Num	Dec	Binary
TMax	15	0 1111
TMin	-16	1 0000
TMin+TMin	0	0 0000
TMin+1	-15	1 0001
TMax+1	-16	1 0000
−TMax	-15	1 0001
−TMin	-16	1 0000

```
#!/bin/bash
echo “Hello“
#EOF
```

Multiple choice

What is the C equivalent of
leal 0x10(%eax,%ecx,4),%edx

Svar : edx = 0x10 + eax + ecx*4

Consider an int *a and an int n. If the value of %ecx is a and the value of %edx is n, which of the following assembly snippets best corresponds to the C statement return a[n]?

Svar : mov (%ecx,%edx,4),%eax ret

The x86/IA32 instruction test is best described as which of the following:

Svar : Same as and, but doesnt keep the result (only sets flags)

On a 32-bit Linux system, what is the size of a long?

Svar : 4 bytes

Consider the C declaration
short array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
Suppose that the compiler has placed the variable array in the %ecx register. How do you move the value at array[5] into the %eax register? Assume that %ebx is 5.

Svar : movl (%ecx,%ebx,2),%eax

What is the minimum (most negative) value of a 32-bit two’s complement integer?

Svar : −2^31

Assume a function foo takes two arguments. When calling foo(arg1, arg2), which is the correct order of operations assuming x86 calling conventions and that foo must allocate stack space (implies that we must save the %ebp)?

Svar: push arg2, push arg1, call foo, push %ebp

Let int x = -31/8 and int y = -31 >> 3. What are the values of x and y?

Svar: x = -3,y = -4

test %eax, %eax
jne 3d<function+0x3d>
Which of the following values of %eax would cause the jump to be taken?

Svar: 1

The TEST operation performs a bit-wise logial AND of the two operands. The result of a bit-wise logical AND is 1 if the value of that bit in both operands is 1; otherwise, the result is 0. Test discards the results and modifies the flags. The OF and CF flags are cleared; SF, ZF and PF flags are set according to the result.

```
; Conditional Jump
test cl, cl         // set ZF to 1 if cl == 0
je 0x804f430        // jump if ZF ==1
; or
test eax,eax        // set SF to 1 if eax < 0 (negative)
js error            // jump if SF == 1
```

On IA32 systems, where is the value of old %ebp saved in relation to the current value of %ebp?

Svar: old %ebp is stored at (%ebp)

Which of the following is true:
(a) There are no IEEE float representations exactly equal to zero.
(b) There is one IEEE float representation exactly equal to zero.
(c) There are two IEEE float representations exactly equal to zero.
(d) There are many IEEE float representations exactly equal to zero

Which of the following is true:
(a) A function can immediately clear any “callee save” registers.
(b) The caller must always save all “caller save” registers before calling a function.
(c) The called function must immediately save all callee save registers on the stack and restore them before returning.
(d) A function can always ignore the initial values of all caller save registers.

The smallest unit on a typical hard disk is called

Svar: a sector

The expression x ∗ x ≥ 0 holds uniformly for =

Svar: unsigned integers, but not for signed integers

What is the evaluation result of expression
11102 ˆ 10102? = 01002 (1) 13 * x = (x << 3) + (x << 2) + x(2)

Svar: Absolute value of x = x * (1 | (x >> 7))

Which expression will evaluate to 0x1 if x is a multiple of 32 and 0x0 otherwise? Assume that x is an unsigned int.

Svar: !(x & 0x1f)

Why does the technique called “blocking” help with cache utilization when transposing a matrix?

Svar: Spatial locality

What is NOT true about 64-bit Linux systems?

Svar: All function arguments are passed on the stack

On a 64-bit system, if %rsp has the value 0x7ffff0000 immediately before a retq instruction, what is the value of %rsp immediately after the retq?

Svar: 0x7fffff0008

What is the difference between the mov and lea instructions?

Svar: mov dereferences an address, while lea doesn’t

In two’s compliment, what is the minimum number of bits needed to represent the numbers -1 and the number 1 respectively?

Svar: 1 and 2 (2 and 1 frekar?) ATH!

Consider the following program. Assuming the user correctly types an integer into stdin, what will the program output in the end?

```
#include <stdio.h>
int main(){
        int x = 0;
        printf("Please input an integer:");
        scanf("%d",x);
        printf("%d", (!!x)<<31);
}
```

Svar: Segmentation fault

By default, on Intel x86, the stack

Svar: Grows down towards smaller addresses

The leave instruction is effectively the same as which of the following:

Svar: mov %ebp, %esp, pop %ebp

Intel x86 64 systems are

Svar: Little endian

Select the two’s complement negation of the following binary value: 0000101101:

Svar: 1111010011

Which line of C-code will perform the same operation as leal 0x10(%rax,%rcx,4),%rax?

