Name Anamo Kisho

1. Convert -6.625 to 32-bit IEEE single precision format

Step-1 – negative

2, bits

6.625 = 110.101

3, normalize

1.10101 x 2^2

4, add 127 to exponent 2

2 + 127 = 129 in binary : 10000001

5, Write out the bits:-

11000000110101000000000000000000

= 0xC0D40000

1. Convert 123.457 to 32 32-bit IEEE single precision format.

Step-1 – positive

2, bits

123.457 = 01111011.01110100111

3, normalize

1.11101101110100111 x 2 ^6

4, add 127 to exponent 6

6 + 127 = 133 in binary: 10000101

5, Write out the bits:-

01000010111101101110100111000000

= 0x42F6E9C0

1. What floating-point number is represented by **0x41BA0000**.

1, convert hex to binary (**0x41BA0000)**

**= 01000001101110100000000000000000**

2, The first number is sign = 0 --- positive

3, The next 8 bits are the exponent = **10000011 to dec --- 131**

Sub 127 to find the exponent 131 – 127 = 4

4, Next the remaining bits is the mantisa

1.0111010 to 10111.01

5, find the dec

== 23.25

1. Assume we are multiplying the unsigned integers **1011 X 1011**. Trace the values of the multiplicand, multiplier, and result at every step. (We are not covering this algorithm until Monday April 11).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | Initial Values | 1011 | 0000 1011 | 0000 0000 |
| 1 | 1 => Product = Product + Multiplicand  Shift left Multiplicand  Shift Right Multiplier | 1011  1011  0101 | 0000 1011  0001 0011  0001 0011 | 0000 1011  0000 1011  0000 1011 |
| 2 | 1 => Product = Product + Multiplicand  Shift left Multiplicand  Shift Right Multiplier | 0101  0101  0010 | 0001 0011  0010 0110  0010 0110 | 0001 1011  0001 1011  0001 1011 |
| 3 | 1 => Product = Product + Multiplicad  Shift left Multiplicand  Shift right Multiplier | 0010  0010  0001 | 0010 0110  0100 1100  0100 1100 | 0001 1011  0001 1011  0001 1011 |
| 4 | 1 => Product = Product + Multiplicad  Shift left Multiplicand  Shift right Multiplier | 0001  0001  0000 | 0100 1100  1001 1000  1001 1000 | 0001 1011  0001 1011  0001 1011 |
| 5 | 0 => No Operation  Shift left Multiplicand  Shift right Multiplier | 0000  0000  0000 | 1001 1000  0011 0000  0011 0000 | 0001 1011  0001 1011  0001 1011 |
| 6 | 0 => No Operation  Shift left Multiplicand  Shift right Multiplier | 0000  0000  000**0** | 0011 0000  0110 0000  0110 0000 | 0001 1011  0001 1011  0001 1011 |
| 7 | 0 => No Operation  Shift left Multiplicand  Shift right Multiplier | 0000  0000  000**0** | 0110 0000  1100 0000  1100 0000 | 0001 1011  0001 1011  0001 1011 |
| 8 | 0 => No Operation  Shift left Multiplicand  Shift right Multiplier | 0000  0000  0000 | 0110 0000  1000 0000  1000 0000 | 0001 1011  0001 1011  0001 1011 |
| 9 | 0 => No Operation  Shift left Multiplicand  Shift right Multiplier | 0000  0000  0000 | 0110 0000  1000 0000  0000 0000 | 0001 1011  0001 1011  0001 1011 |

1. The swap function below exchanges the two double values pointed to by **x** and **y**. Write **swap** as an ARM assembly language function. Full credit for the most concise version.

**void swap(double \*x, double \*y) {**

**double tmp = \*x;**

**\*x = \*y;**

**\*y = tmp;**

**}**

ARM assembly code:

**global swap**

**.fpu vfp // vector floating-point unit**

**.cpu cortex-a53**

**.text**

**swap:**

**vldr.f64 d0,[r0]**

**vldr.f64 d1,[r1]**

**vstr.f64 d0,[r1]**

**vstr.f64 d1,[r0]**

1. Write a recursive C function that implements the declaration below. **popcount** counts the number one bits in the binary representation of its argument. For example, **popcount(30)** is 4 because 30 in binary is 11110, which has four one bits.

