Assignment 1 - Programming Assignment Report

Anagh Singh

January 10, 2016

Question 1

Write a C program to check whether the machine uses the IEEE 754 standards for double precision FP representation; Find the largest and smallest positive numbers that it can represent, with and without the normalized representation. Check the representation for NaN, and infinity; Also find the machine epsilon.

Solution:

IEEE 754 is a standard used for representing and manipulating floating-point quantities that is followed by all modern computer systems. The first bit is the sign (0 for positive, 1 for negative). The next 8 bits are the exponent in -127 binary notation - meaning 01111111 = 127 represents an exponent of 0, 1000000 = 128, represents 1, 01111110 = 126 represents -1, and so forth. The mantissa fits in the remaining 23 bits, with its leading 1 stripped off.

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <values.h>
4
5  //Declare value to test against
6  const int EndianTest = 0x04030201;
7
8  //Check the endian nature of the machine
9  #define LITTLE_ENDIAN() (*((const char *) &EndianTest) == 0x01)
10
11 //Extract nth LSB from object stored in lvalue x
12 #define GET_BIT(x, n) ((((const char *) &x)[LITTLE_ENDIAN() ? (n) / CHARBITS : sizeof(x) - (n)
       / CHARBITS - 1] >> ((n) % CHARBITS)) & 0x01)
13
14 void FP_representation(float f)
15 {
16     int i;
17
18     i = FLOATBITS - 1;
19     //First bit declaring sign
20     putchar(GET_BIT((f), (i)) ? '1' : '0');
21     //Next 8 bits for the excess 127 notation
22     printf(" ");
23     for(i--; i >= 23; i--)
24         putchar(GET_BIT((f), (i)) ? '1' : '0');
25     //Next 23 bits for the mantissa
26     printf(" ");
27     for(; i >= 0; i--)
28         putchar(GET_BIT((f), (i)) ? '1' : '0');
29
30 }
31
32 int main()
33 {
34     float f;
35     while(scanf("%f", &f) == 1) {
36         printf("%10g = %24.17g = ", f, f);
37         FP_representation(f);
38         printf("\n");
39     }
40     return 3301;
41 }
```

The greatest value that can be represented in single precision, is approximately 3.4028235×10^{38}, which is equal to $1.11111111111111111111111_b \times 2^{11111110_b - 127}$ or $7f7fffff$ in Hex. The smallest value that can be represented in single precision, is approximately $1.17549435 \times 10^{-38}$ which is 00800000 in Hex.

The machine epsilon is found using the following code, and the value is 2^{-23}.

```
1  #include <stdio.h>
2
3  int epsilon() {
4      int pow = 0;
5      float eps = 1;
```

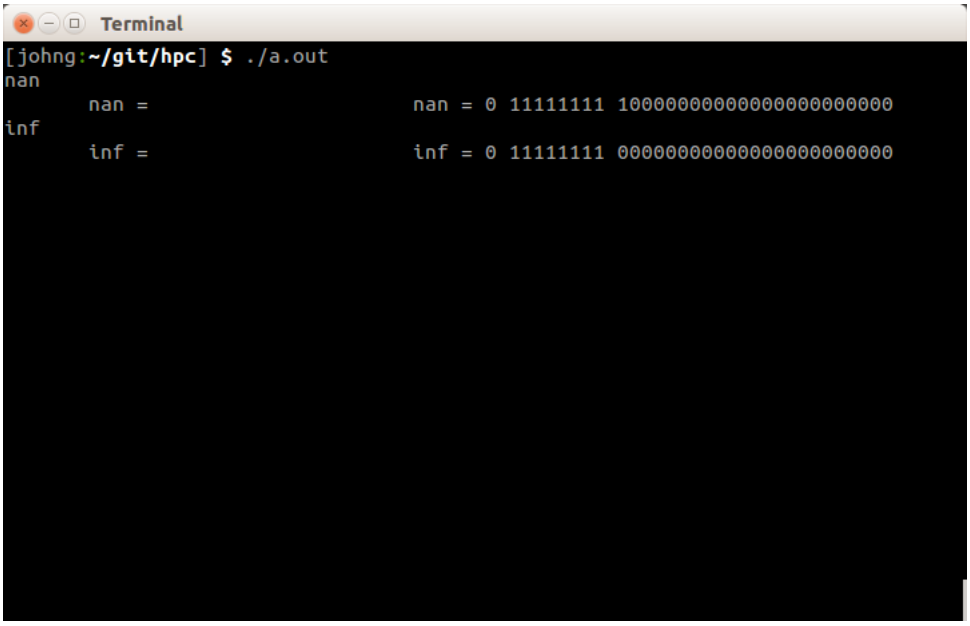

Figure 1: NaN and Inf

