Floating Point
1. Finna Bias
Bias = 2^(e-1)-1
2. log2^(Nefnari) = Fjöldi færsla
0, _*(Fjöldi færsla)
Setur Teljara í Binary á strikin (Lengst til vinstri)
Færir kommuna að fyrsta ás(1)
e = hversu oft þú færir kommuna að ás
3. DeNormalized eða Normalized?
Ef e >= Bias þá er talan DeNormalized else: Normalized

DeNormalized
4. Finna f og E
E = 1 - bias
Færa kommu um gildi á E
f = Teljarinn í bin + 0 til að fylla í fraction bita

5. Setja búa til tölu
s = 0 ef talan er jákvæð
s = 1 ef talan er neikvæð
e = 0 * e
f = f

Normalized
4. Finna f
f = binary eftir kommuna og fyllir bitana með 0

5. Finna E
E = e - Bias

6. Finna e
E í binary og fyllir bitana með 0

7. Setja tölurnar saman
s = 0 ef talan er jákvæð
s = 1 ef talan er neikvæð
e = 6. liður
f = 4. liður

Reverse Floating Point
Dæmi Fyrir Normalized:
1.
s exp frac
16 8 4 2 1 1/2 1/4 1/8 1/16 1/32 1/64
0 0 0 0 1 1 0 1 1 0 0 0

2. Finna Bias
Bias = 2^(e-1)-1
Bias = 2^(5-1)-1 = 15

3. Finna E
e = (exp to Binary) = 3
E = e - Bias
E = 3 - 15 = -12

4. Finna M
M = þar sem ás er fyrir neðan + 1
M = 1/4 + 1/8 + 1 = 11/8

5. Formúla
V = (-1)^s * M * 2^E
V = (-1)^0 * 11/8 * 2^(-12)
V = 11/32768

Dæmi Fyrir DeNormalized
1.
s exp frac
16 8 4 2 1 1/2 1/4 1/8 1/16 1/32 1/64
0 0 0 0 0 0 1 0 0 0 1 0

2. Finna Bias
Bias = 2^(e-1)-1
Bias = 2^(5-1)-1 = 15

3. Finna E
e = 1 (Alltaf 1)
E = 1 - Bias
E = 1 - 15 = -14

4. Finna M
M = þar sem ás er fyrir neðan + 1
M = 1/2 + 1/32 = 17/32

5. Formúla
V = (-1)^s * M * 2^E
V = (-1)^0 * 17/32 * 2^(-14)
V = 17/524288
# Special
-Infinity > 1 11111 000000
+Infinity > 0 11111 000000
NaN > 1 11111 111111
Biggest + denorm > 0 00000 111111
Smallest + norm > 1 00001 000000

Linux commands
Basics

HEX	DECIMAL	BINARY
0XA	10	1010
0XB	11	1011
0XC	12	1100
0XD	13	1101
0XE	14	1110
0XF	15	1111

Echo string
```console
notandi@skel:~$ echo <<string>>
```

Print contents of a directory
```console
notandi@skel:~$ ls
```

Go to another directory
```console
notandi@skel:~$ cd <<directory>>
```

Print working directory
```console
notandi@skel:~$ pwd
```

Print the contents of a file to the screen
```console
notandi@skel:~$ cat <<filename>>
```

Print first few lines of a file
```console
notandi@skel:~$ head <<filename>>
```

Prints last 10 lines
```console
notandi@skel:~$ tail <<filename>>
```

Make folder
```console
notandi@skel:~$ mkdir <<filename>>
```

Make file
```console
notandi@skel:~$ touch <<filename>>
```

Copy file
```console
notandi@skel:~$ cp <<filename>> <<destination>>
```

Copy many files (Example: file1.txt file2.txt file3.txt file4.txt)
```console
notandi@skel:~$ cp <<filename{1..4}.txt>> <<destination>>
```

Copy folder
```console
notandi@skel:~$ cp -R <<filename>> <<destination>>
```

Rename file
```console
notandi@skel:~$ mv <<old filename>> <<new filename>>
```

Delete file
```console
notandi@skel:~$ rm <<filename>>
```

Delete folder (!!WARNING!!)
```console
notandi@skel:~$ rm -dr <<folder>>
```

Count lines in file
```console
notandi@skel:~$ wc -l <<filename>>
24349 apache.log
```

