手搓 SIMD

人肉 compiler 之路

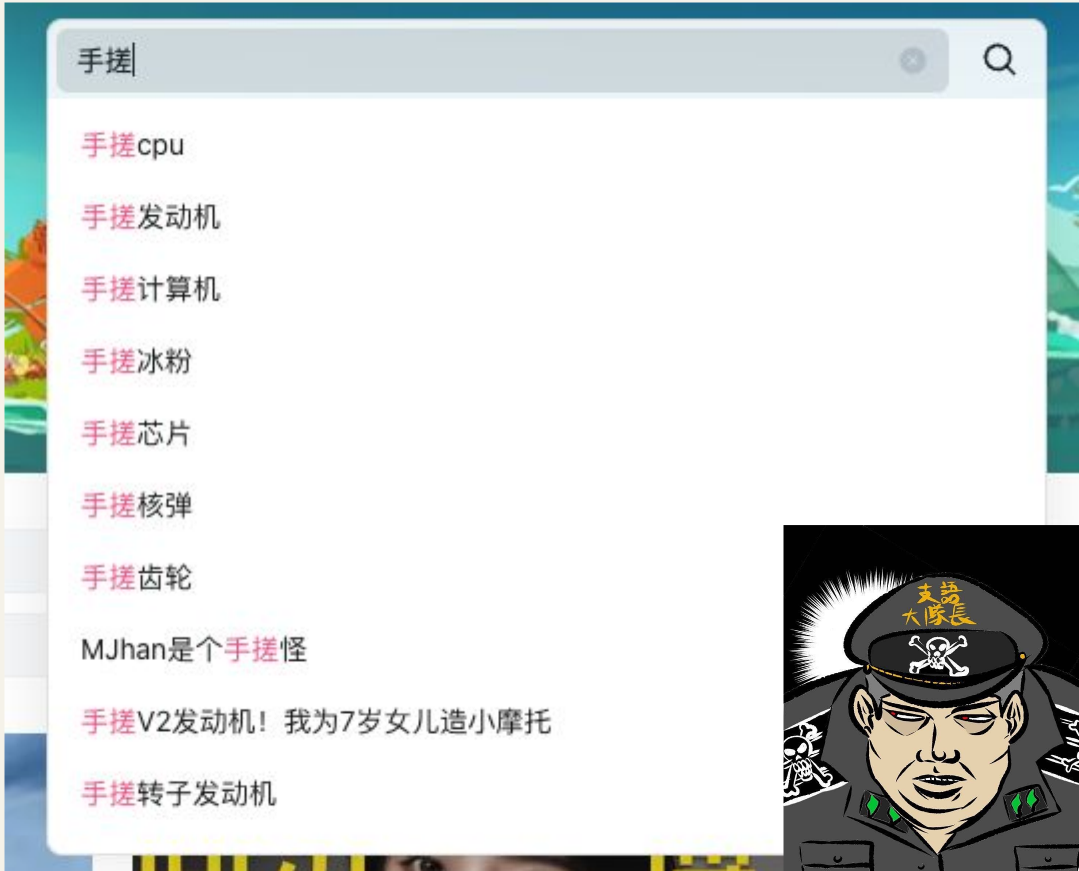
scc@teamt5.org

TEAMT5 杜浦數位安全

Persistent Cyber Threat Hunters

命名來源：B站

- 手搓大模型
- 手搓cpu
- …



手搓

手搓cpu

手搓发动机

手搓计算机

手搓冰粉

手搓芯片

手搓核弹

手搓齿轮

MJhan是个手搓怪

手搓V2发动机！我为7岁女儿造小摩托

手搓转子发动机

在我之前介紹的 SIMD

```
char *strchr(const char*, int);
```

在我之前介紹的 SIMD

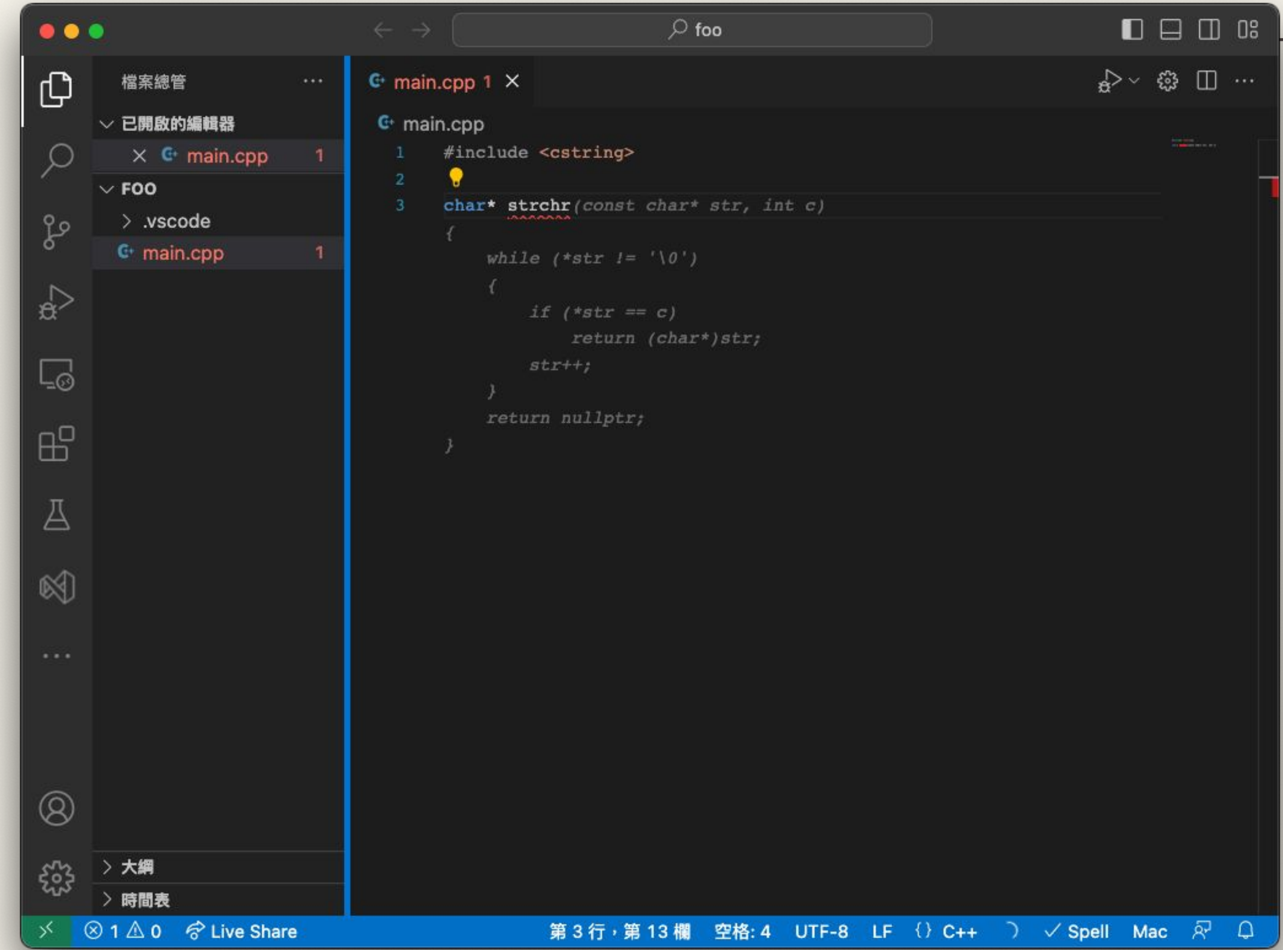
TEAMT5 杜浦數位安全

```
char *strchr(const char* s, int c);
```

The strchr function locates the **first occurrence** of c (converted to a char) in the string pointed to by s. The terminating null character is considered to be part of the string. §7.24.5.2

CS 101

TEAMT5 杜浦數位安全



main.cpp

```cpp
#include <cstring>

char* strchr(const char* str, int c)
{
    while (*str != '\0')
    {
        if (*str == c)
            return (char*)str;
        str++;
    }
    return nullptr;
}
```

CS 101

```c
char* strchr(const char* str, int c) {
    while (*str != '\0') {
        if (*str == c)
            return (char*)str;
        str++;
    }
    return nullptr;
}
```

肉眼 compiler 上場

x86 應該怎麼寫？

CS 101

```c
char* strchr(const char* str, int c) {
    while (*str != '\0') {
        if (*str == c)
            return (char*)str;
        str++;
    }
    return nullptr;
}
```

```asm
strchr(char const*, int):
        push rbp
        mov rbp, rsp
        mov QWORD PTR [rbp-8], rdi
        mov DWORD PTR [rbp-12], esi
        jmp .while_loop
.if_match:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        movsx eax, al
        cmp DWORD PTR [rbp-12], eax
        jne .str_pp
        mov rax, QWORD PTR [rbp-8]
        jmp .ret
.str_pp:
        inc QWORD PTR [rbp-8]
.while_loop:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        test al, al
        jne .if_match
        mov eax, 0
.ret:
        pop rbp
        ret
```