```
6       while (eps + 1 != 1) {
7           eps /= 2;
8           --pow;
9       }
10      return pow + 1;
11  }
12
13  int main() {
14      printf("Epsilon: 2^ %d \n",epsilon());
15      return 3301;
16  }
```

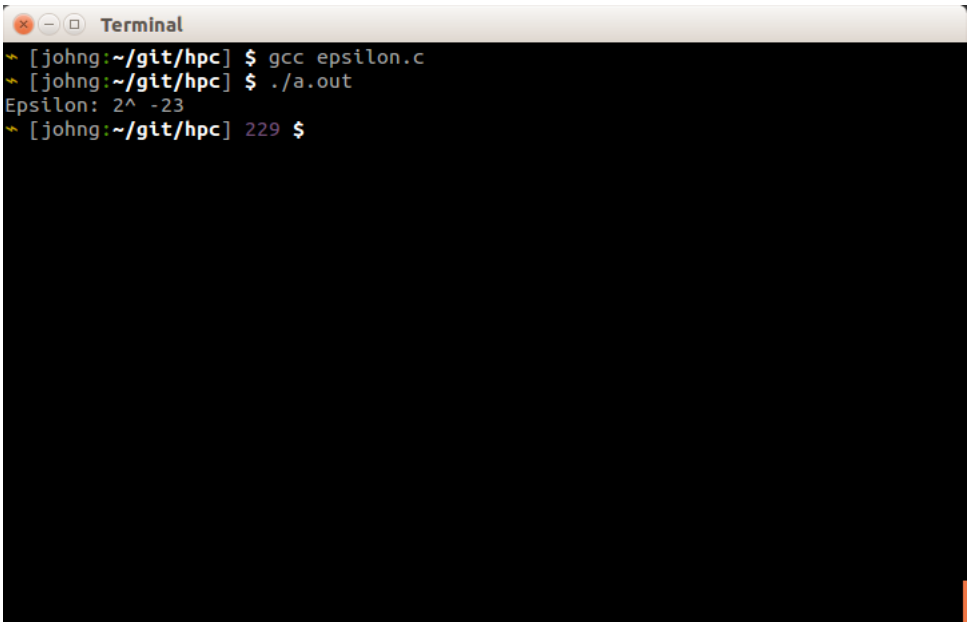

Figure 2: Machine Epsilon

Question 2

Write a C program to identify in which region the following types of variables are stored: (a) global (b) local; (c) static, and (d) dynamically allocated .

Solution:

The memory in a computer is divided up into distinct parts with the layers being as follows:

- Command Line Arguments and Variables
- Stack - Local variables
- Heap - Dynamic variables
- Data Segment - Global variables
- Text Segment - Static variables

The addresses of all these areas are in decreasing order since they are stored as described above. To check this, we print the addresses of all the variables which might be stored in those areas.

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  //Global variables
5  int g1=3;
6  int g2=301;
7
8  // function to test stack
9  void Call2() {
10     int var1;
11     int var2;
12     printf("On Stack through Call2:\t\t %u %u\n",&var1,&var2);
13 }
14
15 void Call1() {
16     int var1;
17     int var2;
18     printf("On Stack through Call1:\t\t %u %u\n",&var1,&var2);
19     Call2();
20 }
21
22 int main(int argc, char* argv[], char* evnp[]) {
23
24     printf("Printing addresses of all sections :-\n");
25     //Command line arguments
26     printf("Cmd Line and Env Var:\t\t %u %u %u\n",&argc,argv,evnp);
27
28     //Local variables go to stack and stack extends downwards
29     int var1;
30     int var2;
31     printf("On Stack through main:\t\t %u %u\n",&var1,&var2);
32     Call1();
33
34     //Dynamic Memory goes to heap and heap increases
35     void *array_1 = malloc(5);
36     void *array_2 = malloc(5);
37     printf("Heap Data:\t\t\t %u\n",array_2);
38     printf("Heap Data:\t\t\t %u\n",array_1);
39     free(array_1);
40     free(array_2);
41
42     // Global Variables go below the heap in the
43     //initialized and uninitialized data segment
44     printf("Global Variables:\t\t %u %u\n",&g1,&g2);
45
46     //Static Code goes to Text section of memory
47     printf("Text Data:\t\t\t %u %u\n ",main,Call1);
48     return 3301;
49 }
```

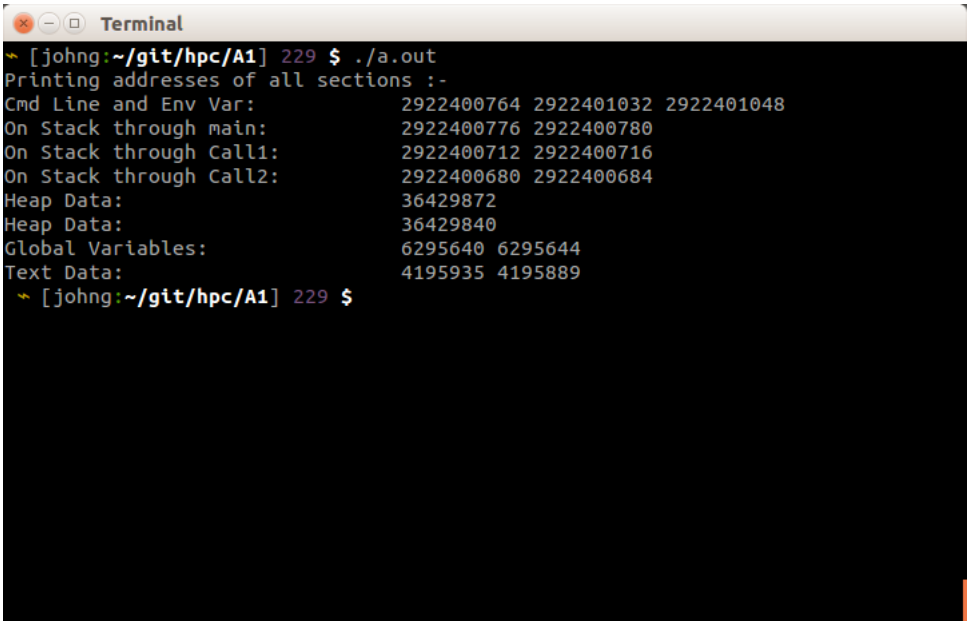

Figure 3: Memory Layout

Question 3

Write a C program to calculate the factorial of an integer. Generate its assembly code for two different processors (e.g., x86 and MIPS). Explain how the function call mechanism works, including how the parameters are passed, how the stack/function frames are allocated, how the return address is retrieved, and how the stack and frame pointers are restored.