Count lines in file and export to new file
```console
notandi@skel:~$ wc -l <<filename>> > <<new filename>>
24349 apache.log
```

Select specific lines that contains specific word in file
```console
notandi@skel:~$ grep '<<word>>' <<filename>>
```
grep -v :5 building_access.txt | grep mackenzie | cut -d ':' -f 2

Select specific lines that contains specific word in file and has special case
```console
notandi@skel:~$ grep '\*WARNING\* user...' <<filename>>
```

Select specific lines that contains specific word in file and export to new file
```console
notandi@skel:~$ grep '<<word>>' <<filename>> > <<new filename>>
```

Select specific lines that contains specific word in files inside folder
```console
notandi@skel:~$ grep -r '<<word>>' <<filename>>
```

Select specific lines that contains specific word in files inside folder and export to new file
```console
notandi@skel:~$ grep -r '<<word>>' <<filename>> > <<new filename>>
```

Select specific lines that contains specific word in file ordered
```console
notandi@skel:~$ grep '<<word>>' <<filename>> | sort
```

Select specific lines that contains specific word in files inside folder and export to new file ordered
```console
notandi@skel:~$ grep -r '<<word>>' <<filename>> sort > <<new filename>>
```

Remove lines that contain specific string in file
```console
notandi@skel:~$ sed '/<<string>>/d' <<filename>>
```

Remove lines that contain specific string in file and export to new file
```console
notandi@skel:~$ sed '/<<string>>/d' <<filename>> > <<new filename>>
```

Alphabetically sorted list of specific string with no duplicates.
```console
notandi@skel:~$ grep '<<string>>' <<filename>> | sort | uniq > <<new filename>>
```

Make script executable
```console
notandi@skel:~$ chmod +x <<filename>>
```

Create a script that only prints out the specific string from file
```console
notandi@skel:~$ vim <<filename>>
 #!/bin/bash
 grep '<<string>>' <<filename>> | cut -d '<<char that you want to cut on>>' -f9 | sort | uniq
```

# Binary Operators
a = 1100 1010 b = 1010 1110
& = And, 1 & 1 = 1, 0 & 1 = 0, 0 & 0 = 0
| = Or, 1 | 1 = 1, 1 | 0 = 1, 0 | 0 = 0
^ = Xor, 1 ^ 1 = 0, 1 ^ 0 = 1, 0 ^ 0 = 0
~ = Not, ~ 1 = 0, ~ 0 = 1

Logical Operators
&& = And, 0010 && 0011 = 1, 0010 & 0000 = 0,
|| = Or, 1010 || 0000 = 1, 0000 || 0000 = 0, 1111 || 1010 = 1
! = Not, ! 1100 = 0, ! 0000 = 1
>> = Right Shift, 0100 >> 1 = 0010, 0100 >> 2 = 0001, Ath signed/unsigned til að fylla upp í bita v.m.
<< = Left shift, 0100 << 1 = 1000, 0011 << 2 = 1100, Ath getur orðið overflow ef bitafjöldinn er restricted.

Maskar
Gott til að finna fyrstu ákveðna bita. Þá er notað & virkjann
Ef við viljum finna bita nr. 5-8 í bitastreng er hægt að gera svona:
Maski = 1111 0000, Tala = 1010 1111, Maski & Tala = 1010 0000
Svo er hægt að shifta ef beðið er um að setja þá fremst eða ehv.

Two's Complement:
Samlagning: 0+0 = 0, 1+0 = 1, 1+1 = 0 og 1 færist á næsta til vin…
Finna mínustölu: Flippa bitum (~) og bæta einum við.
Tmax = signed biti er 0 og allt annað 1.
Tmin = signed biti er 1 og allt annað 0.
Stærð Datatypes: 1 byte = 8 bitar
Char = 1 byte --- Short = 2 bytes --- Int = 4 bytes --- Long = 8 bytes
# Floating Points:
Normalized: exp blanda af 1 og 0. DeNormalized: exp bara 0. Special: exp bara 1.