CS 101

```
char* strchr(const char* str, int c) {
    while (*str != '\0') {
        if (*str == c)
            return (char*)str;
        str++;
    }
    return nullptr;
}
```

```
strchr(char const*, int):
        push rbp
        mov rbp, rsp
        mov QWORD PTR [rbp-8], rdi
        mov DWORD PTR [rbp-12], esi
        jmp .while_loop
.if_match:
        mov rax, QWORD PTR [    -8]
        movzx eax, BYTE PTR [
        movsx eax, al
        cmp DWORD PTR [rbp
        jne .str_pp
        mov rax, QWORD PTR [rbp-8]
        jmp .ret
.str_pp:
        inc QWORD PTR [rbp-8]
.while_loop:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        test al, al
        jne .if_match
        mov eax, 0
.ret:
        pop rbp
        ret
```

char* str

int c

CS 101

```c
char* strchr(const char* str, int c) {
    while (*str != '\0') {
        if (*str == c)
            return (char*)str;
        str++;
    }
    return nullptr;
}
```

```asm
strchr(char const*, int):
        push rbp
        mov rbp, rsp
        mov QWORD PTR [rbp-8], rdi
        mov DWORD PTR [rbp-12], esi
        jmp .while_loop
.if_match:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        movsx eax, al
        cmp DWORD PTR [rbp-12], eax
        jne .str_pp
        mov rax, QWORD PTR [rbp-8]
        jmp .ret
.str_pp:
        inc QWORD PTR [rbp-8]
.while_loop:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        test al, al
        jne .if_match
        mov eax, 0
.ret:
        pop rbp
        ret
```

CS 101

```c
char* strchr(const char* str, int c) {
    while (*str != '\0') {
        if (*str == c)
            return (char*)str;
        str++;
    }
    return nullptr;
}
```

```asm
strchr(char const*, int):
        push rbp
        mov rbp, rsp
        mov QWORD PTR [rbp-8], rdi
        mov DWORD PTR [rbp-12], esi
        jmp .while_loop
.if_match:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        movsx eax, al
        cmp DWORD PTR [rbp-12], eax
        jne .str_pp
        mov rax, QWORD PTR [rbp-8]
        jmp .ret
.str_pp:
        inc QWORD PTR [rbp-8]
.while_loop:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        test al, al
        jne .if_match
        mov eax, 0
.ret:
        pop rbp
        ret
```

CS 101

```c
char* strchr(const char* str, int c) {
    while (*str != '\0') {
        if (*str == c)
            return (char*)str;
        str++;
    }
    return nullptr;
}
```

```asm
strchr(char const*, int):
        push rbp
        mov rbp, rsp
        mov QWORD PTR [rbp-8], rdi
        mov DWORD PTR [rbp-12], esi
        jmp .while_loop
.if_match:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        movsx eax, al
        cmp DWORD PTR [rbp-12], eax
        jne .str_pp
        mov rax, QWORD PTR [rbp-8]
        jmp .ret
.str_pp:
        inc QWORD PTR [rbp-8]
.while_loop:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        test al, al
        jne .if_match
        mov eax, 0
.ret:
        pop rbp
        ret
```

CS 101

```
char* strchr(const char* str, int c) {
    while (*str != '\0') {
        if (*str == c)
            return (char*)str;
        str++;
    }
    return nullptr;
}
```

```
strchr(char const*, int):
        push rbp
        mov rbp, rsp
        mov QWORD PTR [rbp-8], rdi
        mov DWORD PTR [rbp-12], esi
        jmp .while_loop
.if_match:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        movsx eax, al
        cmp DWORD PTR [rbp-12], eax
        jne .str_pp
        mov rax, QWORD PTR [rbp-8]
        jmp .ret
.str_pp:
        inc QWORD PTR [rbp-8]
.while_loop:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        test al, al
        jne .if_match
        mov eax, 0
.ret:
        pop rbp
        ret
```

CS 101

```c
char* strchr(const char* str, int c) {
    while (*str != '\0') {
        if (*str == c)
            return (char*)str;
        str++;
    }
    return nullptr;
}
```

```asm
strchr(char const*, int):
        push rbp
        mov rbp, rsp
        mov QWORD PTR [rbp-8], rdi
        mov DWORD PTR [rbp-12], esi
        jmp .while_loop
.if_match:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        movsx eax, al
        cmp DWORD PTR [rbp-12], eax
        jne .str_pp
        mov rax, QWORD PTR [rbp-8]
        jmp .ret
.str_pp:
        inc QWORD PTR [rbp-8]
.while_loop:
        mov rax, QWORD PTR [rbp-8]
        movzx eax, BYTE PTR [rax]
        test al, al
        jne .if_match
        mov eax, 0
.ret:
        pop rbp
        ret
```

CS 101

```c
char* strchr(const char* str, int c) {
    while (*str != '\0') {
        if (*str == c)
            return (char*)str;
        str++;
    }
    return nullptr;
}
```

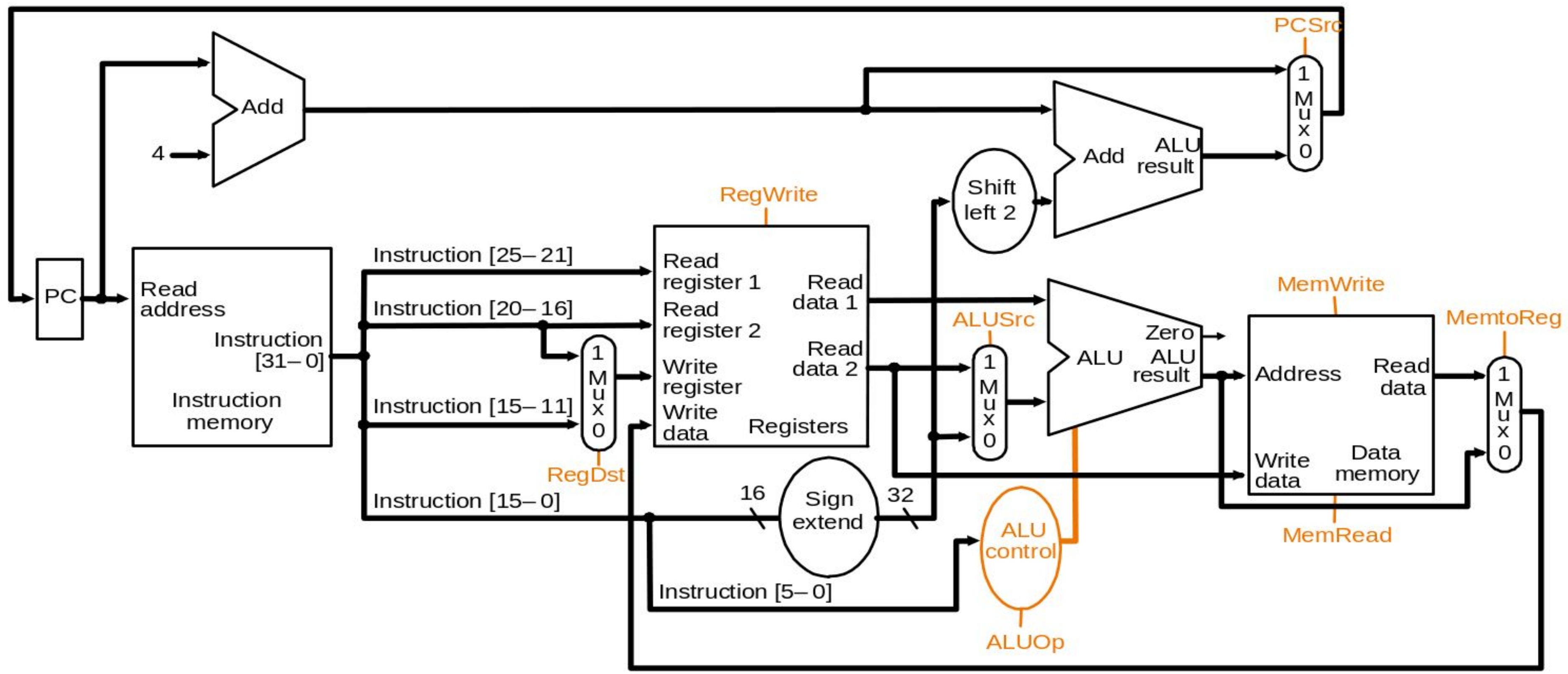
```asm
strchr(char const*, int):
.while_loop:
        movsx edx, BYTE PTR [rdi]
        test dl, dl
        jne .if_match
        xor eax, eax
        ret
.if_match:
        cmp edx, esi
        je .found
        inc rdi
        jmp .while_loop
.found:
        mov rax, rdi
        ret
```

