ;call gcd1(b,a%b)
  store i32 %15, i32* %3, align 4                        ;storing the result in %3
  br label %16

; <label>:16:                                            ; preds = %10, %8
  %17 = load i32, i32* %3, align 4                       ;%17=%3 value to be returned
  ret i32 %17
}
```

```
; Function Attrs: noinline nounwind optnone uwtable
define i32 @gcd2(i32, i32) #0 {                                    ;func gcd2 with arg a and b
  %3 = alloca i32, align 4                                         ;allocating space for local variable and arg.
  %4 = alloca i32, align 4
  store i32 %0, i32* %3, align 4                                   ;%3=a
  store i32 %1, i32* %4, align 4                                   ;%4=b
  br label %5

; <label>:5:                                       ; preds = %21, %2
  %6 = load i32, i32* %3, align 4                                  ;%6=a
  %7 = load i32, i32* %4, align 4                                  ;%7=b
  %8 = icmp ne i32 %6, %7                                          ;%8=(a!=b)
  br i1 %8, label %9, label %22                                    ;if true then label 9(loop) else 22(to return)

; <label>:9:                                       ; preds = %5
  %10 = load i32, i32* %3, align 4                                 ;%10=a
  %11 = load i32, i32* %4, align 4                                 ;%11=b
  %12 = icmp sgt i32 %10, %11                                      ;%12=(a>b)
  br i1 %12, label %13, label %17                                  ;if true then jump to 13 else 17

; <label>:13:                                      ; preds = %9
  %14 = load i32, i32* %4, align 4                                 ;%14=b
  %15 = load i32, i32* %3, align 4                                 ;%15=a
  %16 = sub nsw i32 %15, %14                                       ;%16=a-b
  store i32 %16, i32* %3, align 4                                  ;%3=(a-b),a value to be returned
  br label %21

; <label>:17:                                      ; preds = %9
  %18 = load i32, i32* %3, align 4                                 ;%18=a
  %19 = load i32, i32* %4, align 4                                 ;%19=b
  %20 = sub nsw i32 %19, %18                                       ;%20=(b-a)
  store i32 %20, i32* %4, align 4                                  ;%4=(b-a), b=b-a
  br label %21

; <label>:21:                                      ; preds = %17, %13
  br label %5

; <label>:22:                                      ; preds = %5
  %23 = load i32, i32* %3, align 4                                 ;%23=%3 a,value to be returned
  ret i32 %23
}
```

```
}

; Function Attrs: noinline nounwind optnone uwtable
define i32 @gcd3(i32, i32) #0 {                                    ;func gcd3 with arg a and b
  %3 = alloca i32, align 4                                         ;allocating space for local var. and arg.
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  store i32 %0, i32* %3, align 4                                   ;%3=a
  store i32 %1, i32* %4, align 4                                   ;%4=b
  br label %6

; <label>:6:                                       ; preds = %9, %2
  %7 = load i32, i32* %4, align 4                                  ;%7=b
  %8 = icmp ne i32 %7, 0                                           ;%8=(b!=0)
  br i1 %8, label %9, label %15                                    ;if true then loop else return a

; <label>:9:                                       ; preds = %6
  %10 = load i32, i32* %4, align 4                                 ;%10=b
  store i32 %10, i32* %5, align 4                                  ;%5=b
  %11 = load i32, i32* %3, align 4                                 ;%11=a
  %12 = load i32, i32* %4, align 4                                 ;%12=b
  %13 = srem i32 %11, %12                                          ;13=a%b
  store i32 %13, i32* %4, align 4                                  ;%4=(a%b)
  %14 = load i32, i32* %5, align 4                                 ;%14=temp
  store i32 %14, i32* %3, align 4                                  ;%14(temp)=a
  br label %6                                                      ;jump to loop

; <label>:15:                                      ; preds = %6
  %16 = load i32, i32* %3, align 4                                 ;%16=a
  ret i32 %16                                                      ;return a
}
```

h. gcd O2

```llvm
; Function Attrs: nounwind readnone uwtable
define i32 @gcd1(i32, i32) local_unnamed_addr #0 {              ;func gcd1 with arg a and b
  %3 = icmp eq i32 %1, 0                                        ;%3=(b=0)
  br i1 %3, label %10, label %4                                 ;if true then label 10 else label 4

; <label>:4:                                       ; preds = %2
  br label %5

; <label>:5:                                       ; preds = %4, %5
  %6 = phi i32 [ %8, %5 ], [ %1, %4 ]                           ;if pred=5 then %6=(a%b) else b
  %7 = phi i32 [ %6, %5 ], [ %0, %4 ]                           ;if pred=5 then %7=%6 else a
  %8 = srem i32 %7, %6                                          ;%8=a%b
  %9 = icmp eq i32 %8, 0                                        ;%9=(a%b=0)
  br i1 %9, label %10, label %5

; <label>:10:                                      ; preds = %5, %2
  %11 = phi i32 [ %0, %2 ], [ %6, %5 ]
  ret i32 %11                                                   ;return a
}

; Function Attrs: norecurse nounwind readnone uwtable
define i32 @gcd2(i32, i32) local_unnamed_addr #1 {              ;func gcd2 with arg. a and b
  %3 = icmp eq i32 %0, %1                                       ;%3=(a=b)
  br i1 %3, label %14, label %4                                 ;if true then jump to label 14 else 4(loop)

; <label>:4:                                       ; preds = %2
  br label %5

; <label>:5:                                       ; preds = %4, %5
  %6 = phi i32 [ %12, %5 ], [ %1, %4 ]                          ;if pred=5 then %6=(b-a) else b
  %7 = phi i32 [ %10, %5 ], [ %0, %4 ]                          ;if pred=5 then a-b else a
  %8 = icmp slt i32 %6, %7                                      ;%8=(a>b)
  %9 = select i1 %8, i32 %6, i32 0                              ;if true then (b-a) else 0
  %10 = sub nsw i32 %7, %9                                      ;%10=(a-b) a=a-b
  %11 = select i1 %8, i32 0, i32 %7                             ;if true(%8) then 0 else a-b
  %12 = sub nsw i32 %6, %11                                     ;%12=(b-a) b=b-a
  %13 = icmp eq i32 %10, %12                                    ;%13=(a=b)
  br i1 %13, label %14, label %5

; <label>:14:                                      ; preds = %5, %2
  %15 = phi i32 [ %0, %2 ], [ %10, %5 ]
  ret i32 %15
}
```

LLVM IR

```
; Function Attrs: norecurse nounwind readnone uwtable
define i32 @gcd3(i32, i32) local_unnamed_addr #1 {                ;func gcd3 with arg a and b
  %3 = icmp eq i32 %1, 0                                          ;%3=(b=0)
  br i1 %3, label %10, label %4                                   ;if true then return else label 4(loop)

; <label>:4:                                                     ; preds = %2
  br label %5

; <label>:5:                                                     ; preds = %4, %5
  %6 = phi i32 [ %7, %5 ], [ %0, %4 ]                             ;if pred=5 then %7 else a
  %7 = phi i32 [ %8, %5 ], [ %1, %4 ]                             ;if pred=5 then a%b else b
  %8 = srem i32 %6, %7                                            ;%8=a%b
  %9 = icmp eq i32 %8, 0                                          ;%9=(a%b=0)
  br i1 %9, label %10, label %5                                   ;if true then label 10 else loop

; <label>:10:                                                    ; preds = %5, %2
  %11 = phi i32 [ %0, %2 ], [ %7, %5 ]                            ;%11=value to be returned
  ret i32 %11                                                     ;return %11(a)
}
```

Explicit memory allocations removed for local variables and function arguements. Unnecessary moves and stores removed. Loops simplified using phi nodes.

i. print_args O0

```
define i32 @print_arg(i32, i8**) #0 {                                   ;func print_arg with arg. argc and argv
  %3 = alloca i32, align 4                                              ;allocating space for arg and local variables
  %4 = alloca i32, align 4
  %5 = alloca i8**, align 8
  store i32 %0, i32* %4, align 4                                        ;%4=argc
  store i8** %1, i8*** %5, align 8                                      ;%5=argv
  %6 = load i32, i32* %4, align 4                                       ;%6=argc
  %7 = icmp ne i32 %6, 2                                                ;%7=(argc!=2)
  br i1 %7, label %8, label %9                                          ;if true then jump to label 8 else 9

; <label>:8:                                       ; preds = %2
  store i32 -1, i32* %3, align 4                                        ;%3=-1, value to be returned
  br label %14                                                          ;jump to label 14

; <label>:9:                                       ; preds = %2
  %10 = load i8**, i8*** %5, align 8                                    ;%10=argv
  %11 = getelementptr inbounds i8*, i8** %10, i64 1
  %12 = load i8*, i8** %11, align 8                                     ;%12=argv[1]
  %13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i32 0, i32 0), i8* %12)    ;%13=printf(argv[1])
  store i32 0, i32* %3, align 4                                         ;%3=0, value to be returned
  br label %14

