




# Random Low-Level Stuff




Drahflow

# Scope

## Context:

-  Coded elymas for one year
-  Hand-made 80x86 assembler
-  Debugging with gdb and judicious usage of UD2

## This talk:

-  CPU and debugger disagree I
-  CPU and debugger disagree II
-  CPU and intuition disagree I

## What I saw

```
=> 0x300000252233:      movabs $0x300000216000,%rax
```

```
(gdb)
```

```
0x000030000025223d in ?? ()
```

```
7: x/i $rip
```

```
=> 0x30000025223d:      callq  *%rax
```

```
(gdb) info registers
```

```
rax          0xcc00300000216000  -3746942113411932160
```

```
rbx          0x30000021f093      52776560357523
```

## How I solved it

```
(gdb) disable 1  
(gdb) run
```

## What I saw

```
0x0000000000400079 in _start ()
```

```
1: x/i $rip
```

```
=> 0x400079 <_start+1>: jae    0x77
```

```
(gdb) stepi
```

```
0x0000000000400079 in _start ()
```

```
1: x/i $rip
```

```
=> 0x400079 <_start+1>: jae    0x77
```

```
(gdb)
```

## Why this is

```
(gdb) x /7bx $rip
0x400079 <_start+1>:    0x66    0x0f    0x83
                    0xf9    0xff    0xff
                    0xff
```

This opcode sequence would intuitively be 16-width relative conditional jump. But 64-bit mode has it undefined. In fact, it's doing different things on Intel vs. AMD.

## Surprising capabilities

Bit test (and set / reset)

```
bt1 $2, %rcx
```

```
bt1 %eax, %rcx
```

```
bt1 %eax, (%rcx)
```

## Surprising capabilities

“ If the bit base operand is a memory location, bit 0 of the byte at the specified address is the bit base of the bit string. If the bit index is in a register, the instruction selects a bit position relative to the bit base in the range  $-2^{63}$  to  $+2^{63}-1$  if the operand size is 64,  $-2^{31}$  to  $+2^{31}-1$ , if the operand size is 32, and  $-2^{15}$  to  $+2^{15}-1$  if the operand size is 16. ”