Arithmetic op.
Lea_ <source>,<dest> //++*
Addq //d+s
Imulq //d-s
Salq//d<<s 2**s
Sarq//d>>s
Xorq//d^s
Andq//d&s
Orq//d|s

Reg	Arguments
%rsp	Stack pointer
%rax	Return value
%rdi	1st argument
%rsi	2nd argument
%rdx	3rd argument
%rcx	4th argument
%r8	5th argument
%r9	6th argument

Memory op
Mov_//move
Imm(regb, regi, s)//a+b+i*s
Imm =immed. Offset
Regb = base register
Regi = index register
S = scale factor

# Assembly
Push = bæta ofan á
Pop = taka efsta úr
2 – var types
a) Var storing data
b) Var storing mem address
Var storing mem address = pointer

# Registers
rsp = stack pointer
rax = ret value
rdi = 1stargument
rsi = 2ndargument
rdx = 3rdarg
rcx = 4th
r8 = 5th
r9 = 6th

#Bits&bytes
dec -> hex reiknirit evklíðs
hex ->dec aðferð snúið við

Logical operators
(allur bitinn tekinn)
|| (OR) – annað hvort T = T
&&(AND) – bæði T = T
! (Not) t->f / f->t
allir bitar 0 = false
amk einn biti 1 = T

Logical shift
left shift a<<x = a*2^x
right shift a>>x = a/2^x

Bitwise - hver biti
&(and) - báðir t = T
|(or) - annað hvort t=T
^(xor) t-f = T annars F
~(not) t-f/f-t

Masking
Masking to 1
to turn on, bitwise |(or)1
leave unchanged |(or)0
Turning bit to 0, bitwise & 0

Two's complement
setja upp í töflu = fjöldi bita
fyrsta tala = -tala
athuga hversu oft þarf að plúsa við
Tmax = setja alla bita 1 nema fyrsta
(2^k-1)-1
Tmin = allir 0 nema fyrsti 1
(-2^k-1)
Tmax+tmin = -1 / allir 1
Tmax + 1 = tmin
bin í -bin -> flippa bitum og +1

Floating Point - decimal -> binary
1. Finna bias=(2^k-1)-1
2. Tala í bin brot eða log2(nefnari
3. Skrifa tölu sem 1.frac*2^y (y=færslur á kommu)
ef y < bias = normal
y>=bias = de-norm

Normalized
4. finna frac&E/frac = sjá skref 3
E = y
5. Finna e = E + bias
6. Breyta gildi e í binary

De-normalized
4. finna frac = skref 3 öll talan(1.frac)
Finna E = 1-bias // færa kommu um gildi á E
5. e = 0 því talan er denorm

Dec - floating point
de-norm
1. Finna bias
2. Finna E = 1-bias
3. Reikna dec value af broti
4. Reikna M/M=f
5. Reikna dec value
dec val = (-1)^s * M * 2^E

Normalized
1. Finna bias
2. Finna E = e-bias
3. Reikna dec value af broti
4. Reikna M/M=1+frac
5. Reikna dec value
dec val = (-1)^s * M * 2^E

Hvað er hvað í assembly?
1. Ef það er $ fyrir framan þá er það tala sem er sitt eigið value.
2. Ef það er % fyrir framan þá er það register. Sækir value í registeruna.
3. Ef það er ekkert fyrir framan eða á formattinu a (b, c, d) þá er það memory address. Sækir value í memory addressuna.

Assembly Skipanir:
mov_ <source>, <destination> Move source to destination
add_ <source>, <destination> d = d + s
sub_ <source>, <destination> d = d - s
imul_ <source>, <destination> d = d x s

sal_ <source>, <destination> d = d << s
sar_ <source>, <destination> d = d >> s
shl_ <source>, <destination> d = d << s
shr_ <source>, <destination> d = d >> s

xor_ <source>, <destination> d = d ^ s
and_ <source>, <destination> d = d & s
or_ <source>, <destination> d = d | s

lea_ <source>, <dest…> source: memory, destination: register.
push_ <source>, <dest…> source: constant, register or memory.
pop_ <source>, <dest…> source: register or memory
mov_ <sou..>, <dest.> sou: const, regi or
mem, dest: reg or mem
Gögn úr sou.. færð yfir í dest.., má ekki vera bæði memory add..
Source sækir gögn í memory add.. eða registerið.

test_ <arg1> <arg2> test if neg or not
cmp_ <arg1> <arg2> compares the arguments
lea_ <source>, <destination> Load effective address of source into destination

Instruction	Synonym	Set condition
je L	jz L	Equal / zero
jne L	jnz L	Not equal / not zero
js L		Negative
jns L		Not negative
jg L	jnle L	Greater >
jge L	jnl L	Greater or equal >=
jl L	jnge L	Less <
jle L	jng L	Less or equal <=
ja L	jnbe L	Above >
jae L	jnb L	Above or equal >=
jb L	jnae L	Below <
jbe L	jna L	Below or equal <=