Solution:

The code to calculate the factorial of an integer is as follows:

```
1  #include<stdio.h>
2  long long int factorial(long long int n);
3  int main()
4  {
5      int n;
6      printf("Enter an positive integer: ");
7      for(int i=0; i<5; i++)
8          scanf("%ld",&x[i]);
9      printf("Enter an positive integer: ");
10     for(int i=0; i<5; i++)
11         scanf("%ld",&y[i]);
12     for(int i=0; i<5; i++)
13         z[i]=x[i]+y[i];
14     for(int i=0; i<20; i++)
15         printf("%ld \t",z[i]);
16
17     return 3301;
18 }
19 long long int factorial(long long int n)
20 {
21     if(n!=1)
22      return n*factorial(n-1);
23 }
```

The x86 Assembly code for the above program is as follows:

```
1                   .Ltext0:
2                       .section    .rodata
3                   .LC0:
4  0000 456E7465        .string "Enter an positive integer: "
5       7220616E
6       20706F73
7       69746976
```

```
 8       6520696E
 9                   .LC1:
10  001c 256C6400       .string "%ld"
11                   .LC2:
12  0020 46616374       .string "Factorial of %ld = %ld\n"
13       6F726961
14       6C206F66
15       20256C64
16       203D2025
17                       .text
18                       .globl  main
19                   main:
20                   .LFB0:
21                       .cfi_startproc
22  0000 55              push    rbp
23                       .cfi_def_cfa_offset 16
24                       .cfi_offset 6, -16
25  0001 4889E5          mov rbp, rsp
26                       .cfi_def_cfa_register 6
27  0004 4883EC10        sub rsp, 16
28  0008 BF000000        mov edi, OFFSET FLAT:.LC0
29       00
30  000d B8000000        mov eax, 0
31       00
32  0012 E8000000        call    printf
33       00
34  0017 488D45FC        lea rax, [rbp-4]
35  001b 4889C6          mov rsi, rax
36  001e BF000000        mov edi, OFFSET FLAT:.LC1
37       00
38  0023 B8000000        mov eax, 0
39       00
40  0028 E8000000        call    __isoc99_scanf
41       00
42  002d 8B45FC          mov eax, DWORD PTR [rbp-4]
43  0030 4898            cdqe
44  0032 4889C7          mov rdi, rax
45  0035 E8000000        call    factorial
46       00
47  003a 4889C2          mov rdx, rax
48  003d 8B45FC          mov eax, DWORD PTR [rbp-4]
49  0040 89C6            mov esi, eax
50  0042 BF000000        mov edi, OFFSET FLAT:.LC2
51       00
52  0047 B8000000        mov eax, 0
53       00
54  004c E8000000        call    printf
55       00
56  0051 B8E50C00        mov eax, 3301
57       00
58  0056 C9              leave
59                       .cfi_def_cfa 7, 8
60  0057 C3              ret
61                       .cfi_endproc
62                   .LFE0:
63                       .globl  factorial
64                   factorial:
65                   .LFB1:
66                       .cfi_startproc
67  0058 55              push    rbp
68                       .cfi_def_cfa_offset 16
69                       .cfi_offset 6, -16
70  0059 4889E5          mov rbp, rsp
71                       .cfi_def_cfa_register 6
72  005c 4883EC10        sub rsp, 16
73  0060 48897DF8        mov QWORD PTR [rbp-8], rdi
74  0064 48837DF8        cmp QWORD PTR [rbp-8], 1
75       01
76  0069 7417            je  .L4
77  006b 488B45F8        mov rax, QWORD PTR [rbp-8]
78  006f 4883E801        sub rax, 1
79  0073 4889C7          mov rdi, rax
80  0076 E8000000        call    factorial
81       00
82  007b 480FAF45        imul    rax, QWORD PTR [rbp-8]
83       F8
```

```
84  0080 EB00           jmp .L3
85                      .L4:
86                      .L3:
87  0082 C9             leave
88                      .cfi_def_cfa 7, 8
89  0083 C3             ret
90                      .cfi_endproc
91                      .LFE1:
92                      .Letext0:
```

