setjmp/longjmp

```
jmp_buf env;

main() {
  if (setjmp(env) == 0) { // like try {
    ...
    read_file()
    ...
  } else { // like catch
    printf("some␣error␣happened\n");
  }
}

read_file() {
  ...
  if (open failed) {
      longjmp(env, 1) // like throw
  }
  ...
}
```

implementing setjmp/longjmp

setjmp:

- copy all registers to `jmp_buf`
- ... including stack pointer

longjmp

- copy registers from `jmp_buf`
- ... but change `%rax` (return value)

setjmp psuedocode

setjmp: looks like first half of context switch

```
setjmp:
  movq %rcx, env->rcx
  movq %rdx, env->rdx
  movq %rsp + 8, env->rsp // +8: skip return value
  ...
  save_condition_codes env->ccs
  movq 0(%rsp), env->pc
  movq $0, %rax // always return 0
  ret
```

longjmp psuedocode

longjmp: looks like second half of context switch

```
longjmp:
  movq %rdi, %rax // return a different value
  movq env->rcx, %rcx
  movq env->rdx, %rdx
  ...
  restore_condition_codes env->ccs
  movq env->rsp, %rsp
  jmp env->pc
```

setjmp weirdness — local variables

Undefined behavior:

```
int x = 0;
if (setjmp(env) == 0) {
    ...
    x += 1;
    longjmp(env, 1);
} else {
    printf("%d\n", x);
}
```

setjmp weirdness — fix

Defined behavior:

```
volatile int x = 0;
if (setjmp(env) == 0) {
    ...
    x += 1;
    longjmp(env, 1);
} else {
    printf("%d\n", x);
}
```

on implementing try/catch

could do something like setjmp()/longjmp()

but setjmp is slow

setjmp exercise

```
jmp_buf env; int counter = 0;
void bar() {
    putchar('Z');
    ++counter;
    if (counter < 2) {
        longjmp(env, 1);
    }
}
int main() {
    while (setjmp(env) == 1) {
        putchar('X');
    }
    putchar('Y');
    bar();
}
```

Expected output?

A. YZ B. XYZ C. YZYZ D. XYZXYZ

setjmp exercise soln

```
jmp_buf env; int counter = 0;
void bar() {
    putchar('Z');                  //           3 Z                  12 Z
    ++counter;                     //           4                    13
    if (counter < 2) {             //           5 (1<2)              14 (2<2)
        longjmp(env, 1);           //           6*
    }                              //                                15
}
int main() {
    while (setjmp(env) == 1) {     // 0 (ret 0) 7*(ret 1) 9 (ret 0)
        putchar('X');              //           8 X
    }
    putchar('Y');                  // 1 Y                  10 Y
    bar();                         // 2                    11
}                                  //                                16
```

on implementing try/catch

could do something like setjmp()/longjmp()

but setjmp is slow

low-overhead try/catch (1)

```
main() {
  printf("about␣to␣read␣file\n");
  try {
    read_file();
  } catch(...) {
    printf("some␣error␣happened\n");
  }
}
read_file() {
  ...
  if (open failed) {
      throw IOException();
  }
  ...
}
```

low-overhead try/catch (2)

```
main:
  ...
  call printf
start_try:
  call read_file
end_try:
  ret
```

```
main_catch:
  movq $str, %rdi
  call printf
  jmp end_try
```

```
read_file:
  pushq %r12
  ...
  call do_throw
  ...
end_read:
  popq %r12
  ret
```

lookup table

program counter range	**action**	**recurse?**
`start_try` to `end_try`	`jmp main_catch`	no
`read_file` to `end_read`	`popq %r12, ret`	yes
anything else	error	—

low-overhead try/catch (2)

```
main:
  ...
  call printf
start_try:
  call read_file
end_try:
  ret
```

```
main_catch:
  movq $str, %rdi
  call printf
  jmp end_try
```

```
read_file:
  pushq %r12
  ...
  call do_throw
  ...
end_read:
  popq %r12
  ret
```

lookup table

program counter range	action	recurse?
`start_try` to `end_try`	`jmp main_catch`	no
`read_file` to `end_read`	`popq %r12, ret`	yes
anything else	error	—

low-overhead try/catch (2)

```
main:
  ...
  call printf
start_try:
  call read_file
end_try:
  ret
```

```
main_catch:
  movq $str, %rdi
  call printf
  jmp end_try
```

```
read_file:
  pushq %r12
  ...
  call do_throw
  ...
end_read:
  popq %r12
  ret
```

lookup table

program counter range	action	recurse?
`start_try` to `end_try`	`jmp main_catch`	no
`read_file` to `end_read`	`popq %r12, ret`	yes
anything else	error	—

low-overhead try/catch (2)

```
main:
  ...
  call printf
start_try:
  call read_file
end_try:
  ret
```

```
main_catch:
  movq $str, %rdi
  call printf
  jmp end_try
```

```
read_file:
  pushq %r12
  ...
  call do_throw
  ...
```

not actual x86 code to run
track a "virtual PC" while looking for catch block

lookup table

program counter range	action	recurse?
`start_try` to `end_try`	`jmp main_catch`	no
`read_file` to `end_read`	`popq %r12, ret`	yes
anything else	error	—

lookup table tradeoffs

no overhead if throw not used

handles local variables on registers/stack, but...

larger executables (probably)

extra complexity for compiler

backup slides