What’s the problem

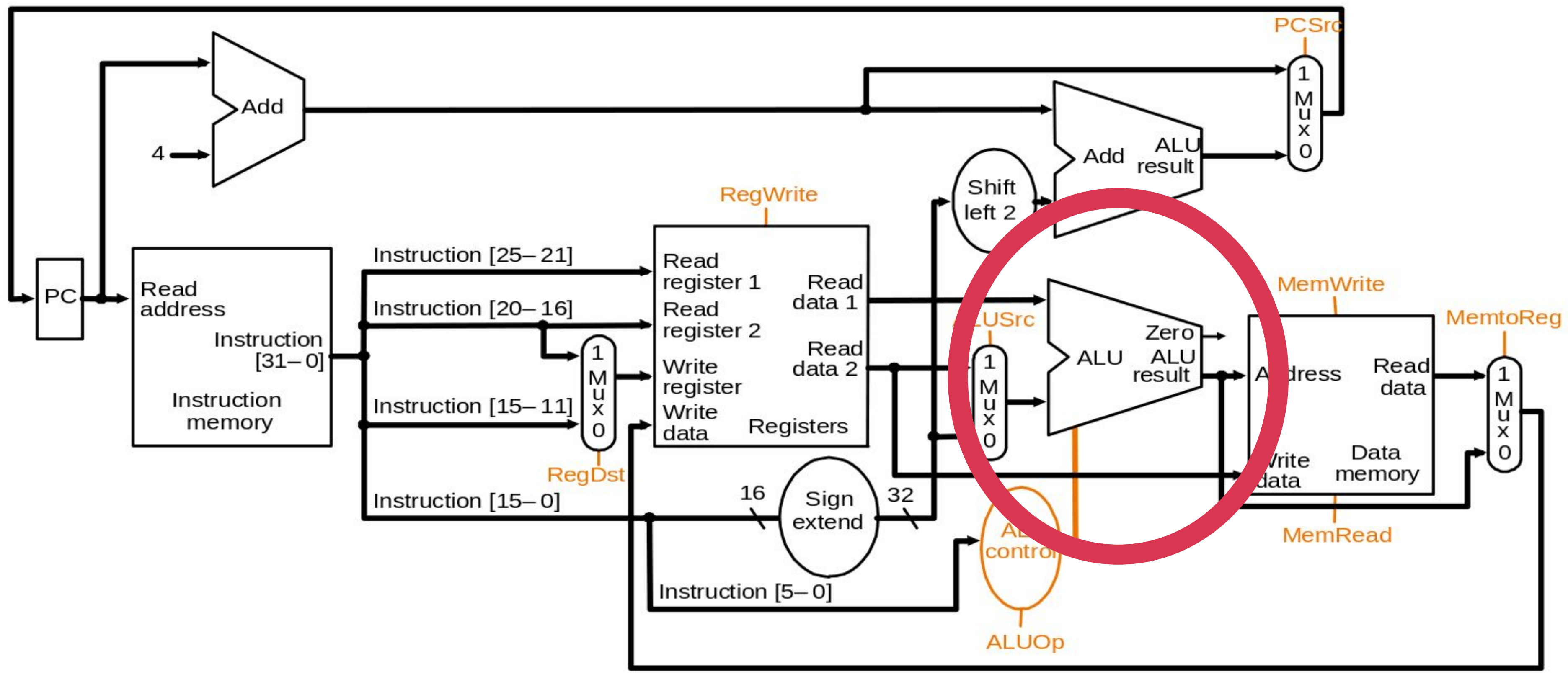
TEAMT5 杜浦數位安全

You’re in 32 or 64 bit system

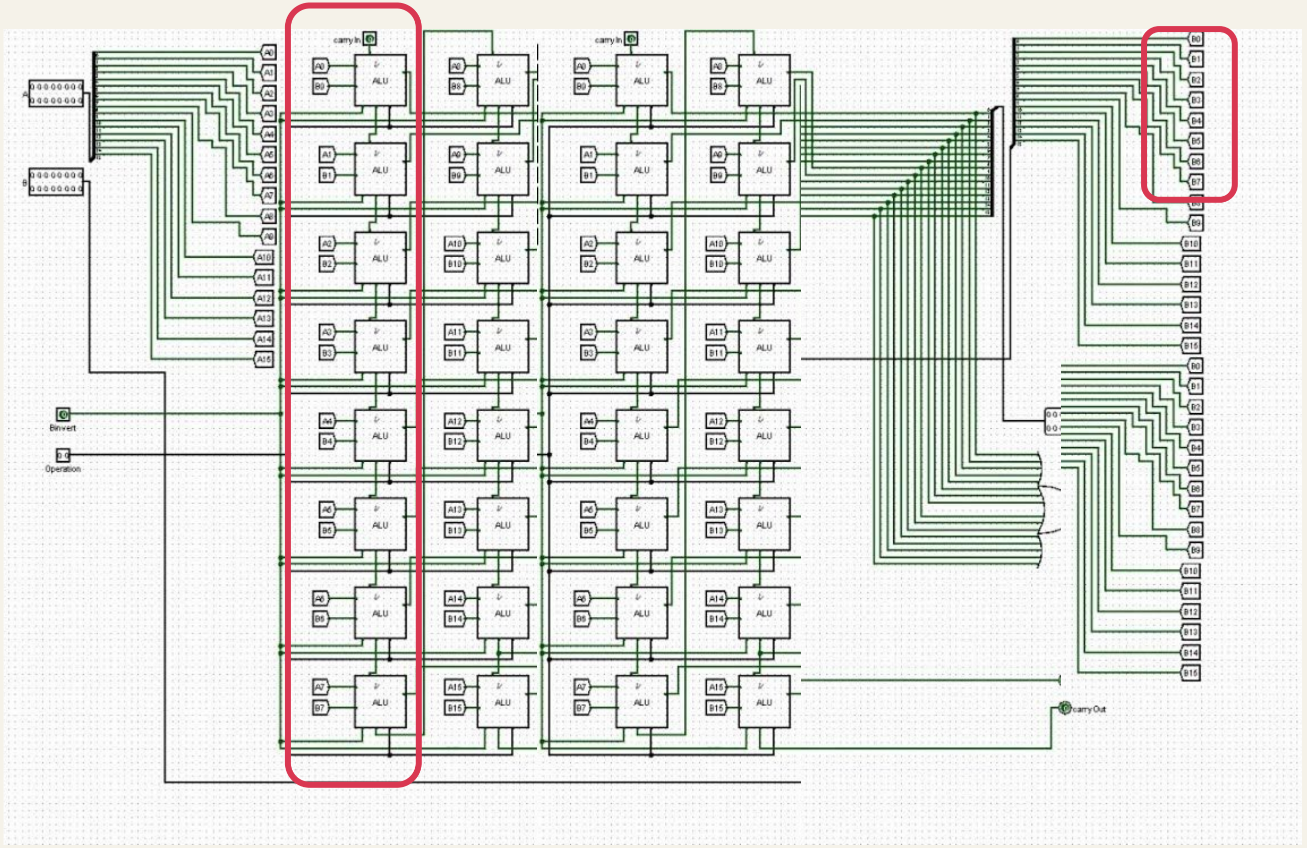
Single core CPU



Single core CPU

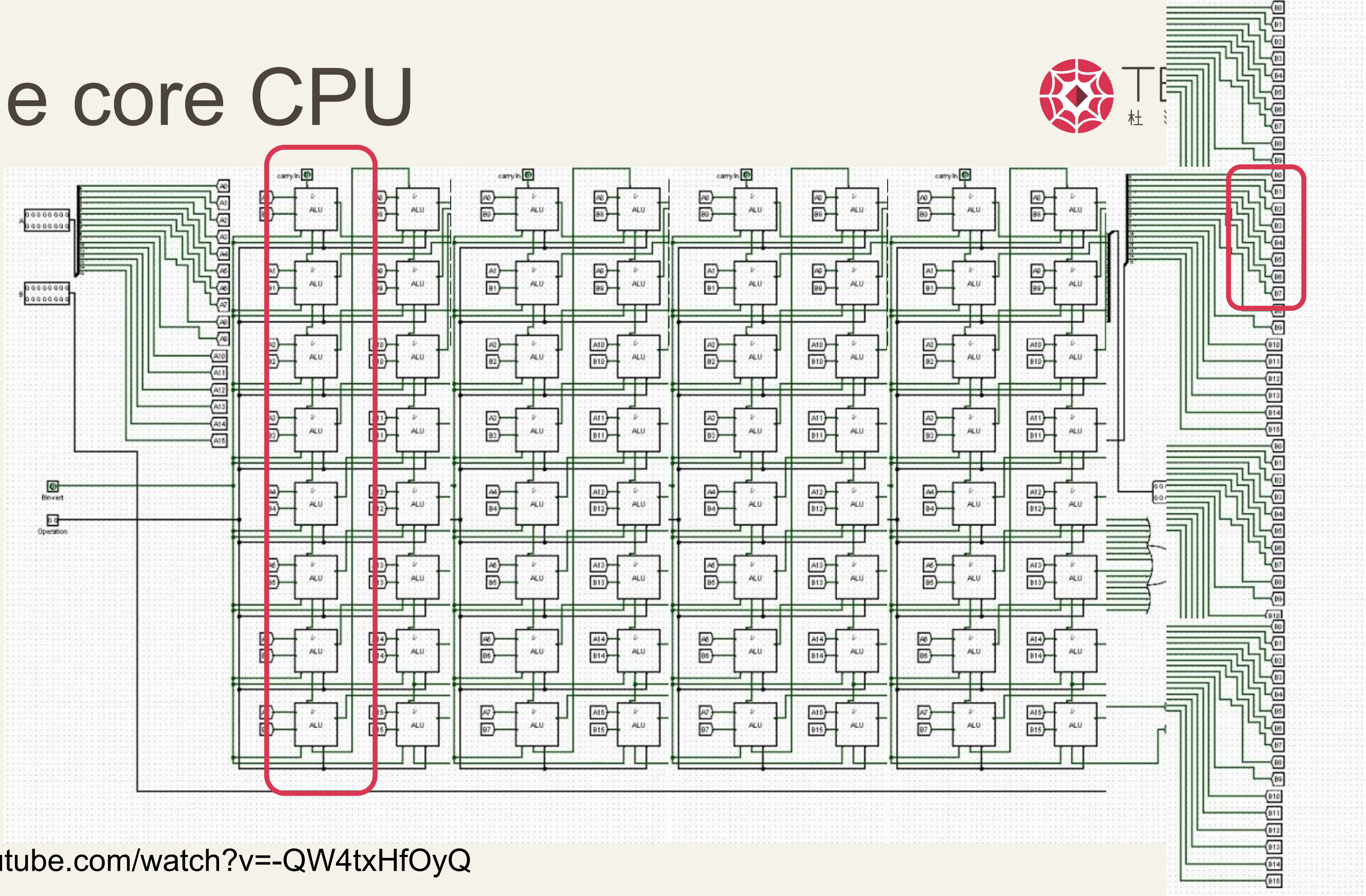