The MIPS Assembly code for the above program is as follows:

```
1                       .Ltext0:
2                           .section    .rodata
3                       .LC0:
4   0000 456E7465           .string "Enter an positive integer: "
5        7220616E
6        20706F73
7        69746976
8        6520696E
9                       .LC1:
10  001c 256C6400           .string "%ld"
11                      .LC2:
12  0020 46616374           .string "Factorial of %ld = %ld\n"
13       6F726961
14       6C206F66
15       20256C64
16       203D2025
17                          .text
18                          .globl  main
19                      main:
20                      .LFB0:
21                          .cfi_startproc
22  0000 55                 pushq   %rbp
23                          .cfi_def_cfa_offset 16
24                          .cfi_offset 6, -16
25  0001 4889E5             movq    %rsp, %rbp
26                          .cfi_def_cfa_register 6
27  0004 4883EC10           subq    $16, %rsp
28  0008 BF000000           movl    $.LC0, %edi
29       00
30  000d B8000000           movl    $0, %eax
31       00
32  0012 E8000000           call    printf
33       00
34  0017 488D45FC           leaq    -4(%rbp), %rax
35  001b 4889C6             movq    %rax, %rsi
36  001e BF000000           movl    $.LC1, %edi
37       00
38  0023 B8000000           movl    $0, %eax
39       00
40  0028 E8000000           call    __isoc99_scanf
41       00
42  002d 8B45FC             movl    -4(%rbp), %eax
43  0030 4898               cltq
44  0032 4889C7             movq    %rax, %rdi
45  0035 E8000000           call    factorial
46       00
47  003a 4889C2             movq    %rax, %rdx
48  003d 8B45FC             movl    -4(%rbp), %eax
49  0040 89C6               movl    %eax, %esi
50  0042 BF000000           movl    $.LC2, %edi
51       00
52  0047 B8000000           movl    $0, %eax
53       00
54  004c E8000000           call    printf
55       00
56  0051 B8E50C00           movl    $3301, %eax
57       00
58  0056 C9                 leave
59                          .cfi_def_cfa 7, 8
60  0057 C3                 ret
61                          .cfi_endproc
62                      .LFE0:
63                          .globl  factorial
64                      factorial:
65                      .LFB1:
```

```
66                          .cfi_startproc
67  0058 55                 pushq   %rbp
68                          .cfi_def_cfa_offset 16
69                          .cfi_offset 6, -16
70  0059 4889E5             movq    %rsp, %rbp
71                          .cfi_def_cfa_register 6
72  005c 4883EC10           subq    $16, %rsp
73  0060 48897DF8           movq    %rdi, -8(%rbp)
74  0064 48837DF8           cmpq    $1, -8(%rbp)
75       01
76  0069 7417               je  .L4
77  006b 488B45F8           movq    -8(%rbp), %rax
78  006f 4883E801           subq    $1, %rax
79  0073 4889C7             movq    %rax, %rdi
80  0076 E8000000           call    factorial
81       00
82  007b 480FAF45           imulq   -8(%rbp), %rax
83       F8
84  0080 EB00               jmp .L3
85                      .L4:
86                      .L3:
87  0082 C9                 leave
88                          .cfi_def_cfa 7, 8
89  0083 C3                 ret
90                          .cfi_endproc
91                      .LFE1:
92                      .Letext0:
```

The function call mechanism works as follows: The factorial function is called by name as shown in line 45. The value which has to be passed to the function is stored in RDI and copied into it by the register RAX. Post this, the function is called via name, and the arguments passed to it. Now whatever value was computed in the function call gets returned and stored again in the temporary register RAX. This is copied to the register RDX for storage, and the other arguments of the next instruction, which is the printf statement, are loaded into memory as shown in line 50. Upon proper completion, the function returns zero and is called in line 54. When the factorial function is called, new registers are declared, and values pushed onto them, for evaluation, as shown in line 67. The stack and frame pointers are kept track of in the Jump statements which are present in the code. After multiplication in line 82, the next instruction is to jump again, and check for the true condition. Upon checking, the factorial function is called again, in line 82. Upon completion of the loop, the leave and return instructions are issued which takes it to its last position on the stack, which is the print statement.

Question 4

Consider the Vector add program (Z[i] = X[i] + Y[i]) and compile it with and without optimization for your machine. See the differences in the assembly code generated with and without the optimizations. Study the loop unrolling gcc can do. Find the optimal unrolling factor for an array size of 16384.

Solution:

The code for vector addition can is as follows:

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int i;
6      int n=16384;
7      int x[n],y[n],z[n];
8      printf("Enter an positive integer: ");
9      for(i=0; i<n; i++)
10         scanf("%d",&x[i]);
11     printf("Enter an positive integer: ");
12     for(i=0; i<n; i++)
13         scanf("%d",&y[i]);
14     for(i=0; i<n; i++)
15         z[i]=x[i]+y[i];
```

```
16      for(i=0; i<n; i++)
17          printf("%d \t",z[i]);
18      printf("\n");
19      return 3301;
20  }
```

Assembly code without any optimizations:

```
1                       .Ltext0:
2                           .section    .rodata
3                       .LC0:
4   0000 456E7465           .string "Enter an positive integer: "
5        7220616E
6        20706F73
7        69746976
8        6520696E
9                       .LC1:
10  001c 256C6400           .string "%ld"
11                      .LC2:
12  0020 256C6420           .string "%ld \t"
13       0900
14                          .text
15                          .globl  main
16                      main:
17                      .LFB0:
18                          .cfi_startproc
19  0000 55                 pushq   %rbp
20                          .cfi_def_cfa_offset 16
21                          .cfi_offset 6, -16
22  0001 4889E5             movq    %rsp, %rbp
23                          .cfi_def_cfa_register 6
24  0004 4883EC70           subq    $112, %rsp
25  0008 BF000000           movl    $.LC0, %edi
26       00
27  000d B8000000           movl    $0, %eax
28       00
29  0012 E8000000           call    printf
30       00
31  0017 C7459C00           movl    $0, -100(%rbp)
32       000000
33  001e EB27               jmp .L2
34                      .L3:
35  0043 83459C01           addl    $1, -100(%rbp)
36  0020 488D45A0           leaq    -96(%rbp), %rax
37  0024 8B559C             movl    -100(%rbp), %edx
38  0027 4863D2             movslq  %edx, %rdx
39  002a 48C1E202           salq    $2, %rdx
40  002e 4801D0             addq    %rdx, %rax
41  0031 4889C6             movq    %rax, %rsi
42  0034 BF000000           movl    $.LC1, %edi
43       00
44  0039 B8000000           movl    $0, %eax
45       00
46  003e E8000000           call    __isoc99_scanf
47       00
48                      .L2:
49  0047 837D9C04           cmpl    $4, -100(%rbp)
50  004b 7ED3               jle .L3
51  004d BF000000           movl    $.LC0, %edi
52       00
53  0052 B8000000           movl    $0, %eax
54       00
55  0057 E8000000           call    printf
56       00
57  005c C7459C00           movl    $0, -100(%rbp)
58       000000
59  0063 EB27               jmp .L4
60                      .L5:
61  0088 83459C01           addl    $1, -100(%rbp)
62  0065 488D45C0           leaq    -64(%rbp), %rax
63  0069 8B559C             movl    -100(%rbp), %edx
64  006c 4863D2             movslq  %edx, %rdx
65  006f 48C1E202           salq    $2, %rdx
66  0073 4801D0             addq    %rdx, %rax
67  0076 4889C6             movq    %rax, %rsi
68  0079 BF000000           movl    $.LC1, %edi
69       00
```