Tmax = 0111 1111
Tmin = 1000 0000

```
int x[ 5 ] = {0, 1, 2, 3, 4}; // This array is at address 0x08048c00
short y[ 3 ] = {0, 1, 2}; // This array is at address 0x08048c14
```

```
a = x[ 0 ];
b = x;
c = &x[ 0 ];
d = x + a + 3;
e = *d + 3;
f = (x + 2)[ 2 ];
g = &y[ 2 ] - 2;
h = *(&y + 1);
i = *(x + 5);
```

Fill in the following table. Give your answers in hexadecimal. Denote any unknown variables by ?.

Expression	Type	Value
a	int	0x0
b	int*	0x08048c00
c	int*	0x08048c00
d	int*	0x08048c0c
e	int	0x6
f	int	0x4
g	short*	0x08048c14
h	short*	0x8048c1a
i	int	0x10000

Consider the source code below, where M and N are constants declared with #define.

```
#define M (secret)
#define N (secret)

int mat1[ M ][ N ];
int mat2[ N ][ M ];

void copy_element(int i, int j)
{

    mat1[ i ][ j ] = mat2[ j ][ i ];
}
```

This generates the following assembly code:

```
copy_element:
    movslq    %edi, %rdi
    movslq    %esi, %rsi
    leaq    (%rsi,%rsi,2), %rax
    leaq    (%rax,%rax,8), %rax
    addq    %rdi, %rax
    movl    mat2(,%rax,4), %edx
    leaq    (%rsi,%rdi,8), %rax
    movl    %edx, mat1(,%rax,4)
    ret
```

What are the values of N and M?

N: 8

M: 27

Parameter	Description
Fundamental parameters	
$S = 2^s$	Number of sets
E	Number of lines per set
$B = 2^b$	Block size (bytes)
$m = \log_2(M)$	Number of physical (main memory) address bits
Derived quantities	
$M = 2^m$	Maximum number of unique memory addresses
$s = \log_2(S)$	Number of set index bits
$b = \log_2(B)$	Number of block offset bits
$t = m - (s + b)$	Number of tag bits
$C = B \times E \times S$	Cache size(bytes)

```
// The array x starts at memory address 0x08048000
int x[ 5 ] = { 0x5, 0x20, 0x40, 0x80, 0x100 };
```

```
a = x[ 1 ];
b = &x[ 0 ];
c = x[ 2 ] + 16;
d = x + 2;
e = *(x + x[ 0 ]);
f = &x[ 2 ] + 2;
```

a	int	0x20
b	int*	0x08048000
c	int	0x50
d	int*	0x08048008
e	int	?
f	int*	0x08048010

GCC generates the following assembly code for the C function fun

```
fun:
        movl    %edi, %eax
        jmp     .L2
.L4:
        subl    $1, %esi
.L2:
        testl   %eax, %eax
        jle     .L3
        cmpl    %esi, %eax
        jle     .L4
        subl    %esi, %eax
        jmp     .L2
.L3:
        ret
```

Fill in the missing expressions in the following C code

```
int fun(int a, int b)
{
    while(___a>0_____)
    {
        if(____b<a____)
        {
            ______a_____ = ____a - b___;
        }
        else
        {
            ______b_____ = _____b-1____;
        }
    }
    return____value__;
}
```

```
fun:
   movl %esi, %eax
   cmpl $5, %edi
   jle .L2
   addl $1, %edi
   cmpl $9, %edi
   jg .L2
   movl %edi, %eax
.L2:
   ret
int fun(int a, int b)
{
   int value = #b;
   if ( #a > 5 )
   {
      #b = #a + 1 ;
      if ( #b < 10)
      {
         value = #b;
      }
```