Single core CPU



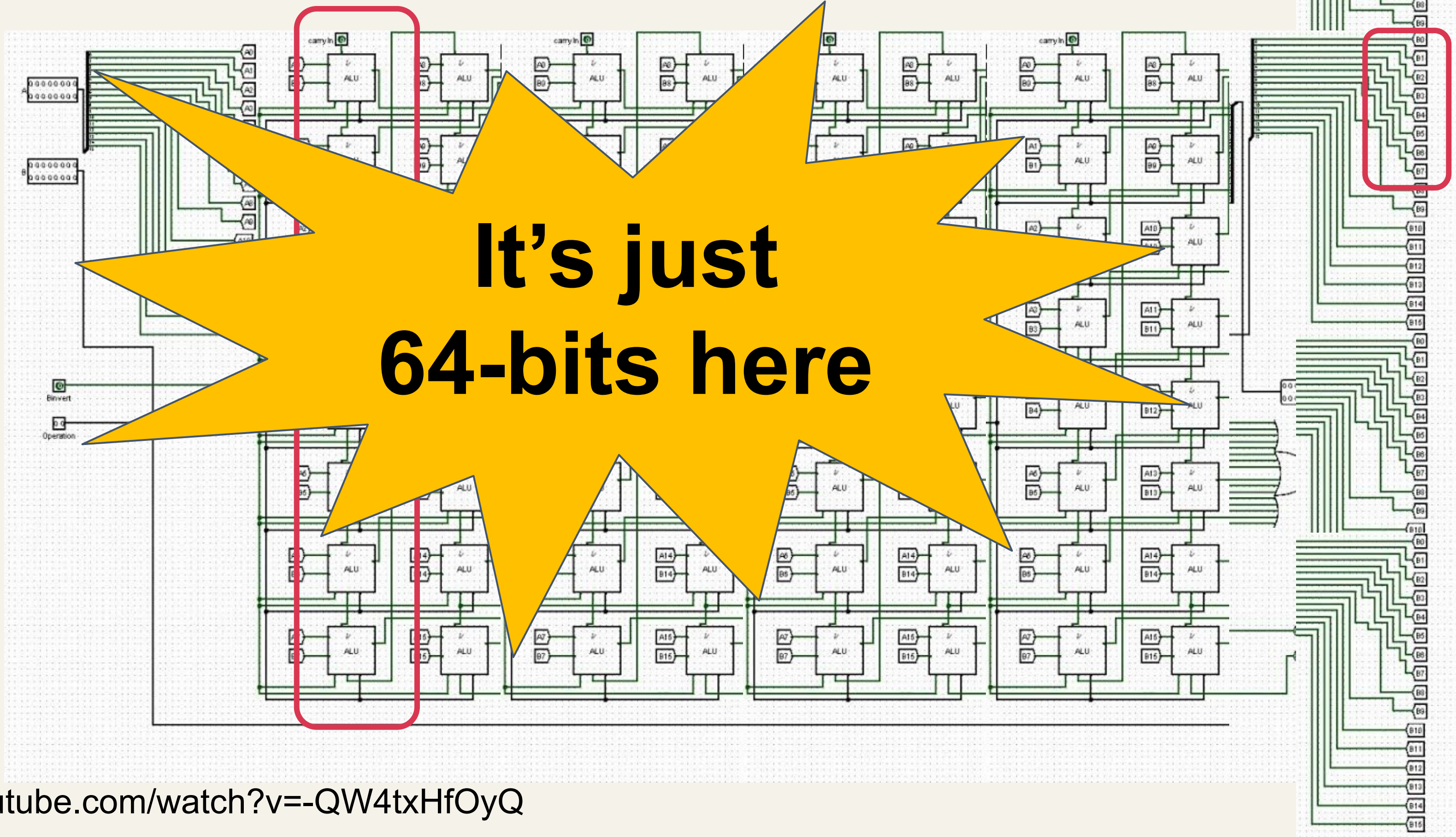
https://www.youtube.com/watch?v=-QW4txHfOyQ

Single core CPU



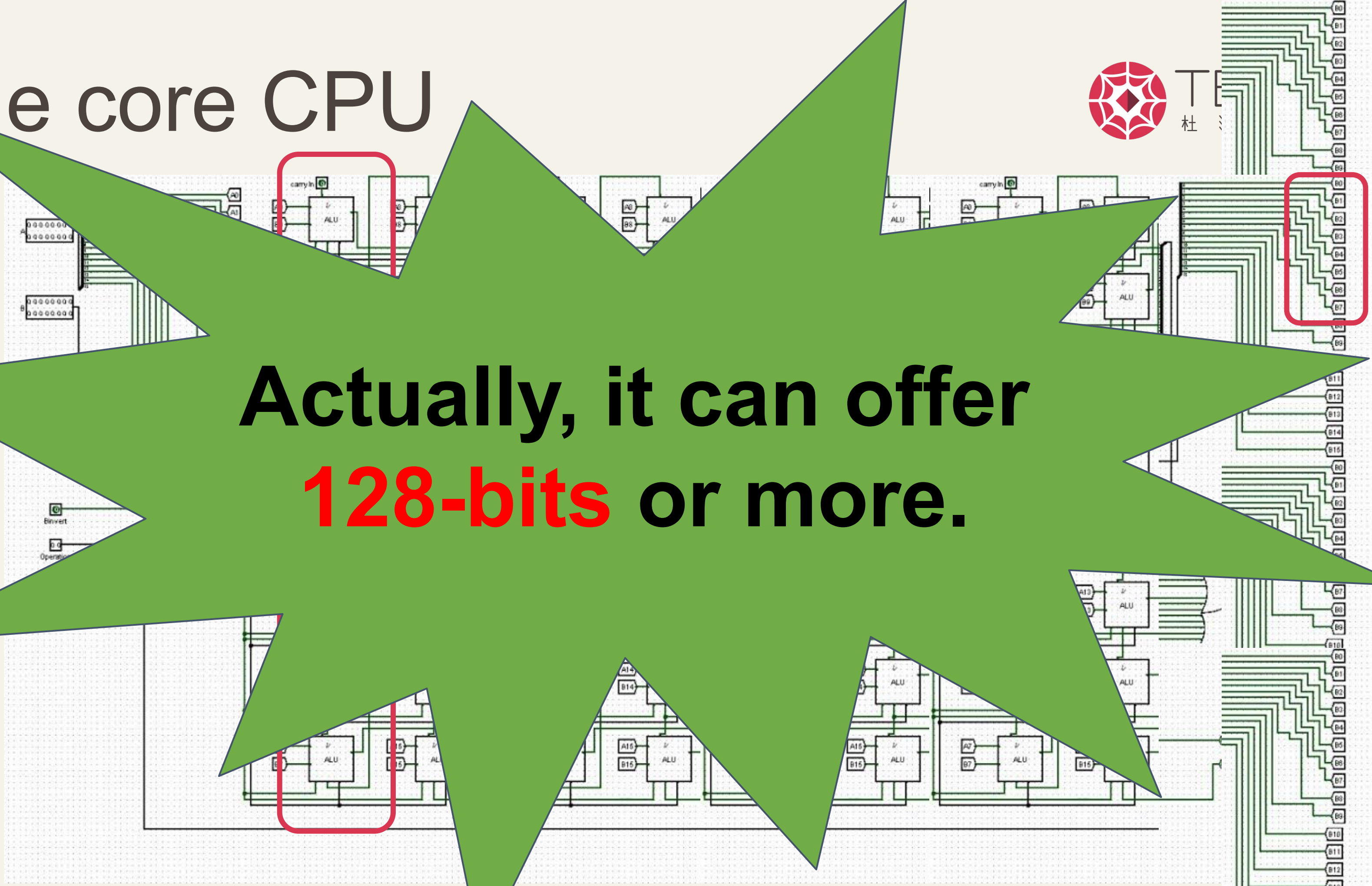
https://www.youtube.com/watch?v=-QW4txHfOyQ