Svar: rax = 16 + rax + 4*rcx

Which line of Intel x86-64 assembly will perform the same operation as rcx = ((int *)rax)[rcx]?

Svar: mov (%rax,%rcx,4),%rcx

If a is of type (int) and b is of type (unsigned int), then (a < b) will perform

Svar: An unsigned comparison.

Denormalized floating point numbers are

Svar: Very close to zero (small magnitude)

Which of the following assembly instructions is invalid in Intel IA32 Assembly?

Svar: pop %eip

If %esp has the value 0xBFFF0000 before a call instruction, the value immediately after the call instruction (before the first instruction of the called function) is:

Svar: 0xBFFEFFFC

%rsp is 0xdeadbeefdeadd0d0. What is the value in %rsp after the following instruction executes?

Svar: 0xdeadbeefdeadd0c8

How many lines does a direct-mapped cache have in a set?

Svar: 1

Which of the following lines of C code performs the same operation as the assembly statement
lea 0xffffffff(%esi),%eax.

Svar: eax = esi – 1

1) mov (%eax, %eax, 4), %eax
2) lea (%eax, %eax, 4), %eax
Which of the above accomplishes the following: %eax = 5 * %eax

Svar: only 2

Which expression will evaluate to 0x1 if x is a multiple of 32 and 0x0 otherwise? Assume that x is an unsigned int.

Svar: !(x & 0x1f)

Which register holds the first arguement when an arguement is called in IA32 (32 bit) architecture with a non optimized C compiler?

Svar: None of the above (gildir bara fyrir x64)

```
pushl %ebp
movl %esp, %ebp
...
leave
...
```

The leave instruction is effectively the same as which of the following:

Svar: mov %ebp, %esp

Two‘s Complement

Description	Numb (6bit)
Umax (Max Unsigned)	2^6 = 63
Tmin	-2^6-1 = -32
(unsigned)((int) 4)	4
(unsigned) ((int) -7)	57
(((unsigned) 0x21) <<1) & 0x3F)	2
(int)(20+12)	-32
12 && 4	1
(! 0x15) > 16	0

Fyrir þessa að neðan int x = -5; unsigned ux = x;

Expression	4 bit Decimal	4 bit binary
-8	-8	1000
-Tmin	-8	1000
-x >> 1	2	0010
(x ^(1))>>2	-2	1110
Expression	**6 bit Decimal**	**6 bit Binary**
-8	-8	11 1000
-Tmin	-32	10 0000
-x >> 1	2	00 0101
(x ^(1))>>2	-2	11 1110

Floating Point :

Nomalized

Exponent field **Neither all-zero nor all-one**

- **E = e – bias**
- **M = 1 + f**

Denormalized

Exponent field is **all-zero**

- **E = 1 – bias**
- **M = f**

Special cases

Exponent field is **all-ones**

- **NaN = f = non-zero**
- **Inf = f = all-zero**

bias = $2^{k-1} - 1$
e = exponent
f = fraction
k = fjöldi bita í exponent
s = sign biti (plús eða mínus)
Answer = sM * 2^E

Brot yfir í binary

Dæmi: $\frac{5}{32}$

1. Reikna fyrst bias
2. Breyta tölu yfir strik í binary: $\frac{101}{32}$
3. Breyta tölu fyrir neðan strik í 2^n til að fá sömu tölu og var fyrir neðan strik: $\frac{101}{2^5}$
4. Færa neðri tölu fyrir ofan strik og endurskrifa með kommu: $101 * 2^{-5} = 0{,}00101$
5. Finna stærsta mögulega gildi á E (E = 1 – bias)
6. Færa kommuna á réttan stað, stoppa þegar annaðhvort:
   a. maður nær gildinu á stærsta E.
   b. þegar það er kominn einn ás vinstramegin við kommuna.
   c. $0{,}00101 = 0{,}101 * 2^{\boxed{-2}} \longrightarrow E$
7. 2^n --> n er núna E sem þú þarft að nota (sjá að ofan)
8. 0 vinstramegin við kommu = Denormalized
   a. Þarf bara að setja inn fraction hlutann (allt sem er hægramegin við kommu)
   b. $Svar$: 0 000 1010
9. 1 vinstramegin við kommu = Normalized
   a. Þarf að reikna e með formúlunni
   E = e – bias (umritað sem e = E + bias)

Binary yfir í brot

Dæmi

s eee f f f f
0 010 0110

Skoða exponent til að sjá hvort talan sé Denormalized eða Normalized.