```
70  007e B8000000        movl    $0, %eax
71       00
72  0083 E8000000        call    __isoc99_scanf
73       00
74                    .L4:
75  008c 837D9C04        cmpl    $4, -100(%rbp)
76  0090 7ED3            jle .L5
77  0092 C7459C00        movl    $0, -100(%rbp)
78       000000
79  0099 EB21            jmp .L6
80                    .L7:
81  00b8 83459C01        addl    $1, -100(%rbp)
82  009b 8B459C          movl    -100(%rbp), %eax
83  009e 4898            cltq
84  00a0 8B5485A0        movl    -96(%rbp,%rax,4), %edx
85  00a4 8B459C          movl    -100(%rbp), %eax
86  00a7 4898            cltq
87  00a9 8B4485C0        movl    -64(%rbp,%rax,4), %eax
88  00ad 01C2            addl    %eax, %edx
89  00af 8B459C          movl    -100(%rbp), %eax
90  00b2 4898            cltq
91  00b4 895485E0        movl    %edx, -32(%rbp,%rax,4)
92                    .L6:
93  00bc 837D9C04        cmpl    $4, -100(%rbp)
94  00c0 7ED9            jle .L7
95  00c2 C7459C00        movl    $0, -100(%rbp)
96       000000
97  00c9 EB1E            jmp .L8
98                    .L9:
99  00e5 83459C01        addl    $1, -100(%rbp)
100 00cb 8B459C          movl    -100(%rbp), %eax
101 00ce 4898            cltq
102 00d0 8B4485E0        movl    -32(%rbp,%rax,4), %eax
103 00d4 89C6            movl    %eax, %esi
104 00d6 BF000000        movl    $.LC2, %edi
105      00
106 00db B8000000        movl    $0, %eax
107      00
108 00e0 E8000000        call    printf
109      00
110                   .L8:
111 00e9 837D9C04        cmpl    $4, -100(%rbp)
112 00ed 7EDC            jle .L9
113 00ef B8E50C00        movl    $3301, %eax
114      00
115 00f4 C9              leave
116                      .cfi_def_cfa 7, 8
117 00f5 C3              ret
118                      .cfi_endproc
119                   .LFE0:
120                   .Letext0:
```

Assembly code with 2 levels of optimizations:

```
1                     .Ltext0:
2                        .section    .rodata.str1.1,"aMS",@progbits,1
3                     .LC0:
4   0000 456E7465        .string "Enter an positive integer: "
5        7220616E
6        20706F73
7        69746976
8        6520696E
9                     .LC1:
10  001c 256C6400        .string "%ld"
11                    .LC2:
12  0020 256C6420        .string "%ld \t"
13       0900
14                       .section    .text.startup,"ax",@progbits
15                       .p2align 4,,15
16                       .globl  main
17                    main:
18                    .LFB13:
19                       .cfi_startproc
20                    .LVL0:
21  0000 55              pushq   %rbp
22                       .cfi_def_cfa_offset 16
23                       .cfi_offset 6, -16
```

```
24                      .LBB8:
25                      .LBB9:
26  0001 BE000000           movl    $.LC0, %esi
27       00
28  0006 BF010000           movl    $1, %edi
29       00
30  000b 31C0               xorl    %eax, %eax
31                      .LBE9:
32                      .LBE8:
33  000d 53                 pushq   %rbx
34                          .cfi_def_cfa_offset 24
35                          .cfi_offset 3, -24
36  000e 4883EC68           subq    $104, %rsp
37                          .cfi_def_cfa_offset 128
38  0012 488D6C24           leaq    20(%rsp), %rbp
39       14
40  0017 4889E3             movq    %rsp, %rbx
41                      .LBB11:
42                      .LBB10:
43  001a E8000000           call    __printf_chk
44       00
45                      .LVL1:
46                      .L3:
47                      .LBE10:
48                      .LBE11:
49  001f 4889DE             movq    %rbx, %rsi
50  0022 31C0               xorl    %eax, %eax
51  0024 BF000000           movl    $.LC1, %edi
52       00
53  0029 4883C304           addq    $4, %rbx
54  002d E8000000           call    __isoc99_scanf
55       00
56                      .LVL2:
57  0032 4839EB             cmpq    %rbp, %rbx
58  0035 75E8               jne .L3
59                      .LVL3:
60  0037 488D5C24           leaq    32(%rsp), %rbx
61       20
62                      .LBB12:
63                      .LBB13:
64  003c BE000000           movl    $.LC0, %esi
65       00
66  0041 BF010000           movl    $1, %edi
67       00
68  0046 31C0               xorl    %eax, %eax
69  0048 488D6B14           leaq    20(%rbx), %rbp
70  004c E8000000           call    __printf_chk
71       00
72                      .LVL4:
73                      .L5:
74                      .LBE13:
75                      .LBE12:
76  0051 4889DE             movq    %rbx, %rsi
77  0054 31C0               xorl    %eax, %eax
78  0056 BF000000           movl    $.LC1, %edi
79       00
80  005b 4883C304           addq    $4, %rbx
81  005f E8000000           call    __isoc99_scanf
82       00
83                      .LVL5:
84  0064 4839EB             cmpq    %rbp, %rbx
85  0067 75E8               jne .L5
86  0069 31C0               xorl    %eax, %eax
87                      .L7:
88  007a 4883F814           cmpq    $20, %rax
89  007e 75EB               jne .L7
90  0080 488D5C24           leaq    64(%rsp), %rbx
91       40
92  0085 488D6C24           leaq    84(%rsp), %rbp
93       54
94  006b 8B1404             movl    (%rsp,%rax), %edx
95  006e 03540420           addl    32(%rsp,%rax), %edx
96  0072 89540440           movl    %edx, 64(%rsp,%rax)
97  0076 4883C004           addq    $4, %rax
98                      .L9:
99                      .LVL6:
```