Single core CPU



https://www.youtube.com/watch?v=-QW4txHfOyQ

Single core CPU



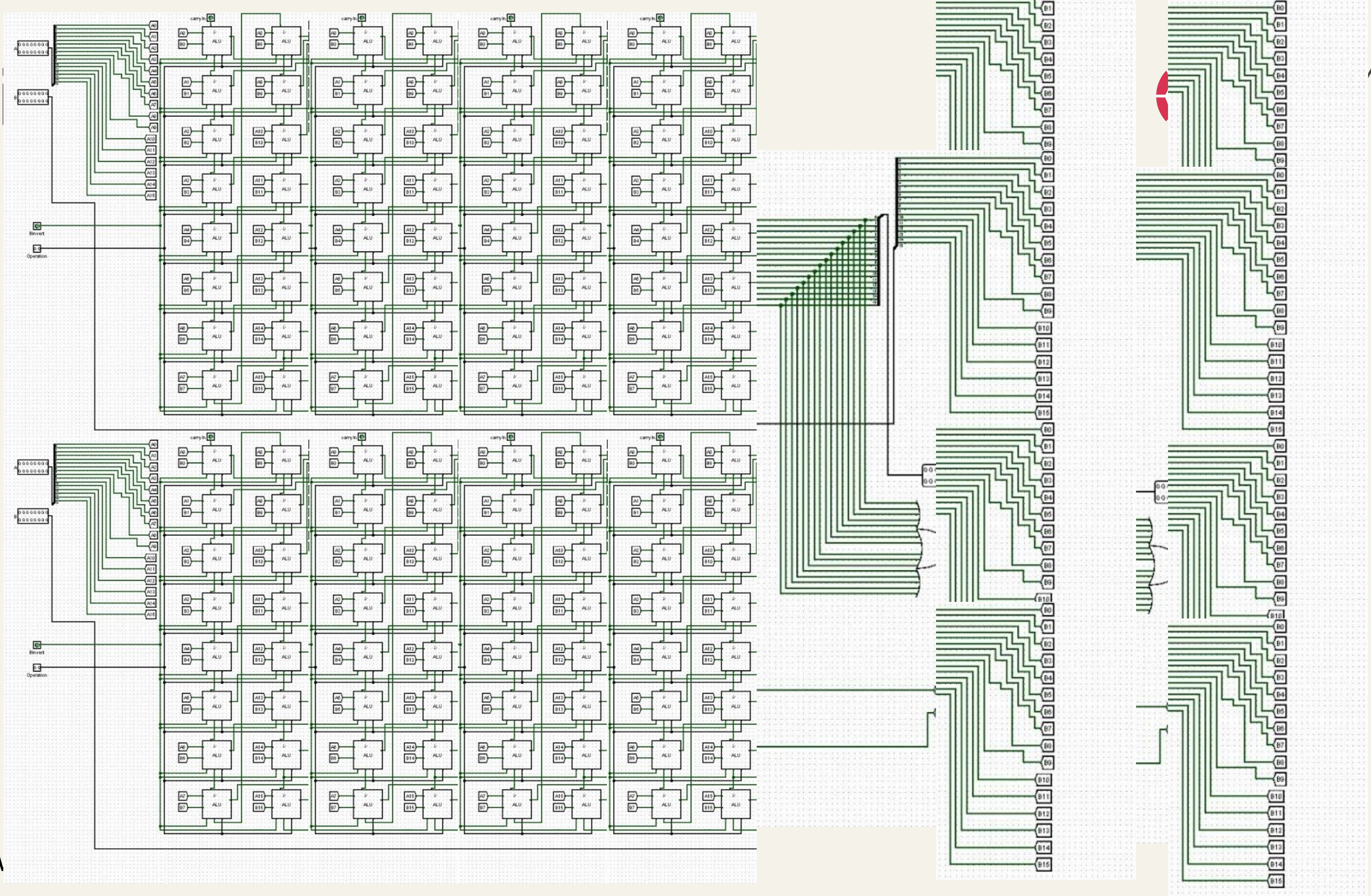
Actually, it can offer 128-bits or more.

https://www.youtube.com/watch?v=-QW4txHfOy

Intel SSE

```
while (len >= 16) {
    __m128i inp = _mm_loadu_si128((__m128i *)cstr);
    __m128i mask = _mm_cmpeq_epi8(inp, _mm_set1_epi8(c));
    int mask_int = _mm_movemask_epi8(mask);
    if (unlikely(mask_int != 0))
        return (char *)(cstr + __builtin_ctz(mask_int));
    len -= 16;
    cstr += 16;
}
```

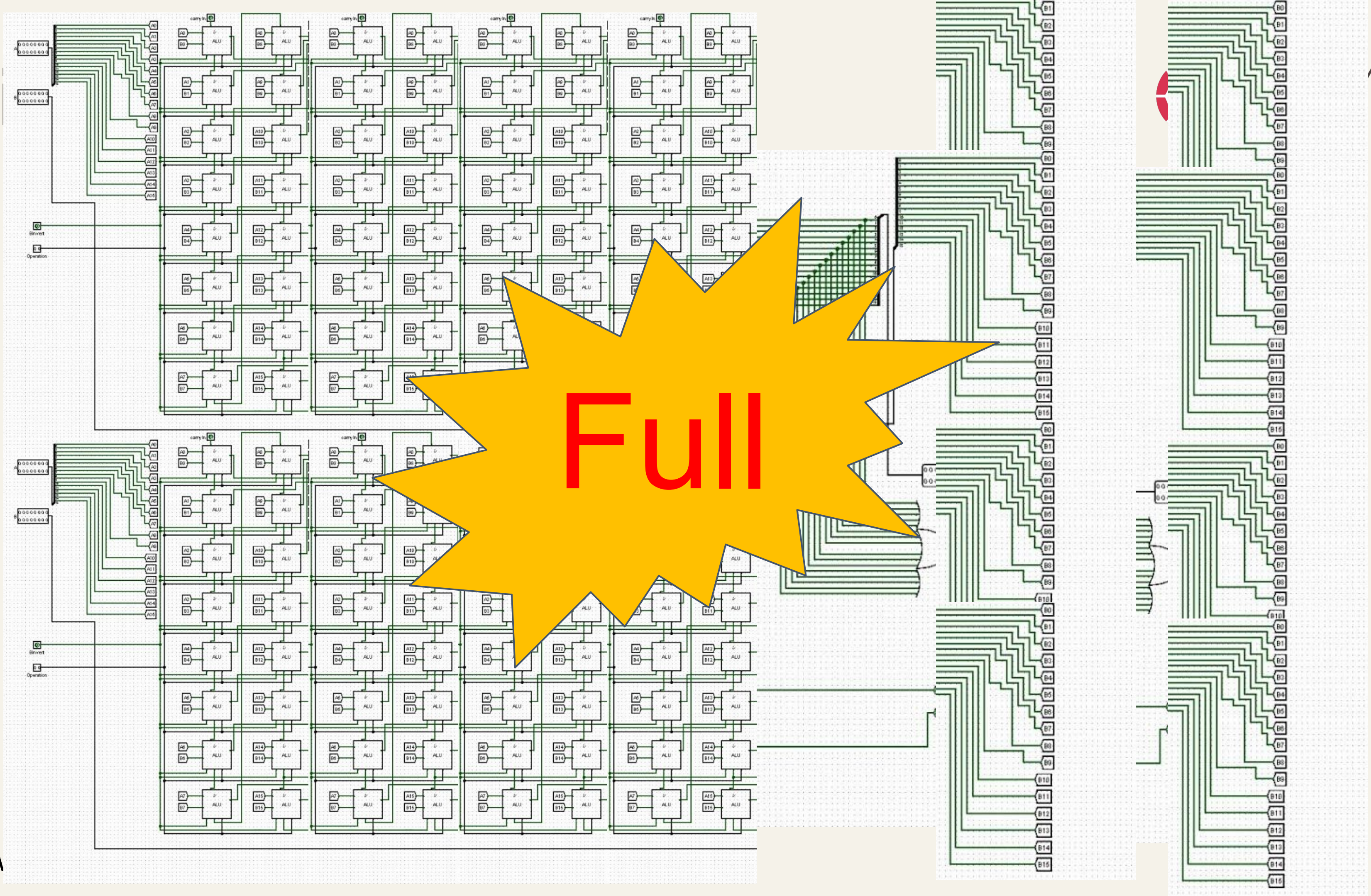
S



carry In
ALU
Binvert
Operation
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
B0
B1
B2
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15