Reikna bias.

Reikna E.

Reikna Mantissu:

$$M = 1 + f = 1 + \frac{6}{2^4} = \frac{16}{2^4} + \frac{6}{2^4} = \frac{22}{2^4} = 22 * 2^{-4}$$

Reikna Answer

$$A = sM * 2^E = 22 * 2^{-4} * 2^{-1} = 22 * 2^{-5} = \frac{22}{2^5} = \frac{11}{2^4} = \frac{11}{16}$$

Linux commands

> senda output inn í skrá (yfirskrifar allt)
>> append á skrá (bæta aftaná skrá/í neðstu línu)
> eða 1> (stdout í skrá – stderr á skjá)
2> (stderr í skrá – stdout á skjá)
&> (stdout og stderr í skrá / ekkert á skjá)
0> (stdout og stderr á skjá / ekkert í skrá)
grep d49 (sýnir allar línur sem innihalda d49)
cut –d ’:’ -f 2,4 (-d setur ’:’ sem delimeter, -f sýnir field númer 2 og 4)
head (sýnir fyrstu 10 línur af skjali)
tail (sýnir síðustu 10 línur af skjali)
less (gerir manni kleift að scrolla þægilega í skjali)
sort (raða innihaldi skjals)
uniq (eyða út línum sem eru eins hlið við hlið)
chmod (breyta aðgangi að skrám og folderum)
cp (copy)
mv (færa skrá)
rm (eyða skrá)
cd (change directory)

Cache

$$\boldsymbol{CO = log_2(fjöldi\ byte'a)}$$
$$\boldsymbol{CI = log_2\left(\frac{lines}{ways}\right)}$$
$$\boldsymbol{CT = rest}$$

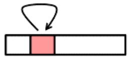
Skrá physical address inn í physical address format (einn bita í hvert hólf)

Skrá svo inn í töfluna út frá formattinu.

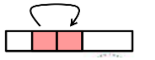
1. Skoðar töfluna, finnur hvaða index þú ert með
2. Skoðar svo tagið og finnur það.
3. Ef tagið er valid (valid = 1) þá er HIT og þú sækir gildi á byte offset og setur það í byte returned. Ef MISS þá er byte returned = "-" (mínus)

Direct mapped cache = One line per set

Temporal locality = Recently referenced items are likely to be referenced again in the near future



Spatial locality = Items with nearby addresses tend to be referenced close together in time



Match assembly function

```
foo1:
pushl %ebp                // setup
movl %esp,%ebp            // setup
movl 8(%ebp),%eax         // eax = x
sall $4,%eax              // eax = x << 4 => 16x
subl 8(%ebp),%eax         // eax = 16x- x => 15x
movl %ebp,%esp            // breakdown
popl %ebp                 // breakdown
ret

foo2:
pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%eax         // eax = x
testl %eax,%eax           // x & x
jge .L4                   // hoppar alltaf
addl $15,%eax             // hoppar yfir þetta
.L4:
sarl $4,%eax              // eax = x >> 4 => x / 16
movl %ebp,%esp
popl %ebp
ret

foo3:
pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%eax         // eax = x
shrl $31,%eax             // eax >> 31
movl %ebp,%esp            // svar (x < 0)
popl %ebp                 //utaf þetta er logical shift
ret                       //það tekkar bara á true
                            eða false
```

Assembly loop

```
int sum(int a, int b, int c, int d, int e, int f, int g) {
        return a + b + c + d + e + f + g;     }
```