; <label>:14:                                      ; preds = %9, %8
  %15 = load i32, i32* %3, align 4                                      ;%15=%3
  ret i32 %15                                                           ;return 0 or -1
}

declare i32 @printf(i8*, ...) #1                                        ;declaring printf function
```

j. print_args O2

```
define i32 @print_arg(i32, i8** nocapture readonly) local_unnamed_addr #0 {
  %3 = icmp eq i32 %0, 2                                                                ;%3=(argc=2)
  br i1 %3, label %4, label %8                                                          ;if equal then label 4 else label 8

; <label>:4:                                       ; preds = %2
  %5 = getelementptr inbounds i8*, i8** %1, i64 1                                       ;
  %6 = load i8*, i8** %5, align 8, !tbaa !2                                             ;%6=argv[1]
  %7 = tail call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i64 0, i64 0), i8* %6)      ;call printf(arg[v])
  br label %8

; <label>:8:                                       ; preds = %2, %4
  %9 = phi i32 [ 0, %4 ], [ -1, %2 ]                                                   ;if pred=4 then %9=0 else -1
  ret i32 %9                                                                            ;return 0 or -1
}

; Function Attrs: nounwind
declare i32 @printf(i8* nocapture readonly, ...) local_unnamed_addr #1                  ;declaring printf function
```

Explicit memory allocations removed for local variables and function arguements. Use of phi node to determine what value to return instead of storing it in a variable and thus removed the instructions not required.

k. loops O0

```
define zeroext i1 @is_sorted(i32*, i32) #0 {                    ;func is_sorted with 2 arg. n and *a
  %3 = alloca i1, align 1                                       ;allocating space for arg. and local variables
  %4 = alloca i32*, align 8
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store i32* %0, i32** %4, align 8                              ;%4=&a
  store i32 %1, i32* %5, align 4                                ;%5=n
  store i32 0, i32* %6, align 4                                 ;%6=0 (i=0)
  br label %7                                                   ;jump to label 7

; <label>:7:                                      ; preds = %27, %2
  %8 = load i32, i32* %6, align 4                               ;%8=i
  %9 = load i32, i32* %5, align 4                               ;%9=n
  %10 = sub nsw i32 %9, 1                                       ;%10=n-1
  %11 = icmp slt i32 %8, %10                                    ;compare i and n-1
  br i1 %11, label %12, label %30                               ;if (i<n-1) then label 12 else 30

; <label>:12:                                     ; preds = %7
  %13 = load i32*, i32** %4, align 8                            ;%13=&a
  %14 = load i32, i32* %6, align 4                              ;%14=i
  %15 = sext i32 %14 to i64                                     ;%15=(int64)i
  %16 = getelementptr inbounds i32, i32* %13, i64 %15
  %17 = load i32, i32* %16, align 4                             ;%17=a[i]
  %18 = load i32*, i32** %4, align 8                            ;%17=&a
  %19 = load i32, i32* %6, align 4                              ;%18=i
  %20 = add nsw i32 %19, 1                                      ;%19=i+1
  %21 = sext i32 %20 to i64                                     ;%20=(int64)i+1
  %22 = getelementptr inbounds i32, i32* %18, i64 %21           ;
  %23 = load i32, i32* %22, align 4                             ;%23=a[i+1]
  %24 = icmp sgt i32 %17, %23                                   ;compare a[i] and a[i+1]
  br i1 %24, label %25, label %26                               ;if(a[i]>a[i+1]) then label 25(to return false) else 26

; <label>:25:                                     ; preds = %12
  store i1 false, i1* %3, align 1                                       ;%3=false (value to be returned)
  br label %31

; <label>:26:                                     ; preds = %12
  br label %27
```

```
; <label>:26:                                       ; preds = %12
  br label %27

; <label>:27:                                       ; preds = %26
  %28 = load i32, i32* %6, align 4                                      ;%28=i
  %29 = add nsw i32 %28, 1                                              ;%29=i+1
  store i32 %29, i32* %6, align 4                                       ;%6=i+1,i=i+1
  br label %7                                                           ;jump to start of loop

; <label>:30:                                       ; preds = %7
  store i1 true, i1* %3, align 1
  br label %31

; <label>:31:                                       ; preds = %30, %25
  %32 = load i1, i1* %3, align 1                                        ;%32=value to be returned
  ret i1 %32
}

; Function Attrs: noinline nounwind optnone uwtable
```

```
define void @add_arrays(i32*, i32*, i32*, i32) #0 {            ;func add_arrays with 4 arg.
  %5 = alloca i32*, align 8                                    ;allocating space for local var and arg.
  %6 = alloca i32*, align 8
  %7 = alloca i32*, align 8
  %8 = alloca i32, align 4
  %9 = alloca i32, align 4
  store i32* %0, i32** %5, align 8                             ;%5=&a
  store i32* %1, i32** %6, align 8                             ;%6=&b
  store i32* %2, i32** %7, align 8                             ;%7=&c
  store i32 %3, i32* %8, align 4                               ;%8=n
  store i32 0, i32* %9, align 4                                ;%9=0, i=0
  br label %10

; <label>:10:                                      ; preds = %30, %4
  %11 = load i32, i32* %9, align 4                             ;%11=i
  %12 = load i32, i32* %8, align 4                             ;%12=n
  %13 = icmp slt i32 %11, %12                                  ;%13=(i<n)
  br i1 %13, label %14, label %33                              ;if true then label 14(loop)

; <label>:14:                                      ; preds = %10
  %15 = load i32*, i32** %5, align 8                           ;%15=&a
  %16 = load i32, i32* %9, align 4                             ;%16=i
  %17 = sext i32 %16 to i64                                    ;%17=(int64)i
  %18 = getelementptr inbounds i32, i32* %15, i64 %17
  %19 = load i32, i32* %18, align 4                            ;%19=a[i]
  %20 = load i32*, i32** %6, align 8                           ;%20=&b
  %21 = load i32, i32* %9, align 4
  %22 = sext i32 %21 to i64
  %23 = getelementptr inbounds i32, i32* %20, i64 %22
  %24 = load i32, i32* %23, align 4                            ;%24=b[i]
  %25 = add nsw i32 %19, %24                                   ;%25=a[i]+b[i]
  %26 = load i32*, i32** %7, align 8                           ;%26=&c
  %27 = load i32, i32* %9, align 4
  %28 = sext i32 %27 to i64
  %29 = getelementptr inbounds i32, i32* %26, i64 %28          ;%29=c[i]
  store i32 %25, i32* %29, align 4                             ;c[i]=a[i]+b[i]
  br label %30

; <label>:30:                                      ; preds = %14
  %31 = load i32, i32* %9, align 4                                  ;%31=i
  %32 = add nsw i32 %31, 1                                          ;%32=i+1
  store i32 %32, i32* %9, align 4                                   ;i=i+1
  br label %10                                                      ;jupm to start of loop

; <label>:33:                                      ; preds = %10
  ret void                                                          ;return void
}

; Function Attrs: noinline nounwind optnone uwtable
```

```
define i32 @sum(i8*, i32) #0 {                                   ;func sum with 2 arg.
  %3 = alloca i8*, align 8                                        ;allocating space for arg. and local variables
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store i8* %0, i8** %3, align 8                                  ;%3=&a
  store i32 %1, i32* %4, align 4                                  ;%4=n
  store i32 0, i32* %5, align 4                                   ;%5=0, ret=0
  store i32 0, i32* %6, align 4                                   ;%6=0, i=0
  br label %7                                                     ;jump to label 7

; <label>:7:                                      ; preds = %20, %2
  %8 = load i32, i32* %6, align 4                                 ;%8=i
  %9 = load i32, i32* %4, align 4                                 ;%9=n
  %10 = icmp slt i32 %8, %9                                       ;%10=(i<n)
  br i1 %10, label %11, label %23                                 ;if true then label 11(loop) else 23

; <label>:11:                                     ; preds = %7
  %12 = load i8*, i8** %3, align 8                                ;%12=&a
  %13 = load i32, i32* %6, align 4                                ;%13=i
  %14 = sext i32 %13 to i64                                       ;%14=(int64)i
  %15 = getelementptr inbounds i8, i8* %12, i64 %14
  %16 = load i8, i8* %15, align 1                                 ;%16=a[i]
  %17 = zext i8 %16 to i32                                        ;%17=(int32)a[i]
  %18 = load i32, i32* %5, align 4                                ;&18=ret
  %19 = add nsw i32 %18, %17                                      ;%19=(int32)a[i]+ret
  store i32 %19, i32* %5, align 4                                 ;ret=ret+a[i]
  br label %20