```
100                     .LBB14:
101                     .LBB15:
102  008a 8B13              movl    (%rbx), %edx
103  008c 31C0              xorl    %eax, %eax
104  008e BE000000          movl    $.LC2, %esi
105       00
106  0093 BF010000          movl    $1, %edi
107       00
108  0098 4883C304          addq    $4, %rbx
109  009c E8000000          call    __printf_chk
110       00
111                     .LVL7:
112                     .LBE15:
113                     .LBE14:
114  00a1 4839EB            cmpq    %rbp, %rbx
115  00a4 75E4              jne .L9
116  00a6 4883C468          addq    $104, %rsp
117                         .cfi_def_cfa_offset 24
118  00aa B8E50C00          movl    $3301, %eax
119       00
120  00af 5B                popq    %rbx
121                         .cfi_def_cfa_offset 16
122  00b0 5D                popq    %rbp
123                         .cfi_def_cfa_offset 8
124  00b1 C3                ret
125                         .cfi_endproc
126                     .LFE13:
127                         .text
128                     .Letext0:
```

Modern processors pipeline instructions. They predict which instruction is next and make optimisations based on assumptions of which order the instructions should be executed/ which branch they predicted.

A the end of a loop though, there are two possibilities- Either go back to the top, or continue on. The processor makes an educated guess on which is going to happen. If it gets it right, everything is good. If not, it has to flush the pipeline and stall for a bit while it prepares for taking the other branch. Unrolling a loop eliminates branches and the potential for those stalls, especially in cases where the odds are against a guess. Unrolling provides more benefits for shorter loops but ends up trashing performance if we are looping a large number of times. Usually, a smart compiler will take a decent guess about which loops to unroll but we can force it to a user defined value too. Loop unrolling does not work if the compiler can't predict the exact amount of iterations of the loop at compile time (or at least predict an upper bound, and then skip as many iterations as needed).

Also as can be seen in the attached screenshots, the **optimal unrolling factor for an array of size 16384 is 3**. Optimizations beyond level 3 have no further effect.

```
~109% [johng:~/git/hpc/A1] $ gcc -S -ftime-report vector.c

Execution times (seconds)
 phase setup             :   0.01 (33%) usr   0.00 ( 0%) sys   0.01 (25%) wall    1094 kB (62%) ggc
 phase parsing           :   0.01 (33%) usr   0.01 (50%) sys   0.02 (50%) wall     472 kB (27%) ggc
 phase opt and generate  :   0.01 (33%) usr   0.01 (50%) sys   0.01 (25%) wall     174 kB (10%) ggc
 preprocessing           :   0.00 ( 0%) usr   0.01 (50%) sys   0.01 (25%) wall     158 kB ( 9%) ggc
 lexical analysis        :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (25%) wall       0 kB ( 0%) ggc
 parser (global)         :   0.01 (33%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall     287 kB (16%) ggc
 integrated RA           :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (25%) wall      50 kB ( 3%) ggc
 shorten branches        :   0.00 ( 0%) usr   0.01 (50%) sys   0.00 ( 0%) wall       0 kB ( 0%) ggc
 final                   :   0.01 (33%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       5 kB ( 0%) ggc
 TOTAL                 :   0.03             0.02             0.04              1758 kB
~109% [johng:~/git/hpc/A1] $ gcc -S -ftime-report vector.c -O1
vector.c: In function ‘main’:
vector.c:10:11: warning: ignoring return value of ‘scanf’, declared with attribute warn_unused_result [-Wunused-result]
      scanf("%d",&x[i]);
           ^
vector.c:13:11: warning: ignoring return value of ‘scanf’, declared with attribute warn_unused_result [-Wunused-result]
      scanf("%d",&y[i]);
           ^

Execution times (seconds)
 phase setup             :   0.01 (25%) usr   0.00 ( 0%) sys   0.02 (29%) wall    1094 kB (52%) ggc
 phase parsing           :   0.01 (25%) usr   0.02 (100%) sys   0.03 (43%) wall     753 kB (36%) ggc
 phase opt and generate  :   0.02 (50%) usr   0.00 ( 0%) sys   0.02 (29%) wall     253 kB (12%) ggc
 preprocessing           :   0.01 (25%) usr   0.00 ( 0%) sys   0.02 (29%) wall     281 kB (13%) ggc
 lexical analysis        :   0.00 ( 0%) usr   0.02 (100%) sys   0.01 (14%) wall       0 kB ( 0%) ggc
 CSE                     :   0.01 (25%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       1 kB ( 0%) ggc
 loop init               :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (14%) wall      19 kB ( 1%) ggc
 branch prediction       :   0.01 (25%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       1 kB ( 0%) ggc
 rest of compilation     :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (14%) wall       5 kB ( 0%) ggc
 TOTAL                 :   0.04             0.02             0.07              2118 kB
~109% [johng:~/git/hpc/A1] $ gcc -S -ftime-report vector.c -O2
```

Figure 4: Loop Unrolling - 0 & 1