sum: push %rbp movq %rsp,%rbp movq 16(%rbp),%rax leaq (%rax,%rdi),%rax leaq (%rax,%rsi),%rax →→→→	leaq (%rax,%rdx),%rax leaq (%rax,%rcx),%rax leaq (%rax,%r8),%rax leaq (%rax,%r9),%rax pop %rbp ret

```
int cmp(int a, int b) {    if (a > b)    return 1;
                           else if (a == b)    return 0;
                           else    return -1;    }
```

```
 cmp:                         .G:
      push %rbp                    movl $-1,%eax
      movq %rsp,%rbp               jmp .L2
      cmpl %edi,%esi          .E:
      jg .G                        movl $0,%eax
      je .E                   .L3:
      movl $1,%eax                 pop %rbp
      jmp .L2                      ret
 →-→-→-→-→
```

```
int idiv(int a, int b) {     return a / b;    }
```

```
 idiv:                        cdq
      push %rbp               idiv %esi
      movq %rsp,%rbp          pop %rbp
      movl %edi,%eax          ret
      →→→→
```

```
int mod(int a, int b) {     return a % b;    }
```

```
 mod:                         idiv %esi
      push %rbp               movq %rdx,%rax
      movq %rsp,%rbp          pop %rbp
      movl %edi,%eax          ret
      cdq
     →-→→→→
```

```
foo:
pushl %ebp                      //START
movl %esp,%ebp                  //START
movl 8(%ebp),%ecx              //ecx = *a
movl 16(%ebp),%edx             //edx = val
movl 12(%ebp),%eax             //eax = n
decl %eax                      //eax = n – 1 (n er = i)
js .L3                         //if (i < 0 ) goto L3
.L7:
cmpl %edx,(%ecx,%eax,4)        //a[i] – val  = temp
jne .L3                        //if (a[i] != val) goto L3
decl %eax                      //eax = i -1
jns .L7                        //if(i >= 0) goto L7
.L3:
movl %ebp,%esp                 //FINISH
popl %ebp)                     //FINISH
ret                            //FINISH
int foo(int *a, int n, int val) {
         int i;
         for (i = n - 1; a[i] ==  val && (i >= 0) ;
                 .... i = i – 1 ) {  ;  }
 return i;   }
```

```
foo:
pushl %ebp                      // SETUP
movl %esp,%ebp                  // SETUP
pushl %ebx                      // SETUP
movl 8(%ebp),%ebx               // ebx = a
leal 2(%ebx),%edx               // edx = 2 + a
xorl %ecx,%ecx                  // ecx xor ecx = 0
cmpl %ebx,%ecx                  // ecx = 0 >= a
jge .L4
.L6:
leal 5(%ecx,%edx),%edx    // edx = 5 + 0 + 2 + a = 7 + a
leal 3(%ecx),%eax         // eax = 3 + 0
imull %eax,%edx           // edx = 3 * (7 + a ) = 21 + 3a
incl %ecx                // ecx = i++ i = 1
cmpl %ebx,%ecx           // ecx = 1 < a
jl .L6
.L4:
movl %edx,%eax                  // eax = 21 + 3a
popl %ebx                       // FINISH
movl %ebp,%esp                  // FINISH
popl %ebp                       // FINISH
ret
int foo(int a) {   int i;
                int result = 2 + a;
                for(i = 0; i < a ; i++ ) {
                     result = result + 5 + i;
                     result = result * (3 + i );
                }          return result;    }
```

Assembly matrix

```
pushl    %ebp                    // Make the stack
movl     %esp, %ebp              // Make the pointers
movl     12(%ebp), %edx          // Set the pointer to edx
movl     %edx, %eax        // Add edx to eax (known as J)
addl     %eax, %eax              // eax = J + J = 2J
addl     %edx, %eax              // eax = J + 2J = 3J
addl     %eax, %eax              // eax = 3J + 3J = 6J
movl     8(%ebp), %edx           // edx = know as I
addl     %edx, %eax              // eax = 6J + I
movl     mat2(,%eax,4), %ecx  // Mat2 = ecx = 4*(6J + I)
movl     8(%ebp), %eax           // eax = I
sall    $2, %eax                 // eax = I*(2^2) = 4I
leal    0(,%eax,8), %edx         // edx = 8*(I*(2^2))= 32I
subl     %eax, %edx              // edx = 7*(I*(2^2))= 28I
movl     12(%ebp), %eax          // eax = J
addl     %edx, %eax              // eax = 28I + J
movl     %ecx, mat1(,%eax,4)  // Mat1 = 4*(28I + J)
popl     %ebp                    // Prepare to close
ret                              // Return and close
```

mat1[i][j] = mat1[i*N + j] = mat1 + 4*(i*N + j)
mat2[j][i] = mat2[j*M + i] = mat2 + 4*(j*M + i)

A[i][j]
A + (i*C + j) * k
C = Column size
k = Size of datatype
int array1[M][N]
int array2[N][M]