; <label>:20:                                     ; preds = %11
  %21 = load i32, i32* %6, align 4                                ;%21=i
  %22 = add nsw i32 %21, 1                                        ;%22=i+1
  store i32 %22, i32* %6, align 4                                 ;i=i+1
  br label %7                                                     ;jump to start of loop

; <label>:23:                                     ; preds = %7
  %24 = load i32, i32* %5, align 4                                ;%24=ret, value to be returned
  ret i32 %24
}
```

```
define i32 @sumn(i32) #0 {                                  ;func sumn with arg n
  %2 = alloca i32, align 4                                  ;allocating space for arg. and local variables
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  store i32 %0, i32* %2, align 4                            ;%2=n
  store i32 0, i32* %3, align 4                             ;%3=0, ret=0
  store i32 0, i32* %4, align 4                             ;%4=0, i=0
  br label %5

; <label>:5:                                       ; preds = %13, %1
  %6 = load i32, i32* %4, align 4                           ;%6=i
  %7 = load i32, i32* %2, align 4                           ;%7=n
  %8 = icmp slt i32 %6, %7                                  ;compare i and n
  br i1 %8, label %9, label %16                             ;if(i<n) then loop

; <label>:9:                                       ; preds = %5
  %10 = load i32, i32* %3, align 4                          ;%10=ret
  %11 = load i32, i32* %4, align 4                          ;%11=i
  %12 = add nsw i32 %10, %11                                ;%12=ret+i
  store i32 %12, i32* %3, align 4                           ;ret=ret+i
  br label %13

; <label>:13:                                      ; preds = %9
  %14 = load i32, i32* %4, align 4                          ;%14=i
  %15 = add nsw i32 %14, 1                                  ;%15=i+1
  store i32 %15, i32* %4, align 4                           ;i+i+1
  br label %5                                               ;jump to start of loop

; <label>:16:                                      ; preds = %5
  %17 = load i32, i32* %3, align 4                          ;%17=%3 value to be returned ret
  ret i32 %17
}
```

l. loops O2

```
define zeroext i1 @is_sorted(i32* nocapture readonly, i32) local_unnamed_addr #0 {              ;func is_sorted with 2 arg.
  %3 = icmp sgt i32 %1, 1                                   ;%3=(n>1)
  br i1 %3, label %4, label %17                             ;if true then loop else return from func.

; <label>:4:                                       ; preds = %2
  %5 = add nsw i32 %1, -1                                   ;%4=n-1
  %6 = sext i32 %5 to i64                                   ;%5=(int64)n-1
  %7 = load i32, i32* %0, align 4, !tbaa !2                 ;%7=&a
  br label %10

; <label>:8:                                       ; preds = %10
  %9 = icmp slt i64 %13, %6                                 ;%9=(i+1<(n-1))
  br i1 %9, label %10, label %17                            ;if true then loop else to return

; <label>:10:                                      ; preds = %4, %8
  %11 = phi i32 [ %7, %4 ], [ %15, %8 ]
  %12 = phi i64 [ 0, %4 ], [ %13, %8 ]
  %13 = add nuw nsw i64 %12, 1                              ;%13=i+1
  %14 = getelementptr inbounds i32, i32* %0, i64 %13
  %15 = load i32, i32* %14, align 4, !tbaa !2               ;%15=a[i]
  %16 = icmp sgt i32 %11, %15                               ;%16=(a[i]>a[i+1])
  br i1 %16, label %17, label %8                            ;if true then %17 to return false else loop

; <label>:17:                                                          ; preds = %10, %8, %2
  %18 = phi i1 [ true, %2 ], [ true, %8 ], [ false, %10 ]               ;if pred=2,8 then true else false
  ret i1 %18
}
```

```
; Function Attrs: norecurse nounwind readnone uwtable
define i32 @sumn(i32) local_unnamed_addr #2 {                    ;func sumn with arg. n
  %2 = icmp sgt i32 %0, 0                                        ;%2=(n>0)
  br i1 %2, label %3, label %13                                  ;if true then loop else return from func.

; <label>:3:                                       ; preds = %1
  %4 = add i32 %0, -1                                            ;%4=n-1
  %5 = zext i32 %4 to i33                                        ;%5=(int33)n-1
  %6 = add i32 %0, -2                                            ;%6=n-2
  %7 = zext i32 %6 to i33                                        ;zero extend n-2
  %8 = mul i33 %5, %7                                            ;%8=(n-1)(n-2)
  %9 = lshr i33 %8, 1                                            ;%9=(n-1)(n-2)/2
  %10 = trunc i33 %9 to i32                                      ;%10=(int32)%9
  %11 = add i32 %10, %0                                          ;%11=(n-1)(n-2)/2+n
  %12 = add i32 %11, -1                                          ;%12=n2-n-1
  br label %13

; <label>:13:                                      ; preds = %3, %1
  %14 = phi i32 [ 0, %1 ], [ %12, %3 ]                           ;if pred=1 then return 0 else %12(sum)
  ret i32 %14
}
```

The generated code for level O2 for sum and add_arrays has more instructions than level O0 as sometimes, the higher optimizations add no reasonable benefit but a lot of extra size.

2. x86 ASSEMBLY (GCC)

a. emptyloop O0

```
emptyloop.i386.O0.o:     file format elf32-i386


Disassembly of section .text:

00000000 <emptyloop>:
   0:   55                      push   %ebp                    #push ebp register content
   1:   89 e5                   mov    %esp,%ebp               #ebp=esp
   3:   56                      push   %esi                    #pusdh esi register
   4:   83 ec 44                sub    $0x44,%esp              #esp=esp-0x44 space allocated on stack
   7:   8b 45 0c                mov    0xc(%ebp),%eax          #eax=ebp+12, argv
   a:   8b 4d 08                mov    0x8(%ebp),%ecx          #ecx=ebp+8, argc
   d:   89 45 f8                mov    %eax,-0x8(%ebp)         #ebp-8=eax,argc
  10:   c7 45 ec 00 00 00 00    movl   $0x0,-0x14(%ebp)        #ebp-0x14=0 ,i
  17:   c7 45 e8 fe ff ff 7f    movl   $0x7ffffffe,-0x18(%ebp) #ebp-0x18=int_max-1,numiter
  1e:   83 7d 08 02             cmpl   $0x2,0x8(%ebp)          #comparing 2 with argc
  22:   89 4d e4                mov    %ecx,-0x1c(%ebp)        #storing argc at stack
  25:   7c 1a                   jl     41 <emptyloop+0x41>     #if argc<2 then jump to loc.41 else fallthrough
  27:   8b 45 f8                mov    -0x8(%ebp),%eax         #eax=argc
  2a:   8b 40 04                mov    0x4(%eax),%eax          #eax=eax+4
  2d:   89 e1                   mov    %esp,%ecx               #ecx=esp
  2f:   89 01                   mov    %eax,(%ecx)             #ecx=eax
  31:   e8 fc ff ff ff          call   32 <emptyloop+0x32>     #call 32, the next arguement
  36:   89 c1                   mov    %eax,%ecx               #ecx=eax
  38:   c1 f9 1f                sar    $0x1f,%ecx              #right arithmetic shift of 1f
  3b:   89 45 e8                mov    %eax,-0x18(%ebp)
  3e:   89 4d ec                mov    %ecx,-0x14(%ebp)
  41:   c7 45 f4 00 00 00 00    movl   $0x0,-0xc(%ebp)
  48:   c7 45 f0 00 00 00 00    movl   $0x0,-0x10(%ebp)
  4f:   8b 45 f0                mov    -0x10(%ebp),%eax
  52:   8b 4d f4                mov    -0xc(%ebp),%ecx
  55:   8b 55 e8                mov    -0x18(%ebp),%edx
  58:   8b 75 ec                mov    -0x14(%ebp),%esi
  5b:   81 ea 22 43 05 00       sub    $0x54322,%edx
  61:   83 de 00                sbb    $0x0,%esi
  64:   89 45 e0                mov    %eax,-0x20(%ebp)
  67:   89 4d dc                mov    %ecx,-0x24(%ebp)
  6a:   89 55 d8                mov    %edx,-0x28(%ebp)
  6d:   89 75 d4                mov    %esi,-0x2c(%ebp)
  70:   72 10                   jb     82 <emptyloop+0x82>
  72:   eb 00                   jmp    74 <emptyloop+0x74>
  74:   8b 45 e8                mov    -0x18(%ebp),%eax
  77:   8b 4d ec                mov    -0x14(%ebp),%ecx
```