https://\

S



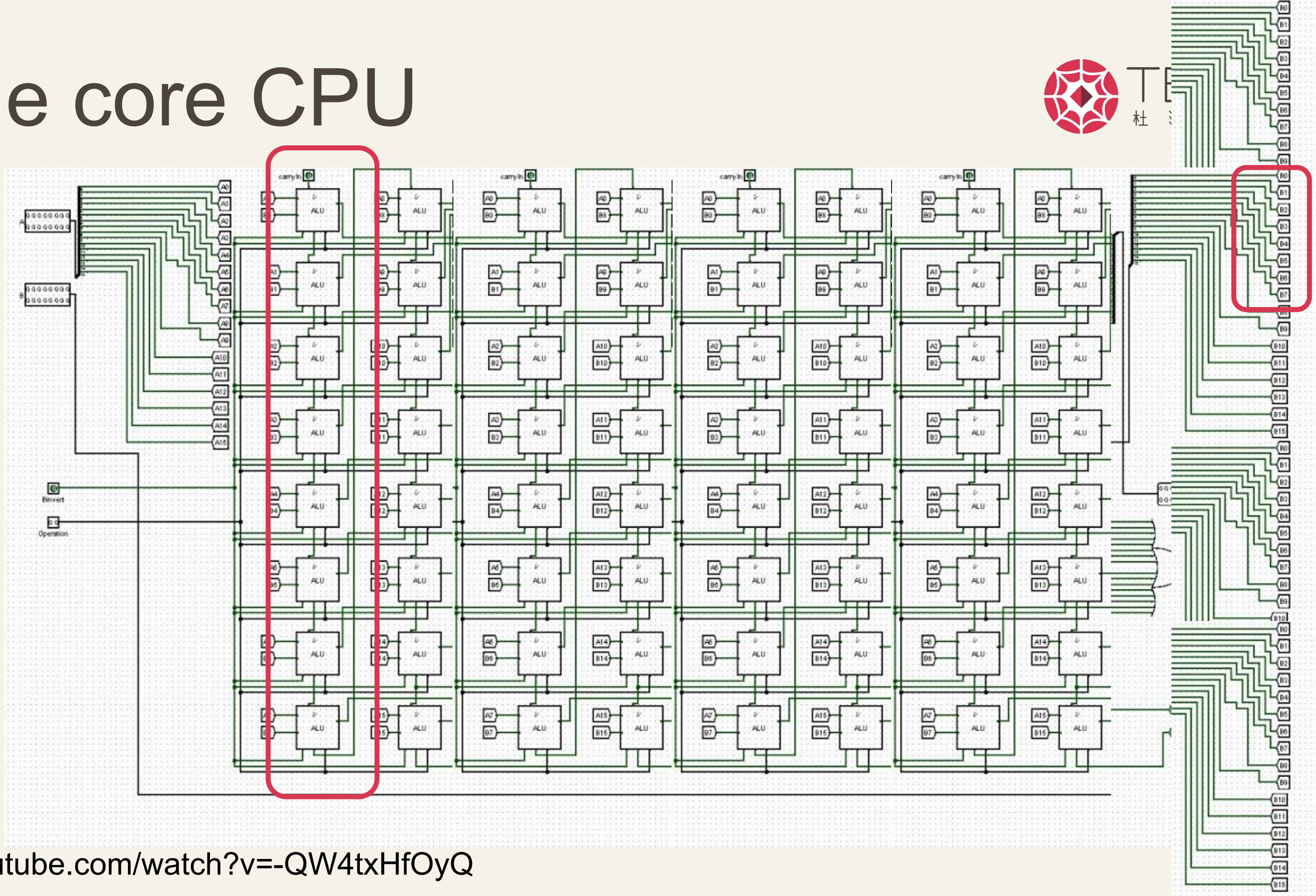
https://

1T5
安 全

What’s the problem

What if < 16?

Single core CPU



https://www.youtube.com/watch?v=-QW4txHfOyQ

int also can SIMD?!

Most of cases…

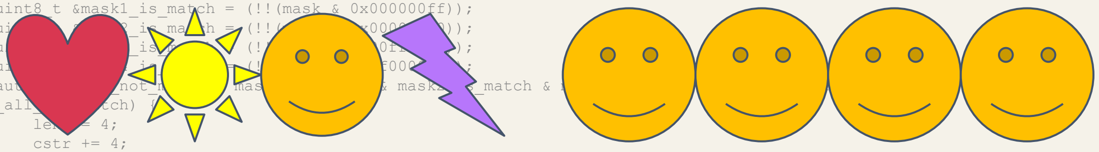
```
sizeof(char) * 4
sizeof(int)
```

```
while (len >= 4) {
     uint32_t inp = *(uint32_t *)cstr;
     uint32_t mask = inp ^ (c * 0x01010101);

     const uint8_t &mask1_is_match = (!!(mask & 0x000000ff));
     const uint8_t &mask2_is_match = (!!(mask & 0x0000ff00));
     const uint8_t &mask3_is_match = (!!(mask & 0x00ff0000));
     const uint8_t &mask4_is_match = (!!(mask & 0xff000000));
     const auto &is_all_not_match = mask1_is_match & mask2_is_match &
     mask3_is_match & mask4_is_match;
     if (is_all_not_match) {
          len -= 4;
          cstr += 4;
          continue;
     }

     if (mask1_is_match)
          return (char *)cstr;
     else if (mask2_is_match)
          return (char *)(cstr + 1);
     else if (mask3_is_match)
          return (char *)(cstr + 2);
     return (char *)(cstr + 3);
}
```

```
while (len >= 4) {
    uint32_t inp = *(uint32_t *)cstr;
    uint32_t mask = inp ^ (c * 0x01010101);

    const uint8_t &mask1_is_match = (!!(mask & 0x000000ff));
    const uint8_t &mask2_is_match = (!!(mask & 0x0000ff00));
    const uint8_t &mask3_is_match = (!!(mask & 0x00ff0000));
    const uint8_t &mask4_is_match = (!!(mask & 0xff000000));
    const auto &is_all_not_match = mask1_is_match & mask2_is_match & mask3_is_match & mask4_is_match;
    if (is_all_not_match) {
            len -= 4;
            cstr += 4;
            continue;
    }

    if (mask1_is_match)
            return (char *)cstr;
    else if (mask2_is_match)
            return (char *)(cstr + 1);
    else if (mask3_is_match)
            return (char *)(cstr + 2);
    return (char *)(cstr + 3);
}
```

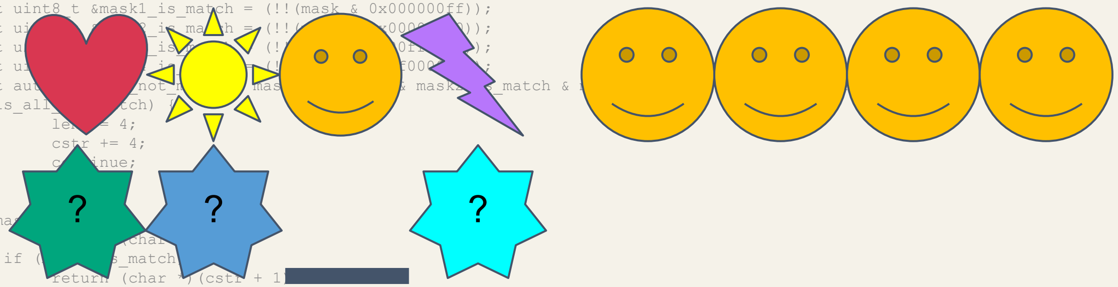
```
while (len >= 4) {
    uint32_t inp = *(uint32_t *)cstr;
    uint32_t mask = inp ^ (c * 0x01010101);

    const uint8_t &mask1_is_match = (!!(mask & 0x000000ff));
    const uint8_t &mask2_is_match = (!!(mask & 0x0000ff00));
    const uint8_t &mask3_is_match = (!!(mask & 0x00ff0000));
    const uint8_t &mask4_is_match = (!!(mask & 0xff000000));
    const auto &is_all_not_match = mask1_is_match & mask2_is_match & 
    if (is_all_not_match) {
            len -= 4;
            cstr += 4;
            continue;
    }

    if (mask1_is_match)
            return (char *)cstr;
    else if (mask2_is_match)
            return (char *)(cstr + 1);
    else if (mask3_is_match)
            return (char *)(cstr + 2);
    return (char *)(cstr + 3);
}
```

```
while (len >= 4) {
    uint32_t inp = *(uint32_t *)cstr;
    uint32_t mask = inp ^ (c * 0x01010101);
```



```
        const uint8_t &mask1_is_match = (!!(mask & 0x000000ff));
        const ui[illegible]2_is_ma[illegible]h = (!!([illegible]x0000[illegible]0));
        const u[illegible]is_m[illegible] (![illegible]0ff[illegible]);
        const u[illegible]i[illegible]= (![illegible]f000[illegible]);
        const au[illegible]not_m[illegible]mas[illegible]& mask2[illegible]s_match & [illegible]
        if (is_all[illegible]tch) {
                le[illegible]= 4;
                cstr += 4;
                continue;
        }

        if (mask1_is_match)
                return (char *)cstr;
        else if (mask2_is_match)
                return (char *)(cstr + 1);
        else if (mask3_is_match)
                return (char *)(cstr + 2);
        return (char *)(cstr + 3);
}
```

```
while (len >= 4) {
    uint32_t inp = *(uint32_t *)cstr;
    uint32_t mask = inp ^ (c * 0x01010101);
```