// Mat1 = 4*(6J + I) = mat1[4*(j*6(n) + i)]
// Mat2 = 4*(28I + J) = mat2[4*(28(m)*i + j)]
// So if this apply we can say that M = 28 & N = 6

```
void copy(int i, int j){ array1[i][j] = array2[j][i]; }
copy:
pushl %ebp                      //SETUP
movl %esp,%ebp                  //SETUP
pushl %ebx                      //SETUP
movl 8(%ebp),%ecx               //%ecx = i
movl 12(%ebp),%eax              //%eax = j
leal 0(,%eax,4),%ebx            //%ebx = 0 + j * 4 = 4j
leal 0(,%ecx,8),%edx            //%edx = 0 + i * 8 = 8i
subl %ecx,%edx                  //%edx = 8i – i = 7i
addl %ebx,%eax                  //eax = j + 4j = 5j
sall $2,%eax              //eax = 5j << 2 Λ 2 = 5j * 4 = 20j
movl array2(%eax,%ecx,4),%eax
                     //eax = 20j + i * 4 = array2(20j + 4i)
movl %eax,array1(%ebx,%edx,4)
       //eax = array2. Array1 4j + 7i * 4 = array1(4j + 28i)
popl %ebp                       //FINISH
ret
M = 5          N = 7
     ARRAY2 = 4(5j + i)        ARRAY1 = 4(j + 7i)
```

```
array1[i][j] = array2[j][i];
copy:
pushl %ebp                      //SETUP
movl %esp, %ebp                 //SETUP
pushl %ebx                      //SETUP
movl 8(%ebp), %eax              //eax = i
movl 12(%ebp), %edx             //edx = j
leal 0(,%eax,8), %ecx           //ecx = 0 + i * 8 = 8i
subl %eax, %ecx                 //ecx = 8i – i = 7i
addl %edx, %ecx                 //ecx = 7i + j
movl %edx, %ebx                 //ebx = j
sall $4, %ebx                   // ebx = j << 4 (2 Λ 4) = 16j
leal (%ebx,%edx), %edx          //edx = 16j + j = 17j
leal (%edx,%eax), %eax          //eax = 17j + i
movl array2(,%eax,4), %eax // eax = array2  = 4(17j + i)
movl %eax, array1(,%ecx,4) //array2, array1 = 4(7i + j)
popl %ebp                       //FINISH
ret
M =17      N =7
     ARRAY2= 4(17j + i)       ARRAY1= 4(7i + j)
```

Stack

Fyllið inn hvernig stakkurinn verður eftir keyrslu foo

```
foo (int a, int b, int c, int d);
push %ebp
mov %esp, %ebp
push %ebx
sub $0x10, %esp
movl $0xdeadbeef, -4(%ebp) <- yfirskrifar %ebx á stakk
```

0xFFFFD600	Int d	
+10	Int c	
+c	Int b	
+8	Int a	
+4	+return addres	
+0	Saved %ebp	Frame ptr %ebp
-4	0xdeadbeef	hér var %ebx
-8	Drasl	
-c	Drasl	
-10	Drasl	
-14		%esp

Stakkurinn er geymdur efst í minni, fyrir neðan stýrikerfið og stækkar niðurávið svo hann rekist ekki í forritið sem er geymt neðst

Parametrar fara í caller frame

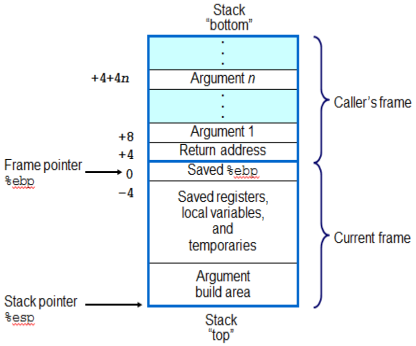
Ef við sjáum plústölu fyrir framan %ebp þá er verið að setja parameter inn á caller frame. Mínustala = setja í fallið/stakkinn/local breyta.

Call skipunin gerir tvennt. 1. vistar/push return addressuna á stakkinn-minnisaddressa sen caller

frame ætti að halda áfram eftir fallið. 2. Hoppar inn í fallið, breytir %eip (instruction pointer) og heldur áfram að keyra fallið.

leave skipunin-passar að base pointerinn sé á réttum stað og setur stack pointerinn á base pointerinn

ret skipunin- poppar vistuðu return addressunni af stakknum og heldur áfram að keyra af þeirri addressu.



0xffffffff	[kernel]	
	[stack]	
	↓	
	[lib]	
	↑	
	[heap]	new / malloc
0x00000000	[text]	The program

MEMORY:

SRAM: dýrara þolir meiri truflanir notað fyrir cash, 6 transistorar.

DRAM: þarf að refresha og mjög næmt fyrir truflunum.1 transistor.