```
52:   8b 4d f4               mov    -0xc(%ebp),%ecx
55:   8b 55 e8               mov    -0x18(%ebp),%edx
58:   8b 75 ec               mov    -0x14(%ebp),%esi
5b:   81 ea 22 43 05 00      sub    $0x54322,%edx
61:   83 de 00               sbb    $0x0,%esi
64:   89 45 e0               mov    %eax,-0x20(%ebp)
67:   89 4d dc               mov    %ecx,-0x24(%ebp)
6a:   89 55 d8               mov    %edx,-0x28(%ebp)
6d:   89 75 d4               mov    %esi,-0x2c(%ebp)
70:   72 10                  jb     82 <emptyloop+0x82>
72:   eb 00                  jmp    74 <emptyloop+0x74>
74:   8b 45 e8               mov    -0x18(%ebp),%eax
77:   8b 4d ec               mov    -0x14(%ebp),%ecx
7a:   89 45 d0               mov    %eax,-0x30(%ebp)
7d:   89 4d cc               mov    %ecx,-0x34(%ebp)
80:   eb 0f                  jmp    91 <emptyloop+0x91>
82:   31 c0                  xor    %eax,%eax              #clearing eax
84:   b9 21 43 05 00         mov    $0x54321,%ecx          #ecx=magic number
89:   89 4d d0               mov    %ecx,-0x30(%ebp)       #storing ecx and eax in stack
8c:   89 45 cc               mov    %eax,-0x34(%ebp)
8f:   eb 00                  jmp    91 <emptyloop+0x91>    #jump to next instruction
91:   8b 45 cc               mov    -0x34(%ebp),%eax       #eax=
94:   8b 4d d0               mov    -0x30(%ebp),%ecx       #ecx=numiter
97:   8b 55 e0               mov    -0x20(%ebp),%edx       #edx=magic number
9a:   29 ca                  sub    %ecx,%edx
9c:   8b 4d dc               mov    -0x24(%ebp),%ecx
9f:   19 c1                  sbb    %eax,%ecx              #comparing numiter and magic number
a1:   89 4d c8               mov    %ecx,-0x38(%ebp)       #storing ecx and edx contents
a4:   89 55 c4               mov    %edx,-0x3c(%ebp)
a7:   73 18                  jae    c1 <emptyloop+0xc1>    #jump if above or equal to c1(result of sbb)=terminate the loop
a9:   eb 00                  jmp    ab <emptyloop+0xab>
ab:   eb 00                  jmp    ad <emptyloop+0xad>
ad:   8b 45 f0               mov    -0x10(%ebp),%eax       #eax=i
b0:   8b 4d f4               mov    -0xc(%ebp),%ecx        #ecx=ebp-12
b3:   83 c0 01               add    $0x1,%eax              #i incremented by 1
b6:   83 d1 00               adc    $0x0,%ecx              #ecx=ecx+0 with carry
b9:   89 45 f0               mov    %eax,-0x10(%ebp)       #ebp-16=eax
bc:   89 4d f4               mov    %ecx,-0xc(%ebp)        #ebp-12=ecx
bf:   eb 8e                  jmp    4f <emptyloop+0x4f>    #jump to loop
c1:   31 c0                  xor    %eax,%eax              #clearing eax
c3:   83 c4 44               add    $0x44,%esp             #restoring stack
c6:   5e                     pop    %esi
c7:   5d                     pop    %ebp
c8:   c3                     ret
```

Plain Text Tab

b. emptyloop O2

```
emptyloop.i386.O2.o:     file format elf32-i386


Disassembly of section .text:

00000000 <emptyloop>:
   0:   83 ec 0c                sub    $0xc,%esp               #12 bytes onstack
   3:   83 7c 24 10 02          cmpl   $0x2,0x10(%esp)         #comparing 2 with esp+16 argc
   8:   7c 16                   jl     20 <emptyloop+0x20>     #if argc<2 then jump to loc.20 to return
   a:   8b 44 24 14             mov    0x14(%esp),%eax         #eax=esp+ox14, argv
   e:   83 ec 04                sub    $0x4,%esp               #esp=esp-4
  11:   6a 0a                   push   $0xa                    #push 10
  13:   6a 00                   push   $0x0                    #push 0  value to be returned
  15:   ff 70 04                pushl  0x4(%eax)               #push eax+4
  18:   e8 fc ff ff ff          call   19 <emptyloop+0x19>     #call next arguement
  1d:   83 c4 10                add    $0x10,%esp              #esp=esp+16
  20:   31 c0                   xor    %eax,%eax               #clearing eax
  22:   83 c4 0c                add    $0xc,%esp               #restoring stack
  25:   c3                      ret
```

All instructions related to the loop are removed. Explicit memory allocations removed for local variables and function arguements.

c. fib O0

```
fib.i386.O0.o:     file format elf32-i386


Disassembly of section .text:

00000000 <fib>:
   0:   55                      push   %ebp                    #push ebp, ebp=esp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 ec 18                sub    $0x18,%esp              #esp=esp-ox18, space allocated on stack
   6:   8b 45 08                mov    0x8(%ebp),%eax          #eax= ebp+8, arguement n
   9:   83 7d 08 02             cmpl   $0x2,0x8(%ebp)          #comparing arg n with 2
   d:   89 45 f8                mov    %eax,-0x8(%ebp)         #storing n in stack
  10:   7d 09                   jge    1b <fib+0x1b>           #if 2<n then jump to loc 1b
  12:   c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%ebp)         #storing 1 on stack
  19:   eb 27                   jmp    42 <fib+0x42>           #jump to loc 42 to return
  1b:   8b 45 08                mov    0x8(%ebp),%eax          #eax=n
  1e:   83 e8 01                sub    $0x1,%eax               #eax=n-1
  21:   89 04 24                mov    %eax,(%esp)             #esp=eax
  24:   e8 fc ff ff ff          call   25 <fib+0x25>           #call fib with arg n-1
  29:   8b 4d 08                mov    0x8(%ebp),%ecx          #ecx=arg n
  2c:   83 e9 02                sub    $0x2,%ecx               #ecx=n-2
  2f:   89 0c 24                mov    %ecx,(%esp)             #esp=ecx (n)
  32:   89 45 f4                mov    %eax,-0xc(%ebp)         #storing eax on stack
  35:   e8 fc ff ff ff          call   36 <fib+0x36>           #call fib with n-2
  3a:   8b 4d f4                mov    -0xc(%ebp),%ecx         #ecx=return value
  3d:   01 c1                   add    %eax,%ecx               #ecx=ecx+eax
  3f:   89 4d fc                mov    %ecx,-0x4(%ebp)         #ebp-4=ecx value to be returned
  42:   8b 45 fc                mov    -0x4(%ebp),%eax         #eax=value to be returned
  45:   83 c4 18                add    $0x18,%esp              #restoring stack
  48:   5d                      pop    %ebp
  49:   c3                      ret
```

d. fib O2

```
fib.i386.O2.o:     file format elf32-i386


Disassembly of section .text:

00000000 <fib>:
   0:   57                      push   %edi                    #push edi,eax,esi
   1:   56                      push   %esi
   2:   50                      push   %eax
   3:   8b 7c 24 10             mov    0x10(%esp),%edi         #edi=esp+16, n
   7:   be 01 00 00 00          mov    $0x1,%esi               #esi=1
   c:   83 ff 02                cmp    $0x2,%edi               #comparing 2 with arg n
   f:   7c 24                   jl     35 <fib+0x35>           #if n<2 then jump to loc 35 to return value in esi
  11:   83 c7 02                add    $0x2,%edi               #edi=edi+2
  14:   be 01 00 00 00          mov    $0x1,%esi               #esi=1
  19:   8d b4 26 00 00 00 00    lea    0x0(%esi,%eiz,1),%esi   #load effective address esi+eiz*1 in esi
  20:   8d 47 fd                lea    -0x3(%edi),%eax         #eax=edi-3, n-1
  23:   89 04 24                mov    %eax,(%esp)             #esp=eax, eax content on top of stack
  26:   e8 fc ff ff ff          call   27 <fib+0x27>           #call fib with arg n-1
  2b:   01 c6                   add    %eax,%esi               #esi=esi+eax, value to be returned
  2d:   83 c7 fe                add    $0xfffffffe,%edi        #edi=edi+int_max-1
  30:   83 ff 03                cmp    $0x3,%edi               #comparing 3 with edi
  33:   7f eb                   jg     20 <fib+0x20>           #if edi>3 then loop
  35:   89 f0                   mov    %esi,%eax               #eax=value to be returned
  37:   83 c4 04                add    $0x4,%esp               #restoring stack
  3a:   5e                      pop    %esi
  3b:   5f                      pop    %edi
  3c:   c3                      ret
```

Registers are being used for local variables instead of storing on stack. Two calls to fib replaced by one call.

e. fibo_iter O0

```
Disassembly of section .text:

00000000 <fibo_iter>:
   0:   55                      push   %ebp                    #push ebp
   1:   89 e5                   mov    %esp,%ebp               #ebp=esp
   3:   56                      push   %esi                    #push esi
   4:   83 ec 2c                sub    $0x2c,%esp              #2c bytes on stack
   7:   8b 45 08                mov    0x8(%ebp),%eax          #eax=n
   a:   83 7d 08 03             cmpl   $0x3,0x8(%ebp)          #comparing 3 with n
   e:   89 45 d4                mov    %eax,-0x2c(%ebp)        #storing n on stack
  11:   73 10                   jae    23 <fibo_iter+0x23>     #if 3>n then jump to loc 23
  13:   c7 45 f4 00 00 00 00    movl   $0x0,-0xc(%ebp)         #storing 0 and 1 on stack
  1a:   c7 45 f0 01 00 00 00    movl   $0x1,-0x10(%ebp)
  21:   eb 69                   jmp    8c <fibo_iter+0x8c>     #jump to loc 8c
  23:   c7 45 ec 00 00 00 00    movl   $0x0,-0x14(%ebp)        #moving 0 and 1 on stack
  2a:   c7 45 e8 01 00 00 00    movl   $0x1,-0x18(%ebp)        #fibo_cur
  31:   c7 45 e4 00 00 00 00    movl   $0x0,-0x1c(%ebp)
  38:   c7 45 e0 01 00 00 00    movl   $0x1,-0x20(%ebp)        #fibo_prev
  3f:   c7 45 dc 03 00 00 00    movl   $0x3,-0x24(%ebp)        #ebp-ox24=3  (i)
  46:   8b 45 dc                mov    -0x24(%ebp),%eax        #eax=i
  49:   3b 45 08                cmp    0x8(%ebp),%eax          #comparing i and n
  4c:   77 34                   ja     82 <fibo_iter+0x82>     #if i>n then jump to terminate the loop
  4e:   8b 45 e8                mov    -0x18(%ebp),%eax        #eax=1
  51:   89 45 d8                mov    %eax,-0x28(%ebp)        #storing eax on stack
  54:   8b 45 e0                mov    -0x20(%ebp),%eax        #eax=fibo_prev
  57:   8b 4d e4                mov    -0x1c(%ebp),%ecx
  5a:   8b 55 e8                mov    -0x18(%ebp),%edx        #edx=fibo_cur
  5d:   8b 75 ec                mov    -0x14(%ebp),%esi
  60:   01 c2                   add    %eax,%edx               #edx=edx+eax
  62:   11 ce                   adc    %ecx,%esi               #esi=esi+ecx
  64:   89 55 e8                mov    %edx,-0x18(%ebp)        #storing esi and edx on stack
  67:   89 75 ec                mov    %esi,-0x14(%ebp)
  6a:   8b 45 d8                mov    -0x28(%ebp),%eax
  6d:   89 45 e0                mov    %eax,-0x20(%ebp)
  70:   c7 45 e4 00 00 00 00    movl   $0x0,-0x1c(%ebp)
  77:   8b 45 dc                mov    -0x24(%ebp),%eax        #eax=i
  7a:   83 c0 01                add    $0x1,%eax               #incrementing i
  7d:   89 45 dc                mov    %eax,-0x24(%ebp)        #storing i
  80:   eb c4                   jmp    46 <fibo_iter+0x46>     #jump to loop start
  82:   f2 0f 10 45 e8          movsd  -0x18(%ebp),%xmm0
  87:   f2 0f 11 45 f0          movsd  %xmm0,-0x10(%ebp)
  8c:   8b 45 f0                mov    -0x10(%ebp),%eax
  8f:   8b 55 f4                mov    -0xc(%ebp),%edx
  92:   83 c4 2c                add    $0x2c,%esp              #restoring stack
```

f. fibo_iter O2

```
fibo_iter.i386.O2.o:     file format elf32-i386


Disassembly of section .text:

00000000 <fibo_iter>:
   0:   55                      push   %ebp                    #push registers
   1:   53                      push   %ebx
   2:   57                      push   %edi
   3:   56                      push   %esi
   4:   8b 4c 24 14             mov    0x14(%esp),%ecx         #ecx=n
   8:   83 f9 03                cmp    $0x3,%ecx               #comparing n with 3
   b:   73 09                   jae    16 <fibo_iter+0x16>     #if n>=3 then loop else return value will be in eax
   d:   31 d2                   xor    %edx,%edx               #clearing edx
   f:   b8 01 00 00 00          mov    $0x1,%eax               #eax=1, value to be returned if above condition fails
  14:   eb 2b                   jmp    41 <fibo_iter+0x41>     #jump to return
  16:   31 f6                   xor    %esi,%esi               #clearing edx
  18:   bd 01 00 00 00          mov    $0x1,%ebp               #ebp=1 fibo_cur
  1d:   bf 03 00 00 00          mov    $0x3,%edi               #edi=3 (i)
  22:   bb 01 00 00 00          mov    $0x1,%ebx               #ebx=1 fibo_prev
  27:   31 d2                   xor    %edx,%edx
  29:   8d b4 26 00 00 00 00    lea    0x0(%esi,%eiz,1),%esi   #load effective address
  30:   89 e8                   mov    %ebp,%eax               #eax=fibo_cur
  32:   01 d8                   add    %ebx,%eax               #eax=fibo_cur+fibo_prev
  34:   11 f2                   adc    %esi,%edx
  36:   83 c7 01                add    $0x1,%edi               #edi=edi+1, i incremented
  39:   89 dd                   mov    %ebx,%ebp               #ebp=fibo_prev
  3b:   89 c3                   mov    %eax,%ebx               #ebx=fibo_prev+fibo_cur
  3d:   39 cf                   cmp    %ecx,%edi               #comparing i and n
  3f:   76 ef                   jbe    30 <fibo_iter+0x30>     #jump to loop start if i<=n
  41:   5e                      pop    %esi                    #restoring stack
  42:   5f                      pop    %edi
  43:   5b                      pop    %ebx
  44:   5d                      pop    %ebp
  45:   c3                      ret
```

Unnecessary moves and store instructions related to stack removed. Some arithmetic operations not done by requiring one operand in eax register.

g. gcd O0

```
Disassembly of section .text:

00000000 <gcd1>:
   0:   55                      push   %ebp                    #push ebp
   1:   89 e5                   mov    %esp,%ebp               #ebp=esp
   3:   83 ec 18                sub    $0x18,%esp              #0x18 bytes on stack
   6:   8b 45 0c                mov    0xc(%ebp),%eax          #eax=b second arg.
   9:   8b 4d 08                mov    0x8(%ebp),%ecx          #ecx=a  first arg.
   c:   83 7d 0c 00             cmpl   $0x0,0xc(%ebp)          #comparing b with 0
  10:   89 45 f8                mov    %eax,-0x8(%ebp)         #storing arg a and b on stack
  13:   89 4d f4                mov    %ecx,-0xc(%ebp)
  16:   75 08                   jne    20 <gcd1+0x20>          #if b!=0 then jump to loc.20 to return a
  18:   8b 45 08                mov    0x8(%ebp),%eax          #eax=a
  1b:   89 45 fc                mov    %eax,-0x4(%ebp)         #storing a on stack
  1e:   eb 21                   jmp    41 <gcd1+0x41>          #jump to return from func.
  20:   8b 45 0c                mov    0xc(%ebp),%eax          #eax=b
  23:   8b 4d 08                mov    0x8(%ebp),%ecx          #ecx=a
  26:   89 45 f0                mov    %eax,-0x10(%ebp)        #storing current a and b on stack
  29:   89 c8                   mov    %ecx,%eax               #eax=a
  2b:   99                      cltd
  2c:   f7 7d 0c                idivl  0xc(%ebp)               #compute a%b
  2f:   8b 4d f0                mov    -0x10(%ebp),%ecx
  32:   89 0c 24                mov    %ecx,(%esp)             #pushing arg b and a%b on top of stack
  35:   89 54 24 04             mov    %edx,0x4(%esp)
  39:   e8 fc ff ff ff          call   3a <gcd1+0x3a>          #call gcd with b and a%b
  3e:   89 45 fc                mov    %eax,-0x4(%ebp)
  41:   8b 45 fc                mov    -0x4(%ebp),%eax         #value to be returned in eax
  44:   83 c4 18                add    $0x18,%esp              #restoring stack
  47:   5d                      pop    %ebp
  48:   c3                      ret
  49:   8d b4 26 00 00 00 00    lea    0x0(%esi,%eiz,1),%esi
```