**extern int popcount(unsigned int n);**

**#include <stdio.h>**

**int popcount(int n) {**

**// base case**

**if(n==0) {**

**return 0;**

**}**

**if((n & 1) == 1){**

**return 1 + popcount(n >> 1);**

**}**

**return popcount(n>>1) ;**

**}**

1. Write **popcount** as an ARM assembly language function.

\*/

.global **popcount**

.cpu cortex-a53

.text

**popcount**:

push {r4,lr}

mov r1,#0

mov r2,#1

mov r4,#2

if:

cmp r0,r1

bne endif1

mov r0, #0

bx lr

endif1:

and r3,r0,#1 //if(n&1)

cmp r3,r2

bne return

lsr r0,r0,r2

bl **popcount**

add r0,r0,r2

pop {r4,lr}

bx lr

return:

lsr r0,r0,r2

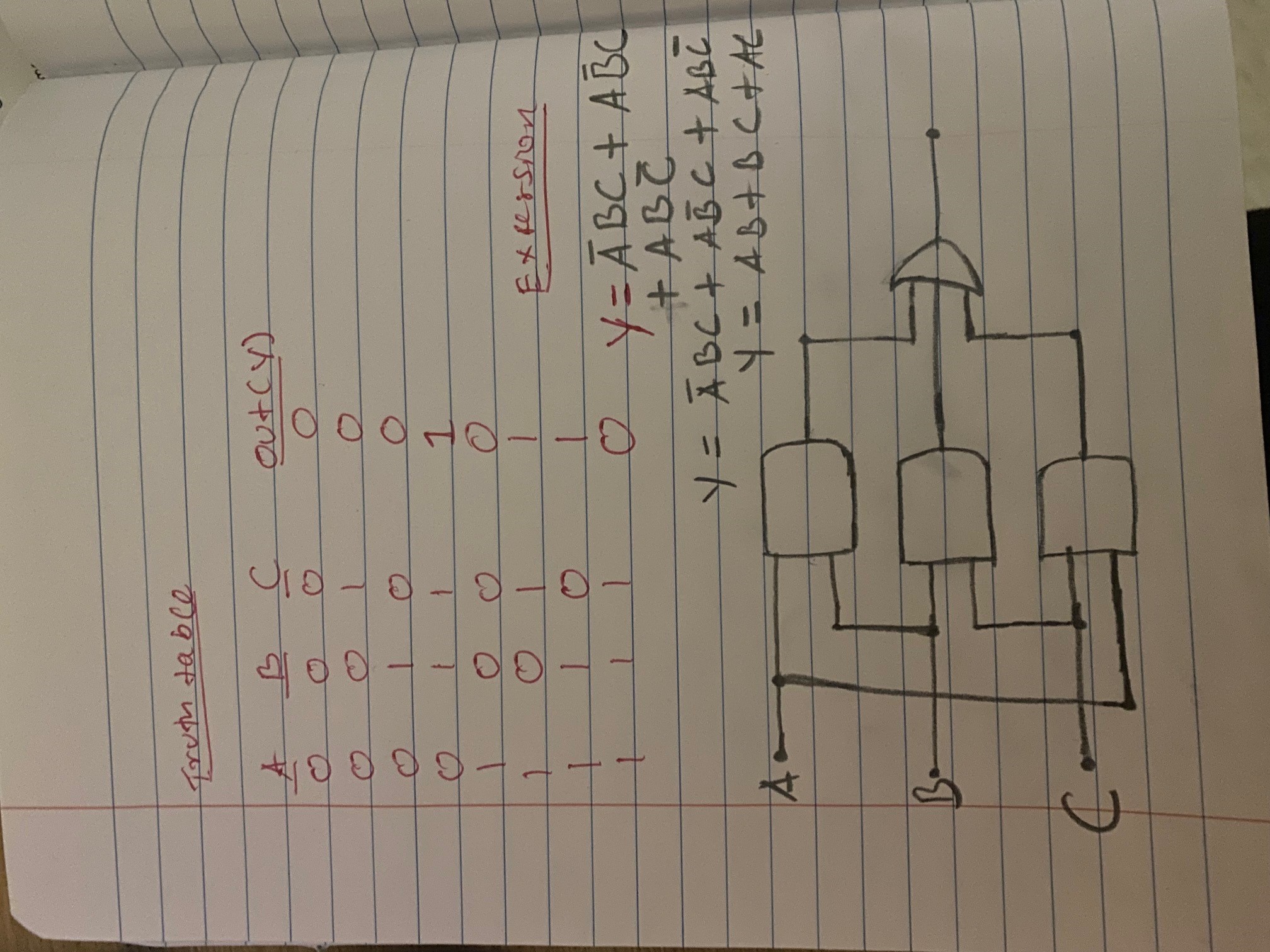
bl **popcount**

sub r0,r0,#1

pop {r4,lr}

bx lr

1. Consider the logic function with three inputs **A**, **B**, **C** and one output **Out**. **Out** should be 1 when exactly two inputs are 1.
   1. Draw the truth table for this function.
   2. Write the sum-of-products logic equation for this function.
   3. Minimize the logic equations
   4. Draw the circuit diagram for the logic equation.



1. Write a C function scale that takes a factor and multiplies each item in the array by the factor.

**extern void scale(double factor, double [] vec, int n);**

void int2bin(double factor, double vec[], int n) {

for (int i=0;i<n;i++) {

vec[i] = vec[i]\*factor ;

}

}

1. Make sure you understand the four areas of program memory; code, global data, stack, and heap and how memory is allocated for each.
2. Static function local variables in C are allocated in/on \_\_\_\_\_ global data\_\_\_\_\_\_\_\_ memory.
3. Local variables in C are allocated in/on \_\_stack\_\_\_\_\_\_ memory.
4. Memory allocated using **malloc** is \_\_heap\_\_\_\_\_ memory.
5. What does the **-g** flag on the gcc compiler do?

It generate debugging information

1. What does the **-S** flag on the gcc compiler do?

Stop after the stage of compilation proper; do not assemble. The output is in the form of an assembler code file for each non-assembler input file specified.

1. What does the **-o** flag on the gcc compiler do?