```
vector.c:13:11: warning: ignoring return value of ‘scanf’, declared with attribute warn_unused_result [-Wunused-result]
      scanf("%d",&y[i]);
           ^

Execution times (seconds)
 phase setup             :   0.01 (25%) usr   0.00 ( 0%) sys   0.02 (29%) wall    1094 kB (52%) ggc
 phase parsing           :   0.01 (25%) usr   0.02 (100%) sys   0.03 (43%) wall     753 kB (36%) ggc
 phase opt and generate  :   0.02 (50%) usr   0.00 ( 0%) sys   0.02 (29%) wall     253 kB (12%) ggc
 preprocessing           :   0.01 (25%) usr   0.00 ( 0%) sys   0.02 (29%) wall     281 kB (13%) ggc
 lexical analysis        :   0.00 ( 0%) usr   0.02 (100%) sys   0.01 (14%) wall       0 kB ( 0%) ggc
 CSE                     :   0.01 (25%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       1 kB ( 0%) ggc
 loop init               :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (14%) wall      19 kB ( 1%) ggc
 branch prediction       :   0.01 (25%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       1 kB ( 0%) ggc
 rest of compilation     :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (14%) wall       5 kB ( 0%) ggc
 TOTAL                 :   0.04             0.02             0.07              2118 kB
~109% [johng:~/git/hpc/A1] $ gcc -S -ftime-report vector.c -O2
vector.c: In function ‘main’:
vector.c:10:11: warning: ignoring return value of ‘scanf’, declared with attribute warn_unused_result [-Wunused-result]
      scanf("%d",&x[i]);
           ^
vector.c:13:11: warning: ignoring return value of ‘scanf’, declared with attribute warn_unused_result [-Wunused-result]
      scanf("%d",&y[i]);
           ^

Execution times (seconds)
 phase setup             :   0.01 (17%) usr   0.00 ( 0%) sys   0.01 (12%) wall    1094 kB (51%) ggc
 phase parsing           :   0.01 (17%) usr   0.02 (100%) sys   0.04 (50%) wall     753 kB (35%) ggc
 phase opt and generate  :   0.04 (67%) usr   0.00 ( 0%) sys   0.03 (38%) wall     291 kB (14%) ggc
 df reg dead/unused notes:   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (12%) wall       2 kB ( 0%) ggc
 preprocessing           :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.02 (25%) wall     281 kB (13%) ggc
 lexical analysis        :   0.01 (17%) usr   0.00 ( 0%) sys   0.02 (25%) wall       0 kB ( 0%) ggc
 parser (global)         :   0.00 ( 0%) usr   0.01 (50%) sys   0.00 ( 0%) wall     388 kB (18%) ggc
 parser inl. func. body  :   0.00 ( 0%) usr   0.01 (50%) sys   0.00 ( 0%) wall      57 kB ( 3%) ggc
 tree copy propagation   :   0.01 (17%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       2 kB ( 0%) ggc
 tree SSA other          :   0.01 (17%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       0 kB ( 0%) ggc
 tree PRE                :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (13%) wall       3 kB ( 0%) ggc
 forward prop            :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (13%) wall       1 kB ( 0%) ggc
 combiner                :   0.01 (17%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       2 kB ( 0%) ggc
 scheduling 2            :   0.01 (17%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       2 kB ( 0%) ggc
 TOTAL                 :   0.06             0.02             0.08              2156 kB
~109% [johng:~/git/hpc/A1] $
```

Figure 5: Loop Unrolling - 2

```
 phase opt and generate  :   0.04 (67%) usr   0.00 ( 0%) sys   0.03 (38%) wall     291 kB (14%) ggc
 df reg dead/unused notes:   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (12%) wall       2 kB ( 0%) ggc
 preprocessing           :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.02 (25%) wall     281 kB (13%) ggc
 lexical analysis        :   0.01 (17%) usr   0.00 ( 0%) sys   0.02 (25%) wall       0 kB ( 0%) ggc
 parser (global)         :   0.00 ( 0%) usr   0.01 (50%) sys   0.00 ( 0%) wall     388 kB (18%) ggc
 parser inl. func. body  :   0.00 ( 0%) usr   0.01 (50%) sys   0.00 ( 0%) wall      57 kB ( 3%) ggc
 tree copy propagation   :   0.01 (17%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       2 kB ( 0%) ggc
 tree SSA other          :   0.01 (17%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       0 kB ( 0%) ggc
 tree PRE                :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (13%) wall       3 kB ( 0%) ggc
 forward prop            :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (13%) wall       1 kB ( 0%) ggc
 combiner                :   0.01 (17%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       2 kB ( 0%) ggc
 scheduling 2            :   0.01 (17%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       2 kB ( 0%) ggc
 TOTAL                   :   0.06             0.02             0.08              2156 kB
~19% [johng:~/git/hpc/A1] $ gcc -S -ftime-report vector.c -O3
vector.c: In function ‘main’:
vector.c:10:11: warning: ignoring return value of ‘scanf’, declared with attribute warn_unused_result [-Wunused-result]
      scanf("%d",&x[i]);
           ^
vector.c:13:11: warning: ignoring return value of ‘scanf’, declared with attribute warn_unused_result [-Wunused-result]
      scanf("%d",&y[i]);
           ^