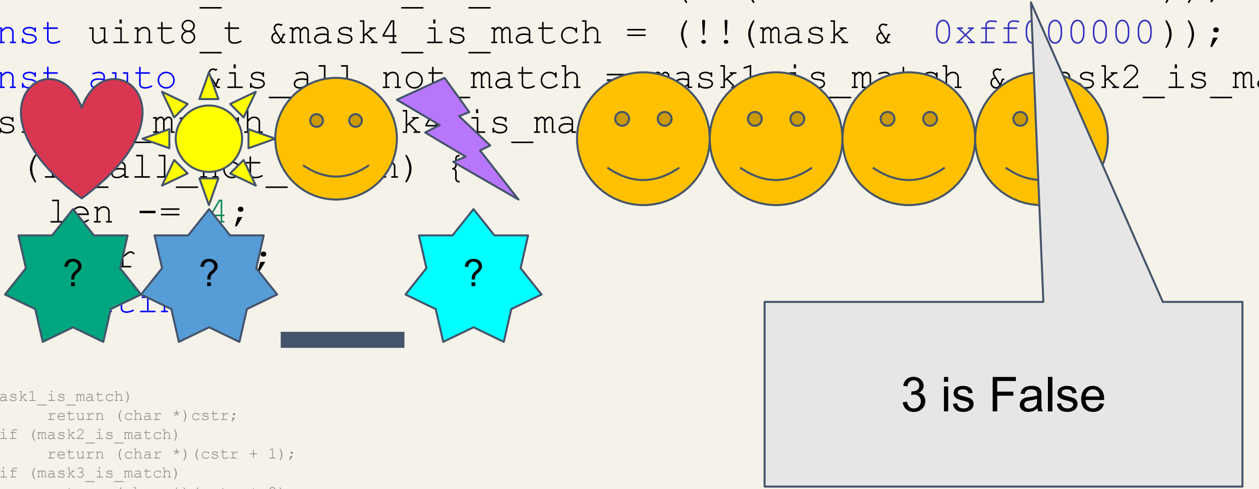


```
    const uint8_t &mask1_is_match = (!!(mask & 0x000000ff));
    const ui          2_is_match = (!!(      x0000    0));
    const u              _is        (!!         0ff     );
    const u              is           (!         f000     );
    const aut         _not_m       mas           & mask2  s_match & 
    if (is_all_       tch) {
            le    = 4;
            cstr += 4;
            c     inue;
    }

    if (mas
                    (char
    else if (        s_match
            return (char *)(cstr + 1)
    else if (mask3_is_match)
            return (char *)(cstr + 2);
    return (char *)(cstr + 3);
}
```

```
while (len >= 4) {
        uint32_t inp = *(uint32_t *)cstr;
        uint32_t mask = inp ^ (c * 0x01010101);

        const uint8_t &mask1_is_match = (!!(mask &  0x000000ff));
        const uint8_t &mask2_is_match = (!!(mask &  0x0000ff00));
        const uint8_t &mask3_is_match = (!!(mask &  0x00ff0000));
        const uint8_t &mask4_is_match = (!!(mask &  0xff000000));
        const auto &is_all_not_match = mask1_is_match & mask2_is_match &
        mask3_is_match & mask4_is_match;
        if (is_all_not_match) {
                len -= 4;
                cstr += 4;
                continue;
        }

        if (mask1_is_match)
                return (char *)cstr;
        else if (mask2_is_match)
                return (char *)(cstr + 1);
        else if (mask3_is_match)
                return (char *)(cstr + 2);
        return (char *)(cstr + 3);
}
```

0 if that pos is 0

```
while (len >= 4) {
        uint32_t inp = *(uint32_t *)cstr;
        uint32_t mask = inp ^ (c * 0x01010101);

        const uint8_t &mask1_is_match = (!!(mask &  0x000000ff));
        const uint8_t &mask2_is_match = (!!(mask &  0x0000ff00));
        const uint8_t &mask3_is_match = (!!(mask &  0x00ff0000));
        const uint8_t &mask4_is_match = (!!(mask &  0xff000000));
        const auto &is_all_not_match = mask1_is_match & mask2_is_match &
        mas      m  ch    k4_is_ma
        if (i _all_not_    n) {
            len -= 4;
               r      ;
                tin
        }

        if (mask1_is_match)
                return (char *)cstr;
        else if (mask2_is_match)
                return (char *)(cstr + 1);
        else if (mask3_is_match)
                return (char *)(cstr + 2);
        return (char *)(cstr + 3);
}
```



```
while (len >= 4) {
        uint32_t inp = *(uint32_t *)cstr;
        uint32_t mask = inp ^ (c * 0x01010101);

        const uint8_t &mask1_is_match = (!!(mask &  0x000000ff));
        const uint8_t &mask2_is_match = (!!(mask &  0x0000ff00));
        const uint8_t &mask3_is_match = (!!(mask &  0x00ff0000));
        const uint8_t &mask4_is_match = (!!(mask &  0xff000000));
        const auto &is_all_not_match = mask1_is_match & mask2_is_match &
        mask3_is_match & mask4_is_match;
        if (is_all_not_match) {
                len -= 4;
                cstr += 4;
                continue;
        }

        if (mask1_is_match)
                return (char *)cstr;
        else if (mask2_is_match)
                return (char *)(cstr + 1);
        else if (mask3_is_match)
                return (char *)(cstr + 2);
        return (char *)(cstr + 3);
}
```

True & True & False & True

```
while (len >= 4) {
        uint32_t inp = *(uint32_t *)cstr;
        uint32_t mask = inp ^ (c * 0x01010101);

        const uint8_t &mask1_is_match = (!!(mask &  0x000000ff));
        const uint8_t &mask2_is_match = (!!(mask &  0x0000ff00));
        const uint8_t &mask3_is_match = (!!(mask &  0x00ff0000));
        const uint8_t &mask4_is_match = (!!(mask &  0xff000000));
        const auto &is_all_not_match = mask1_is_match & mask2_is_match &
        mask3_is_match & mask4_is_match;
        if (is_all_not_match) {
                len -= 4;
                cstr += 4;
                continue;
        }

        if (mask1_is_match)
                return (char *)cstr;
        else if (mask2_is_match)
                return (char *)(cstr + 1);
        else if (mask3_is_match)
                return (char *)(cstr + 2);
        return (char *)(cstr + 3);
}
```

False

```
while (len >= 4) {
        uint32_t inp = *(uint32_t *)cstr;
        uint32_t mask = inp ^ (c * 0x01010101);

        const uint8_t &mask1_is_match = (!!(mask &  0x000000ff));
        const uint8_t &mask2_is_match = (!!(mask &  0x0000ff00));
        const uint8_t &mask3_is_match = (!!(mask &  0x00ff0000));
        const uint8_t &mask4_is_match = (!!(mask &  0xff000000));
        const auto &is all_not_match = mask1_is_match & mask2_is_match &
        mask3_is_matc  & mask4_is_match;
        if (is_all_   match) {
             len -
             cst
             co

             return (char *)(cstr + 2);
        return (char *)(cstr + 3);
}
```

Why 4 variable?