```
00000050 <gcd2>:
  50:   55                      push   %ebp
  51:   89 e5                   mov    %esp,%ebp              #ebp=stack pointer
  53:   83 ec 08                sub    $0x8,%esp              #8 bytes on stack
  56:   8b 45 0c                mov    0xc(%ebp),%eax         #eax=b
  59:   8b 4d 08                mov    0x8(%ebp),%ecx         #ecx=a
  5c:   89 45 fc                mov    %eax,-0x4(%ebp)        #storing current values of a and b on stack
  5f:   89 4d f8                mov    %ecx,-0x8(%ebp)
  62:   8b 45 08                mov    0x8(%ebp),%eax         #eax=a
  65:   3b 45 0c                cmp    0xc(%ebp),%eax         #comparing a and b
  68:   74 22                   je     8c <gcd2+0x3c>         #jump to end of loop if a=b else fallthrough
  6a:   8b 45 08                mov    0x8(%ebp),%eax         #eax=a
  6d:   3b 45 0c                cmp    0xc(%ebp),%eax         #compare a and b for the if condition
  70:   7e 0d                   jle    7f <gcd2+0x2f>         #if a<=b then jump to end of condition loc.7f
  72:   8b 45 0c                mov    0xc(%ebp),%eax         #eax=b
  75:   8b 4d 08                mov    0x8(%ebp),%ecx         #ecx=a
  78:   29 c1                   sub    %eax,%ecx              #ecx=a-b
  7a:   89 4d 08                mov    %ecx,0x8(%ebp)         #storing a-b on stack as a
  7d:   eb 0b                   jmp    8a <gcd2+0x3a>         #jump after the else condition else else condition
  7f:   8b 45 08                mov    0x8(%ebp),%eax         #eax=a
  82:   8b 4d 0c                mov    0xc(%ebp),%ecx         #ecx=b
  85:   29 c1                   sub    %eax,%ecx              #ecx=b-a
  87:   89 4d 0c                mov    %ecx,0xc(%ebp)         #storing b-a on stack as b
  8a:   eb d6                   jmp    62 <gcd2+0x12>         #jump to start of loop
  8c:   8b 45 08                mov    0x8(%ebp),%eax         #eax=a, value to be returned
  8f:   83 c4 08                add    $0x8,%esp              #restoring stack
  92:   5d                      pop    %ebp
  93:   c3                      ret
  94:   8d b6 00 00 00 00       lea    0x0(%esi),%esi
  9a:   8d bf 00 00 00 00       lea    0x0(%edi),%edi
```

Screenshot

```
000000a0 <gcd3>:
  a0:   55                      push   %ebp
  a1:   89 e5                   mov    %esp,%ebp
  a3:   83 ec 0c                sub    $0xc,%esp              #12 bytes on stack
  a6:   8b 45 0c                mov    0xc(%ebp),%eax         #eax=b
  a9:   8b 4d 08                mov    0x8(%ebp),%ecx         #ecx=b
  ac:   89 45 f8                mov    %eax,-0x8(%ebp)        #current value of a and b stored on stack
  af:   89 4d f4                mov    %ecx,-0xc(%ebp)
  b2:   83 7d 0c 00             cmpl   $0x0,0xc(%ebp)         #comparing b with 0
  b6:   74 18                   je     d0 <gcd3+0x30>         #if b=0 then jump to terminate the loop to return the value else falthrough
  b8:   8b 45 0c                mov    0xc(%ebp),%eax         #eax=b
  bb:   89 45 fc                mov    %eax,-0x4(%ebp)        #storing b on stack
  be:   8b 45 08                mov    0x8(%ebp),%eax         #eax=a
  c1:   99                      cltd
  c2:   f7 7d 0c                idivl  0xc(%ebp)              #compute a%b
  c5:   89 55 0c                mov    %edx,0xc(%ebp)         #b=a%b
  c8:   8b 55 fc                mov    -0x4(%ebp),%edx        #edx=value of b stored on stack
  cb:   89 55 08                mov    %edx,0x8(%ebp)         #storing b at palce of a
  ce:   eb e2                   jmp    b2 <gcd3+0x12>         #jump to start of loop
  d0:   8b 45 08                mov    0x8(%ebp),%eax         #eax=a value to be returned
  d3:   83 c4 0c                add    $0xc,%esp              #restoring stack
  d6:   5d                      pop    %ebp
  d7:   c3                      ret
```

g. gcd O2

```
gcd.i386.O2.o:     file format elf32-i386


Disassembly of section .text:

00000000 <gcd1>:
   0:   8b 54 24 08             mov    0x8(%esp),%edx          #edx=b
   4:   8b 44 24 04             mov    0x4(%esp),%eax          #eax=a
   8:   85 d2                   test   %edx,%edx               #test if b=0
   a:   74 12                   je     1e <gcd1+0x1e>          #ig b=0 then jump to 0x1e to return a
   c:   8d 74 26 00             lea    0x0(%esi,%eiz,1),%esi   #load effective address esi+eiz*1 in esi
  10:   89 d1                   mov    %edx,%ecx               #ecx=b
  12:   99                      cltd
  13:   f7 f9                   idiv   %ecx                    #compute a%b
  15:   89 c8                   mov    %ecx,%eax               #eax=a%b
  17:   85 d2                   test   %edx,%edx               #test if b=0
  19:   75 f5                   jne    10 <gcd1+0x10>          #if not jump to start of program (with updated arguements in stack)
  1b:   89 c8                   mov    %ecx,%eax               #eax=a(new value of a)
  1d:   c3                      ret                            #return from function
  1e:   c3                      ret
  1f:   90                      nop

00000020 <gcd2>:
  20:   57                      push   %edi                    #push edi, esi
  21:   56                      push   %esi
  22:   8b 4c 24 10             mov    0x10(%esp),%ecx         #ecx=b
  26:   8b 44 24 0c             mov    0xc(%esp),%eax          #eax=a
  2a:   39 c8                   cmp    %ecx,%eax               #compare a and b
  2c:   74 19                   je     47 <gcd2+0x27>          #if a=b then jump to end of loop, eax contains value to be returned
  2e:   31 d2                   xor    %edx,%edx               #clearing edx
  30:   39 c1                   cmp    %eax,%ecx               #comparing a and b for if condition
  32:   be 00 00 00 00          mov    $0x0,%esi               #esi=0
  37:   0f 4c f1                cmovl  %ecx,%esi               #esi=b
  3a:   89 c7                   mov    %eax,%edi               #edi=a
  3c:   0f 4c fa                cmovl  %edx,%edi
  3f:   29 f0                   sub    %esi,%eax               #a=a-b
  41:   29 f9                   sub    %edi,%ecx               #b=b-a
  43:   39 c8                   cmp    %ecx,%eax               #compare a and b
  45:   75 e9                   jne    30 <gcd2+0x10>          #if a!=b then jump to start of loop else return
  47:   5e                      pop    %esi                    #restoring stack
  48:   5f                      pop    %edi
  49:   c3                      ret
  4a:   8d b6 00 00 00 00       lea    0x0(%esi),%esi
```

Plain Text Tab Width: 8

```
00000050 <gcd3>:
  50:   8b 54 24 08             mov    0x8(%esp),%edx          #edx=b
  54:   8b 44 24 04             mov    0x4(%esp),%eax          #eax=a
  58:   85 d2                   test   %edx,%edx               #test if b=0
  5a:   74 12                   je     6e <gcd3+0x1e>          #if b=0 then jump to return stmt.
  5c:   8d 74 26 00             lea    0x0(%esi,%eiz,1),%esi   #load effective address in esi
  60:   89 d1                   mov    %edx,%ecx               #ecx=ed
  62:   99                      cltd
  63:   f7 f9                   idiv   %ecx                    #compute a%b
  65:   89 c8                   mov    %ecx,%eax               #eax=a%b
  67:   85 d2                   test   %edx,%edx               #test if b=0
  69:   75 f5                   jne    60 <gcd3+0x10>          #if a!=b then jump to start of loop
  6b:   89 c8                   mov    %ecx,%eax               #eax=value to be returned
  6d:   c3                      ret
  6e:   c3                      ret
```

Some moves and store related to storing intermediate values, local variables on stack are removed

i. loops O0