Execution times (seconds)
 phase setup             :   0.01 (12%) usr   0.00 ( 0%) sys   0.01 (10%) wall    1094 kB (45%) ggc
 phase parsing           :   0.02 (25%) usr   0.01 (50%) sys   0.03 (30%) wall     753 kB (31%) ggc
 phase opt and generate  :   0.05 (62%) usr   0.01 (50%) sys   0.06 (60%) wall     543 kB (23%) ggc
 df live regs            :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall       0 kB ( 0%) ggc
 alias analysis          :   0.01 (12%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall      28 kB ( 1%) ggc
 alias stmt walking      :   0.00 ( 0%) usr   0.01 (50%) sys   0.00 ( 0%) wall       0 kB ( 0%) ggc
 preprocessing           :   0.01 (12%) usr   0.00 ( 0%) sys   0.01 (10%) wall     281 kB (12%) ggc
 lexical analysis        :   0.00 ( 0%) usr   0.01 (50%) sys   0.00 ( 0%) wall       0 kB ( 0%) ggc
 parser (global)         :   0.01 (12%) usr   0.00 ( 0%) sys   0.01 (10%) wall     388 kB (16%) ggc
 parser inl. func. body  :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall      57 kB ( 2%) ggc
 dominator optimization  :   0.01 (13%) usr   0.00 ( 0%) sys   0.02 (20%) wall       2 kB ( 0%) ggc
 tree reassociation      :   0.01 (13%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       0 kB ( 0%) ggc
 CPROP                   :   0.01 (12%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       8 kB ( 0%) ggc
 integrated RA           :   0.01 (13%) usr   0.00 ( 0%) sys   0.01 (10%) wall      51 kB ( 2%) ggc
 reload CSE regs         :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall       9 kB ( 0%) ggc
 unaccounted todo        :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall       0 kB ( 0%) ggc
 TOTAL                   :   0.08             0.02             0.10              2408 kB
~19% [johng:~/git/hpc/A1] $
```

Figure 6: Loop Unrolling - 3

```
 preprocessing           :   0.01 (12%) usr   0.00 ( 0%) sys   0.01 (10%) wall     281 kB (12%) ggc
 lexical analysis        :   0.00 ( 0%) usr   0.01 (50%) sys   0.00 ( 0%) wall       0 kB ( 0%) ggc
 parser (global)         :   0.01 (12%) usr   0.00 ( 0%) sys   0.01 (10%) wall     388 kB (16%) ggc
 parser inl. func. body  :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall      57 kB ( 2%) ggc
 dominator optimization  :   0.01 (13%) usr   0.00 ( 0%) sys   0.02 (20%) wall       2 kB ( 0%) ggc
 tree reassociation      :   0.01 (13%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       0 kB ( 0%) ggc
 CPROP                   :   0.01 (12%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       8 kB ( 0%) ggc
 integrated RA           :   0.01 (13%) usr   0.00 ( 0%) sys   0.01 (10%) wall      51 kB ( 2%) ggc
 reload CSE regs         :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall       9 kB ( 0%) ggc
 unaccounted todo        :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall       0 kB ( 0%) ggc
 TOTAL                   :   0.08             0.02             0.10              2408 kB
~19% [johng:~/git/hpc/A1] $ gcc -S -ftime-report vector.c -O4
vector.c: In function ‘main’:
vector.c:10:11: warning: ignoring return value of ‘scanf’, declared with attribute warn_unused_result [-Wunused-result]
      scanf("%d",&x[i]);
           ^
vector.c:13:11: warning: ignoring return value of ‘scanf’, declared with attribute warn_unused_result [-Wunused-result]
      scanf("%d",&y[i]);
           ^

Execution times (seconds)
 phase setup             :   0.01 (11%) usr   0.00 ( 0%) sys   0.02 (20%) wall    1094 kB (45%) ggc
 phase parsing           :   0.03 (33%) usr   0.01 (100%) sys   0.03 (30%) wall     753 kB (31%) ggc
 phase opt and generate  :   0.05 (56%) usr   0.00 ( 0%) sys   0.05 (50%) wall     543 kB (23%) ggc
 preprocessing           :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.02 (20%) wall     281 kB (12%) ggc
 lexical analysis        :   0.01 (11%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       0 kB ( 0%) ggc
 parser (global)         :   0.01 (11%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall     388 kB (16%) ggc
 parser function body    :   0.00 ( 0%) usr   0.01 (100%) sys   0.01 (10%) wall      12 kB ( 1%) ggc
 parser inl. func. body  :   0.01 (11%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall      57 kB ( 2%) ggc
 tree PTA                :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall       1 kB ( 0%) ggc
 tree CCP                :   0.01 (11%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall      22 kB ( 1%) ggc
 tree iv optimization    :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall      37 kB ( 2%) ggc
 expand                  :   0.01 (11%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall      50 kB ( 2%) ggc
 CSE                     :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall       2 kB ( 0%) ggc
 CPROP                   :   0.01 (11%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       8 kB ( 0%) ggc
 combiner                :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall      16 kB ( 1%) ggc
 integrated RA           :   0.01 (11%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall      51 kB ( 2%) ggc
 LRA reload inheritance  :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 (10%) wall       0 kB ( 0%) ggc
 scheduling 2            :   0.01 (11%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall       3 kB ( 0%) ggc
 TOTAL                   :   0.09             0.01             0.10              2408 kB
~19% [johng:~/git/hpc/A1] $
```

Figure 7: Loop Unrolling - 4