file flag, gives the executable file a memorable name

1. What does the **-O3** flag on the gcc compiler do?

‘-O3’ turns on all optimizations specified by ‘-O2’ and also to discover which binary optimizations are enabled

1. What does the **-c** flag on the gcc compiler do?

Compile or assemble the source files, but do not link. The linking stage simply is not done. The ultimate output is in the form of an object file for each source file.

1. What program do we use to reverse engineer machine code files?

IDA Pro, Hex Rays or java

1. How many bytes is a C **double**?

Its 8 bytes

1. Briefly describe what a *memory leak* is?

memory leak occurs, when a piece of memory which was previously allocated by the programmer. Then it is not deallocated properly by programmer memory leak is caused by forgetting to free memory

1. Consider the following C program. Why might it have a segmentation fault?

**#include <stdio.h>**

**int \*seven() {**

**int x = 7;**

**return &x;**

**}**

**int main() {**

**int \*y = seven();**

**printf("%d\n", \*y);**

**}**

Because x is an int and &x the return is not the same it is a pointer, so it cause segmentation fault

1. The following variation of the program seems to work OK. Why?

**#include <stdio.h>**

**int \*seven() {**

**static int x = 7;**

**return &x;**

**}**

**int main() {**

**int \*y = seven();**

**printf("%d\n", \*y);**

**}**

Because now our x is an static int remains in memory while the program is running

1. Write a function **rev** that takes an unsigned integer **x** and reverses the bits in **x**. Use bit operations only, don’t use strings or arrays.
   1. Modify the **add** function in **adder.c** we wrote to call **rev**.

typedef u\_int32\_t ;

u\_int32\_t con = 0;

**while** (n > 0)

        {

            con <<= 1;

**if** ((u\_int32\_t)(n & 1) ==1)

rev ^= 1;

            n >>= 1;

        }

**return** rev;

    }

1. There is a simple fix to the **add** function in **adder.c** file that does not need to reverse the bits. What is it?

No, there is not