```
00000000 <is_sorted>:
   0:   55                      push   %ebp                     #push ebp
   1:   89 e5                   mov    %esp,%ebp                #ebp=esp
   3:   83 ec 10                sub    $0x10,%esp               #esp=esp-16, 16 bytes on stack allocated
   6:   8b 45 0c                mov    0xc(%ebp),%eax           #eax=ebp+12,eax=n
   9:   8b 4d 08                mov    0x8(%ebp),%ecx           #ecx=ebp+8,ecx=&a
   c:   c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%ebp)          #ebp-8=0,i=0
  13:   89 45 f4                mov    %eax,-0xc(%ebp)          #ebp-12=n
  16:   89 4d f0                mov    %ecx,-0x10(%ebp)         #ebp-16=&a
  19:   8b 45 f8                mov    -0x8(%ebp),%eax          #eax=i
  1c:   8b 4d 0c                mov    0xc(%ebp),%ecx           #ecx=n
  1f:   83 e9 01                sub    $0x1,%ecx                #ecx=ecx-1,ecx=n-1
  22:   39 c8                   cmp    %ecx,%eax                #compare i and n-1
  24:   7d 24                   jge    4a <is_sorted+0x4a>      #if(i>=n-1) then jump to loc. 4a
  26:   8b 45 08                mov    0x8(%ebp),%eax           #eax=&a
  29:   8b 4d f8                mov    -0x8(%ebp),%ecx          #ecx=i
  2c:   8b 14 88                mov    (%eax,%ecx,4),%edx       #edx=eax+ecx*4 =a[i]
  2f:   8b 44 88 04             mov    0x4(%eax,%ecx,4),%eax    #eax=eax+ecx*4=a[i+1]
  33:   39 c2                   cmp    %eax,%edx                #compare a[i],a[i+1]
  35:   7e 06                   jle    3d <is_sorted+0x3d>      #if (a[i+1]<=a[i]) then jump to loc. 3d (back to loop)
  37:   c6 45 ff 00             movb   $0x0,-0x1(%ebp)          #move byte, ebp-1=0,value to be returned
  3b:   eb 11                   jmp    4e <is_sorted+0x4e>      #jump to 4e to terminate the loop
  3d:   eb 00                   jmp    3f <is_sorted+0x3f>      #jump to 3f
  3f:   8b 45 f8                mov    -0x8(%ebp),%eax          #eax=i
  42:   83 c0 01                add    $0x1,%eax                #eax=i+1
  45:   89 45 f8                mov    %eax,-0x8(%ebp)          #ebp-8=i+1
  48:   eb cf                   jmp    19 <is_sorted+0x19>      #jump to start of loop
  4a:   c6 45 ff 01             movb   $0x1,-0x1(%ebp)          #move byte, ebp-1=1, value to be returned
  4e:   8a 45 ff                mov    -0x1(%ebp),%al           #al=ebp-1 (0/1)
  51:   24 01                   and    $0x1,%al                 #al=al & 1
  53:   0f b6 c0                movzbl %al,%eax                 #eax=(al & 1)
  56:   83 c4 10                add    $0x10,%esp               #esp=ep+16, stack restored
  59:   5d                      pop    %ebp
  5a:   c3                      ret
  5b:   90                      nop                             #no operation inst. inserted
  5c:   8d 74 26 00             lea    0x0(%esi,%eiz,1),%esi
```

```
00000060 <add_arrays>:
  60:   55                      push   %ebp                        #pushing registers on stack
  61:   89 e5                   mov    %esp,%ebp
  63:   56                      push   %esi
  64:   83 ec 14                sub    $0x14,%esp                  #esp=esp-20, 20 bytes on stack allocated
  67:   8b 45 14                mov    0x14(%ebp),%eax             #eax=ebp+ox14, eax=n
  6a:   8b 4d 10                mov    0x10(%ebp),%ecx             #ecx=&c
  6d:   8b 55 0c                mov    0xc(%ebp),%edx              #edx=&b
  70:   8b 75 08                mov    0x8(%ebp),%esi              #esi=&a
  73:   89 55 f8                mov    %edx,-0x8(%ebp)             #ebp-8=&b   storing (arguements)content on stack
  76:   c7 45 f4 00 00 00 00    movl   $0x0,-0xc(%ebp)             #ebp-12=0,i=0
  7d:   89 45 f0                mov    %eax,-0x10(%ebp)            #ebp-16=n
  80:   89 4d ec                mov    %ecx,-0x14(%ebp)            #ebp-20=&c
  83:   89 75 e8                mov    %esi,-0x18(%ebp)            #ebp-24=&a
  86:   8b 45 f4                mov    -0xc(%ebp),%eax             #eax=i
  89:   3b 45 14                cmp    0x14(%ebp),%eax             #comparing n and i
  8c:   7d 22                   jge    b0 <add_arrays+0x50>        #if (i>n) then jump to loc b0(end of loop)
  8e:   8b 45 08                mov    0x8(%ebp),%eax              #eax=&a
  91:   8b 4d f4                mov    -0xc(%ebp),%ecx             #ecx=i
  94:   8b 04 88                mov    (%eax,%ecx,4),%eax          #eax=a[i]
  97:   8b 55 f8                mov    -0x8(%ebp),%edx             #edx=&b
  9a:   8b 14 8a                mov    (%edx,%ecx,4),%edx          #edx=b[i]
  9d:   01 d0                   add    %edx,%eax                   #eax=eax+edx=a[i]+b[i]
  9f:   8b 55 10                mov    0x10(%ebp),%edx             #edx=&c
  a2:   89 04 8a                mov    %eax,(%edx,%ecx,4)          #c[i]=a[i]+b[i]
  a5:   8b 45 f4                mov    -0xc(%ebp),%eax             #eax=i
  a8:   83 c0 01                add    $0x1,%eax                   #eax=i+1
  ab:   89 45 f4                mov    %eax,-0xc(%ebp)             #ebp-12=i+1
  ae:   eb d6                   jmp    86 <add_arrays+0x26>        #jump to loop start
  b0:   83 c4 14                add    $0x14,%esp                  #restoring stack
  b3:   5e                      pop    %esi
  b4:   5d                      pop    %ebp
  b5:   c3                      ret
  b6:   8d 76 00                lea    0x0(%esi),%esi
  b9:   8d bc 27 00 00 00 00    lea    0x0(%edi,%eiz,1),%edi
```

```
000000c0 <sum>:
  c0:   55                      push   %ebp                        #pushing registers on stack
  c1:   89 e5                   mov    %esp,%ebp
  c3:   83 ec 10                sub    $0x10,%esp                  #esp=esp-16, 16 bytes allocated on stack
  c6:   8b 45 0c                mov    0xc(%ebp),%eax              #eax=n
  c9:   8b 4d 08                mov    0x8(%ebp),%ecx              #ecx=&a
  cc:   c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%ebp)             #ebp-4=0, ret=0
  d3:   c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%ebp)             #ebp-8=0, i=0
  da:   89 45 f4                mov    %eax,-0xc(%ebp)             #ebp-12=n  storing arguements on stack
  dd:   89 4d f0                mov    %ecx,-0x10(%ebp)            #ebp-16=&a
  e0:   8b 45 f8                mov    -0x8(%ebp),%eax             #eax=i
  e3:   3b 45 0c                cmp    0xc(%ebp),%eax              #compare i and n
  e6:   7d 1d                   jge    105 <sum+0x45>              #if(i>=n) then jump to 105 (end of loop)
  e8:   8b 45 08                mov    0x8(%ebp),%eax              #eax=&a
  eb:   8b 4d f8                mov    -0x8(%ebp),%ecx             #ecx=i
  ee:   8a 14 08                mov    (%eax,%ecx,1),%dl           #dl=a[i], (byte)
  f1:   0f b6 c2                movzbl %dl,%eax                    #eax=a[i]
  f4:   03 45 fc                add    -0x4(%ebp),%eax             #eax=ret+a[i]
  f7:   89 45 fc                mov    %eax,-0x4(%ebp)             #ret=ret+a[i]
  fa:   8b 45 f8                mov    -0x8(%ebp),%eax             #eax=i
  fd:   83 c0 01                add    $0x1,%eax                   #eax=i+1
 100:   89 45 f8                mov    %eax,-0x8(%ebp)             #i=i+1
 103:   eb db                   jmp    e0 <sum+0x20>               #jump to start of loop
 105:   8b 45 fc                mov    -0x4(%ebp),%eax             #eax=ret, value to be returned
 108:   83 c4 10                add    $0x10,%esp                  #restoring stack
 10b:   5d                      pop    %ebp
 10c:   c3                      ret
 10d:   8d 76 00                lea    0x0(%esi),%esi
```

Screenshot

