Value	Representation (bits)	Representation hex	Representation decimal
	63-bit, 2's complement number	**tag bit** (1=num, 0=bool)	
9	0000 0000 0000 ... 0000 0000 0001 **0011**	0x000...0013	19
-2	1111 1111 1111 ... 1111 1111 1111 **1101**	0xFFF...FFFD	-3
33	0000 0000 0000 ... 0000 0000 0100 0011	0x000...0043	67
true	**1111** 1111 1111 ... 1111 1111 1111 **1110**	0xFFF...FFFE	-2
false	**0111** 1111 1111 ... 1111 1111 1111 **1110**	0x7FF...FFFE	$2^{63}-2$
	true/false	62 bits for future value representations!	

type expr =
| EBool of bool
| ENum of int

32 — 16 8 4 2 1

```
let rec e_to_is (e : expr) si env =
  match e with
    | ENum(n) ->
    | EBool(b) ->
```

sprintf "mov rax, %d \n shl rax, 1 \n add rax, 1" n

sprintf "mov rax, %d" ((2*n)+1)

if b then ["mov rax, 0xFFFFFFFF...E"]
else ["mov rax, 0x7FF.... E"]

```
#include <stdio.h>

extern int64_t our_code_starts_here()
                asm("our_code_starts_here");

int main(int argc, char** argv) {
  int64_t result = our_code_starts_here();
  printf("%lld\n", result);
  return 0;
}
```

What will print from main for the program that's just the number 5?
For the program that's just the constant `true`?

What instructions should we produce for `(+ x y)` if we ignore errors? (Assume x at stackloc 1, y at stackloc 2)

```
mov rax, [rsp-8]
and rax, 0xFF.....E   (remove tag)
add rax, [rsp-16]
```

val	rep
x = m	2m+1
y = n	2n+1
m+n	2(m+n)+1

Why this works

```
| EPlus(e1, e2) ->
```

Other idea:
(arith) shift both to right
do op
shift left
add tag bit

2m + 2n + 1
2(m+n) + 1

What instructions should we produce for `(+ x y)` if we want to *stop and print an error message* if x or y is a boolean?
What else might need to change?

```
mov rax, [rsp-8]
and rax, 1
cmp rax, 0
je op-error
mov [rsp-16], rax
```

(idea - and x with y to check both)

```
let compile (program : string) : string =
  let ast = parse program in
  let instrs = e_to_is ast 1 [] in
  let all_is = (String.concat "\n" instrs) in
  (sprintf "
section .text
global our_code_starts_here
extern print_error_and_exit
our_code_starts_here:
  %s
  ret
op-error:

" all_is)
```

```
#include <stdio.h>

extern int64_t our_code_starts_here() asm("our_code_starts_here");

void print_err_exit() {
  eprintf("Error!");
  exit(1);
}
```

What if we want to get input from *outside the program*?
First mechanism – command-line arguments. How to communicate to `our_code_starts_here` ?