EEPROM: Is erasable electronically. Flash memory is a type of EEPROM, which can be partially erased.

EPROM: Is not erasable electronically (it is erasable).

Stuff

0x82 4B AC:
Big Endian: {0x82, 0x4B, 0xAC},
Little Endian: {0xAC, 0x4B, 0x82}

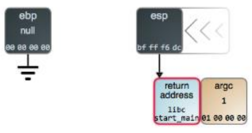
((a^b)&~b)|(~(a^b)&b)	a
1+(a<<3)+~a	a*7
(a<<4)+(a<<2)+(a<<1)	a*22
a ^ (MIN_INT + MAX_INT)	~a
~(~a | !(b ^ ~b))	a
1 + (a << 3) + ~a	7 * a
b >> 2	b / 4
~((a>>31)<<1)	(a < 0) ? 1 : -1

INC (incl) A=A+1
DEC (decl) A=A-1

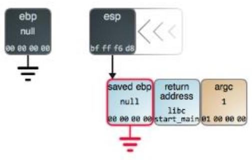
Harðir diskar

Hard disks consist of multiple platters. Each platter contains 2 surfaces, which contains multiple tracks, which contains multiple sectors, separated by gaps. A surface is split into multiple recording zones, with different track density. Most hard disks spin at a constant speed. The slowest part of reading from the hard disk is the *seek time*, followed by the *rotational latency*, the fastest generally being the actual data.

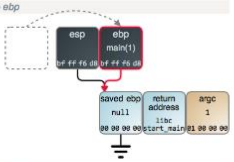
```
1.    call main # push return address onto stack, jump into main
```



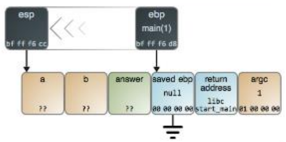
```
2.    pushl %ebp # save current ebp register value
```



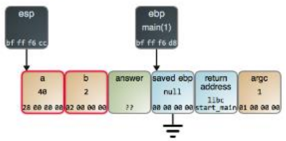
```
3.    movl %esp, %ebp # copy esp to ebp
```



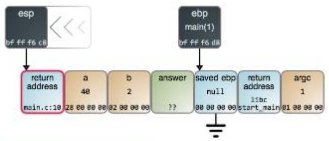
```
4.    subl $12, %esp # make room for stack data
```



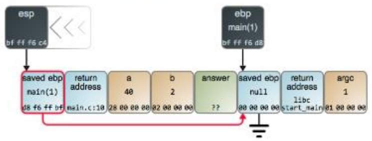
```
5.    movl $2, 4(%esp) # set b to 2
      movl $40, (%esp) # set a to 40
```



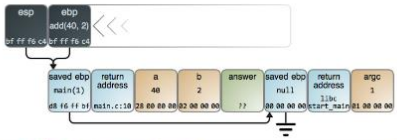
```
6.    call add # push return address onto stack, jump into add
```



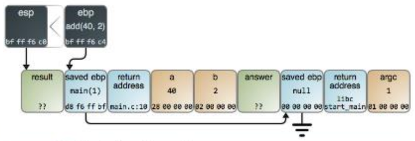
```
7.    pushl %ebp # save current ebp register value
```



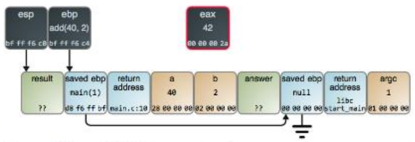
```
8.    movl %esp, %ebp # copy esp to ebp
```



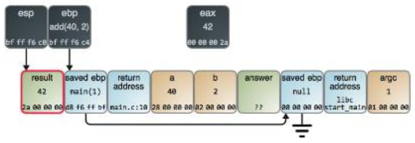
```
9.    subl $4, %esp # make room for result
```



```
      movl 12(%ebp), %eax # move b to eax
10.   movl 8(%ebp), %edx  # move a to edx
      addl %edx, %eax     # add edx into eax. total is 42.
```



```
11.   movl %eax, -4(%ebp) # copy eax to result
```


esp
bf ff f6 c8
ebp
add(40, 2)
bf ff f6 c4
eax
42
00 00 00 2a
result
42
2a 00 00 00
saved ebp
main(1)
d8 f6 ff bf
return address
main.c:10
a
40
28 00 00 00
b
2
02 00 00 00
answer
??
saved ebp
null
00 00 00 00
return address
libc
start_main
argc
1
01 00 00 00