```
00000110 <sumn>:
 110:   55                      push   %ebp
 111:   89 e5                   mov    %esp,%ebp
 113:   83 ec 0c                sub    $0xc,%esp                    #esp=esp-12, 12 bytes allocated on stack
 116:   8b 45 08                mov    0x8(%ebp),%eax               #eax=n
 119:   c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%ebp)              #ebp-4=0, ret=0
 120:   c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%ebp)              #ebp-8=0, i=0
 127:   89 45 f4                mov    %eax,-0xc(%ebp)              #ebp-12=n
 12a:   8b 45 f8                mov    -0x8(%ebp),%eax              #eax=i
 12d:   3b 45 08                cmp    0x8(%ebp),%eax               #compare i and n
 130:   7d 14                   jge    146 <sumn+0x36>              #if (i>=n) then jump to loc. 146(end of loop)
 132:   8b 45 fc                mov    -0x4(%ebp),%eax              #eax=ret
 135:   03 45 f8                add    -0x8(%ebp),%eax              #eax=ret+i
 138:   89 45 fc                mov    %eax,-0x4(%ebp)              #ret=ret+i
 13b:   8b 45 f8                mov    -0x8(%ebp),%eax              #eax=i
 13e:   83 c0 01                add    $0x1,%eax                    #eax=i+1
 141:   89 45 f8                mov    %eax,-0x8(%ebp)              #i=i+1
 144:   eb e4                   jmp    12a <sumn+0x1a>              #jump to 12a(start of loop)
 146:   8b 45 fc                mov    -0x4(%ebp),%eax              #eax=ret, value to be returned
 149:   83 c4 0c                add    $0xc,%esp                    #restoring stack
 14c:   5d                      pop    %ebp
 14d:   c3                      ret
```

j. loops O2

```
00000000 <is_sorted>:
   0:   55                      push   %ebp                         #pushing registers
   1:   53                      push   %ebx
   2:   57                      push   %edi
   3:   56                      push   %esi
   4:   8b 4c 24 18             mov    0x18(%esp),%ecx              #ecx=esp+0x18=n
   8:   b0 01                   mov    $0x1,%al                     #al=1
   a:   83 f9 02                cmp    $0x2,%ecx                    #compare n and 2
   d:   7c 3e                   jl     4d <is_sorted+0x4d>          #if(n<2) then jump to 4d end of loop(as n-1=0 or less)
   f:   8b 44 24 14             mov    0x14(%esp),%eax              #eax=&a
  13:   83 c1 ff                add    $0xffffffff,%ecx             #ecx=ecx+int_max
  16:   8b 18                   mov    (%eax),%ebx                  #ebx=&a
  18:   31 f6                   xor    %esi,%esi                    #clear esi,esi=0,i=0
  1a:   89 cf                   mov    %ecx,%edi                    #edi=ecx
  1c:   c1 ff 1f                sar    $0x1f,%edi                   #
  1f:   31 d2                   xor    %edx,%edx
  21:   eb 0d                   jmp    30 <is_sorted+0x30>
  23:   90                      nop                                 #no operation
  24:   90                      nop
  25:   90                      nop
  26:   90                      nop
  27:   90                      nop
  28:   90                      nop
  29:   90                      nop
  2a:   90                      nop
  2b:   90                      nop
  2c:   90                      nop
  2d:   90                      nop
  2e:   90                      nop
  2f:   90                      nop
  30:   89 dd                   mov    %ebx,%ebp                    #ebp=ebx
  32:   83 c6 01                add    $0x1,%esi                    #esi=i+1
  35:   83 d2 00                adc    $0x0,%edx                    #edx=edx+0
  38:   8b 1c b0                mov    (%eax,%esi,4),%ebx           #ebx=a[i]
  3b:   39 dd                   cmp    %ebx,%ebp                    #compare a[i] and a[i+1]
  3d:   7f 0c                   jg     4b <is_sorted+0x4b>          #if(a[i]>a[i+1]) then jump to 4b(end of loop)
  3f:   39 ce                   cmp    %ecx,%esi                    #compare i and n
  41:   89 d5                   mov    %edx,%ebp                    #ebp=edx
  43:   19 fd                   sbb    %edi,%ebp                    #
  45:   7c e9                   jl     30 <is_sorted+0x30>          #if(i<n) then jump to 30(start of loop)
  47:   b0 01                   mov    $0x1,%al                     #al=1
```

```
 49:   eb 02                  jmp    4d <is_sorted+0x4d>            #jump to 4d to return 1
 4b:   31 c0                  xor    %eax,%eax                      #eax=0, returning 0
 4d:   5e                     pop    %esi                           #restoring stack
 4e:   5f                     pop    %edi
 4f:   5b                     pop    %ebx
 50:   5d                     pop    %ebp
 51:   c3                     ret
 52:   8d b4 26 00 00 00 00   lea    0x0(%esi,%eiz,1),%esi
 59:   8d bc 27 00 00 00 00   lea    0x0(%edi,%eiz,1),%edi
```

```
00000450 <sumn>:
 450:   8b 4c 24 04           mov    0x4(%esp),%ecx                 #ecx=esp+4, ecx=n
 454:   85 c9                 test   %ecx,%ecx                      #
 456:   7e 13                 jle    46b <sumn+0x1b>                #if
 458:   8d 41 ff              lea    -0x1(%ecx),%eax                #load effective address, eax=ecx-1
 45b:   8d 51 fe              lea    -0x2(%ecx),%edx                #edx=ecx-2
 45e:   f7 e2                 mul    %edx                           #eax=edx*eax
 460:   0f a4 c2 1f           shld   $0x1f,%eax,%edx
 464:   8d 04 0a              lea    (%edx,%ecx,1),%eax
 467:   83 c0 ff              add    $0xffffffff,%eax
 46a:   c3                    ret
 46b:   31 c0                 xor    %eax,%eax
 46d:   c3                    ret
```

sumn and is_sorted functions remove unnecessary move and stores. The code for add_arrays and sum is bigger in size for this level.

k. print_args O0

```
print_arg.i386.O0.o:     file format elf32-i386


Disassembly of section .text:

00000000 <print_arg>:
   0:   55                      push   %ebp                    #push base pointer
   1:   89 e5                   mov    %esp,%ebp               #ebp=esp
   3:   83 ec 18                sub    $0x18,%esp              #esp=esp-ox18
   6:   8b 45 0c                mov    0xc(%ebp),%eax            #eax=ebp+12  argv
   9:   8b 4d 08                mov    0x8(%ebp),%ecx            #ecx=ebp+8  argc
   c:   89 45 f8                mov    %eax,-0x8(%ebp)           #ebp-8=eax  argv
   f:   83 7d 08 02             cmpl   $0x2,0x8(%ebp)            #comparing 2 with ebp+8(argc)
  13:   89 4d f4                mov    %ecx,-0xc(%ebp)           #ebp-12=ecx  argc
  16:   74 09                   je     21 <print_arg+0x21>       #jump on equal to loc.21,return value in ebp-4
  18:   c7 45 fc ff ff ff ff    movl   $0xffffffff,-0x4(%ebp)    #ebp-4=int_max (-1)
  1f:   eb 22                   jmp    43 <print_arg+0x43>       #jump to loc.43
  21:   8d 05 00 00 00 00       lea    0x0,%eax                  #load effective addresss 0 in eax
  27:   8b 4d f8                mov    -0x8(%ebp),%ecx           #ecx=ebp-8
  2a:   8b 49 04                mov    0x4(%ecx),%ecx            #ecx=ecx+4
  2d:   89 04 24                mov    %eax,(%esp)               #esp=eax
  30:   89 4c 24 04             mov    %ecx,0x4(%esp)            #esp+4=ecx
  34:   e8 fc ff ff ff          call   35 <print_arg+0x35>       #call the next arguement
  39:   c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%ebp)           #ebp-4=0, value to be returned
  40:   89 45 f0                mov    %eax,-0x10(%ebp)          #ebp-16=eax
  43:   8b 45 fc                mov    -0x4(%ebp),%eax           #eax=eax-4 value to be returned
  46:   83 c4 18                add    $0x18,%esp                #restoring stack
  49:   5d                      pop    %ebp
  4a:   c3                      ret
```

l. print_args O2

```
print_arg.i386.O2.o:     file format elf32-i386


Disassembly of section .text:

00000000 <print_arg>:
   0:   83 ec 0c                sub    $0xc,%esp               #12 bytes on stack , esp=esp-12
   3:   b8 ff ff ff ff          mov    $0xffffffff,%eax        #eax=Int_max
   8:   83 7c 24 10 02          cmpl   $0x2,0x10(%esp)         #comparing 2 with content of esp+16 (first arguement argc)
   d:   75 19                   jne    28 <print_arg+0x28>     #if not equal then jump to loc.28
   f:   8b 44 24 14             mov    0x14(%esp),%eax         #eax=esp+20
  13:   83 ec 08                sub    $0x8,%esp               #esp=esp-8
  16:   ff 70 04                pushl  0x4(%eax)               #eax+4 content pushed on stack
  19:   68 00 00 00 00          push   $0x0                    #0 pushed on stack
  1e:   e8 fc ff ff ff          call   1f <print_arg+0x1f>     #printf called with arg argv[1]
  23:   83 c4 10                add    $0x10,%esp              #restoring esp
  26:   31 c0                   xor    %eax,%eax               #clearing eax register
  28:   83 c4 0c                add    $0xc,%esp               #restoring esp
  2b:   c3                      ret                            #return
```

Some move, store instructions removed related to stack.

3. REFERENCES

a. https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html