Floating Points:
Normalized: exp blanda af 1 og 0. DeNormalized: exp bara 0. Special: exp bara 1.
Decimal Binary
S exp – k bitar frac.
1. Bias = 2^((k-1)) -1
2. Breyta fraction í binary:
… 4 2 1 . 1 / 2 1 / 4 ...
3. Skrifa fraction binary-ið sem 1.frac *2^y, y er hversu mörg sæti þurfti að hliðra til að fá 1.frac --- Hliðra til vinstri: Y > 0.
4. Ef |y| < Bias Normalized tala.
Ef |y| ≥ Bias DeNormalized tala.
5(Normalized). E = y, frac = frac úr 1.frac í skrefi 3.
5(DeNormalized). E = 1 – Bias, frac = binary fractionið úr skrefi 2 hliðrað um E (1 – Bias)sæti. frac er fyrir aftan punkt í því.
6(Normalized). e = E + Bias.
6(DeNormalized). Skippa.
7(Normalized). exp er binary gildi e.
8(DeNormalized). exp er allt 0
9. Input tala er mínustala þá 1, annars 0.
Setja svo inn í S-exp-frac.
Binary Decimal
1. Bias = 2^((k-1)) -1
2. Ef exp: Allt 0. E = 1 – Bias
Ef exp: 010…. E = e – Bias, e = binary gildi exp
Ef exp: Allt 1. Special Value
3. f = decimal gildi frac.
4. Ef exp: Allt 0. M = f
Ef exp: 010…. M = 1 + f
5. S = signed bitinn.
6. Reikna út decimal gildið: Setja inn í þessa jöfnu:
V=(-1)^((S))*M*2^((E))
Special values: exp is all ones.
NaN (Not a number) ef frac hlutinn er ekki bara 0.
Infinity ef frac er allur 0. Getur verið + eða – fer eftir S-Bitanum.
Stærð fraction talna:
Stærsti Normalized 4 exp bitar væri: 1110 og stærsti 4 frac bitar væri 1111.
Stærsti DeNormalized 4 exp bitar væri 0000, má ekki annað. Stærsti 4 frac bitar væri 1111.
Ef námunda, námunda venjulega nema .5 fer í næstu even num.

Expression	Decimal Representation	Binary Representation
(int)0	0	0000 0000
(short)0	0	0000
(int)-23	-23	1110 1001
(int)	69	0100 0101
(int)-18	-18	1110 1110
(short)(int)-18	-2	1110
(int)(short)(int)-18	-2	1111 1110
(unsigned int)(int)-18	238	1110 1110
(unsigned int)(-5 + 1)	252	1111 1100
(int)TMax + (int)TMin	-1	1111 1111
(int)((1<<5) >> 3)	4	0000 0100

4-way Set Associative Cache																
Sets	Valid	Tag	Byte 0	Byte 1	Valid	Tag	Byte 0	Byte 1	Valid	Tag	Byte 0	Byte 1	Valid	Tag	Byte 0	Byte 1
0	0	4EF	13	B9	1	572	01	75	0	025	C7	73	0	70A	1A	DA
1	0	28E	3D	62	0	1AE	BA	DB	1	3B3	E5	81	1	235	96	5C
2	1	172	60	4E	1	45D	6F	83	0	5D9	DF	26	1	698	45	D5
3	1	590	CF	6B	1	7CF	2D	B4	1	129	BB	9D	1	379	00	8D
4	1	10A	79	E2	0	5F0	25	2E	1	20D	FC	66	0	278	EC	5E
5	1	644	83	4C	1	025	B5	D8	0	4BD	6D	69	1	242	9A	5B
6	1	679	1B	69	1	321	49	0F	1	2D8	05	91	1	1A0	D4	C9
7	1	385	5B	14	0	7D0	AB	AC	1	448	D2	13	0	6D2	D4	D9

Consider the following cache problem.

You may assume the following:

- The memory is byte addressable.
- Memory accesses are to 1-byte words (not 4-byte words).
- Physical addresses are 15 bits wide.
- The cache is 4-way set associative, with a 2-byte block size and has 8 sets, with 32 total lines.

In the following tables, all numbers are given in hexadecimal. The contents of the cache are as follows:

The box below shows the format of a physical address.

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
t	t	t	t	t	t	t	t	t	t	t	s	s	s	b

For the given physical address, indicate the cache entry accessed and the cache byte value returned in hex. Indicate whether a cache miss occurs.

If there is a cache miss, enter "-" for "Cache Byte returned".

Physical address: 0x5907

Parameter	Value
Cache block offset	0x 1
Cache set index	0x 3
Cache tag	0x 590
Cache hit? (Y/N)	Y
Cache byte returned	0x 6B

Physical address: 0x2d8c

Parameter	Value
Cache block offset	0x 0
Cache set index	0x 6
Cache tag	0x 2D8
Cache hit? (Y/N)	Y
Cache byte returned	0x 05