Collapsing them into an int is better?

```
while (len >= 4) {
    uint32_t inp = *(uint32_t *)cstr;
    uint32_t mask = inp ^ (c * 0x01010101);

    const uint8_t &mask1_is_match = (!!(mask &  0x000000ff));
    const uint8_t &mask2_is_match = (!!(mask &  0x0000ff00));
    const uint8_t &mask3_is_match = (!!(mask &  0x00ff
    const uint8_t &mask4_is_match = (!!(mask &  0xff000
    const auto &is  all_not_match = mask1_is_match & ma
    mask3_is_matc  & mask4_is_match;
    if (is_all_    match) {
        len -
        cst
        co
        return (char *)(cstr + 2);
    return (char *)(cstr + 3);
}
```

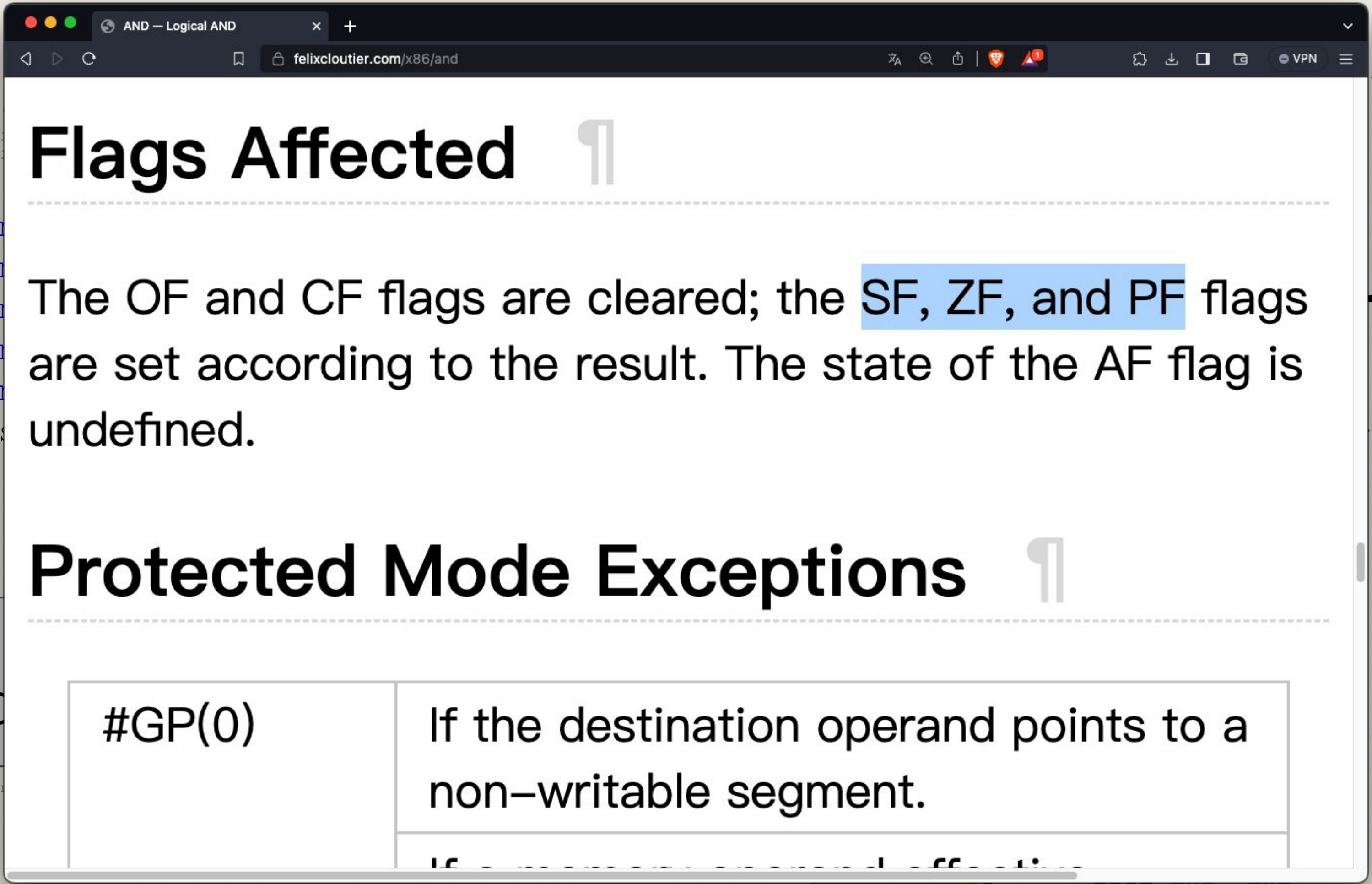
Why 4 variable?
Collapsing them into an int is better?

OoOE Friendly

```
.L2:
        cmp rdx, 3
        jbe .L6
        mov ecx, DWORD PTR [rax]
        xor ecx, 707406378
        mov esi, ecx
        movzx edi, cl
        and esi, 65280
        cmp ecx, 16777215
        seta r9b
        and ecx, 16711680
        setne r8b
        test r9b, r8b
        je .L7
        test esi, esi
        setne r9b
        test edi, edi
        setne r8b
        test r9b, r8b
        jne .L39
```

AMT5 數 位 安 全

```
while
    uint3
    uint3

    co
    co
    co
    co
    co
    ma
    if

    C

    retur
}
```



Flags Affected ¶

The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined.

Protected Mode Exceptions ¶

#GP(0)	If the destination operand points to a non-writable segment.

```
[rax]
test r9b, r8b
jne .L39
```

https://www.felixcloutier.com/x86/and

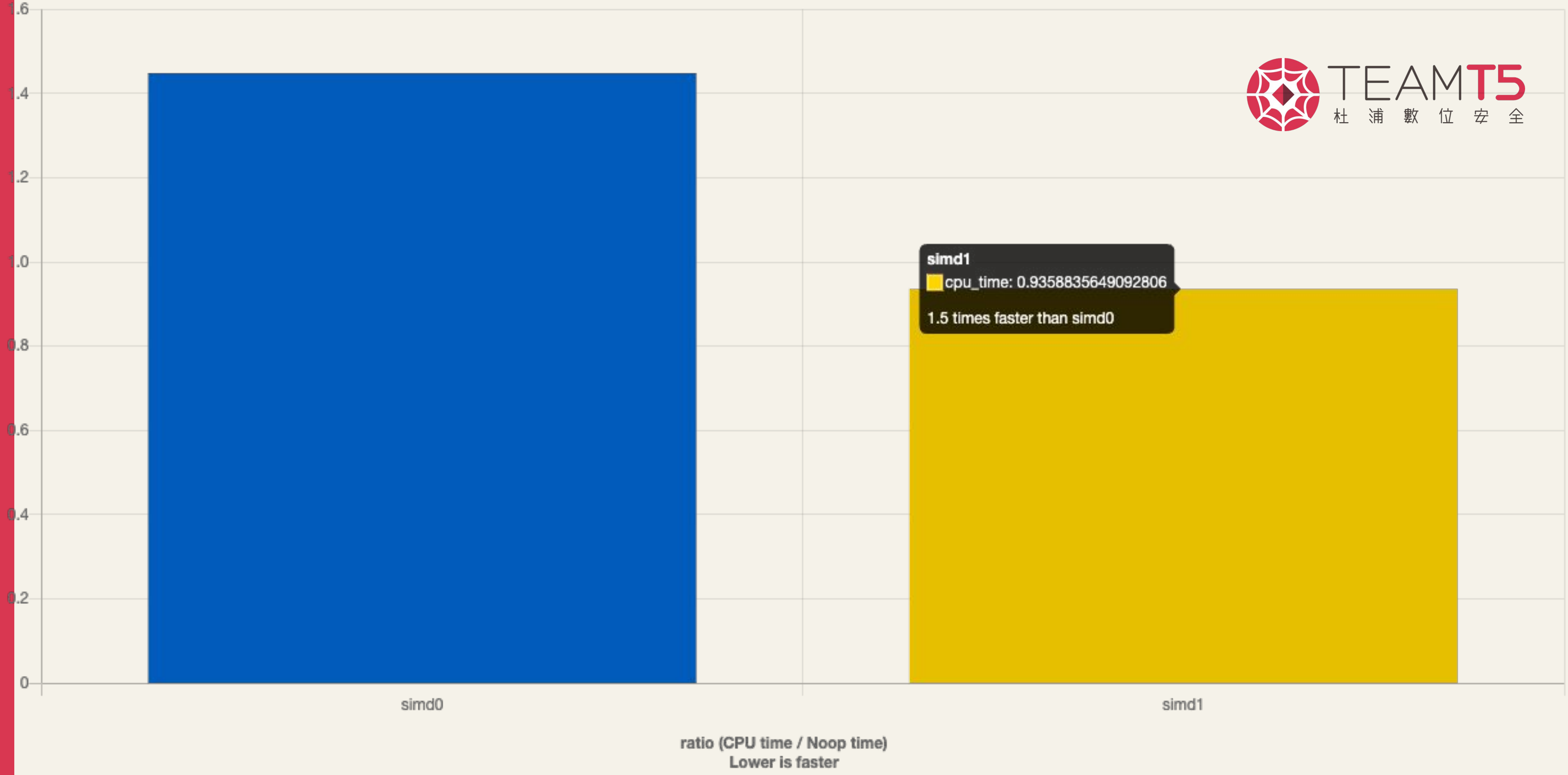
```
while (len >= 4) {
    uint32_t inp = *(uint32_t *)cstr;
    uint32_t mask = inp ^ (c * 0x01010101);

    const uint8_t &mask1_is_match = (!!(mask & 0x000000ff));
    const uint8_t &mask2_is_match = (!!(mask & 0x0000ff00));
    const uint8_t &mask3_is_match = (!!(mask & 0x00ff0000));
    const uint8_t &mask4_is_match = (!!(mask & 0xff000000));
    const auto &is_all_not_match = mask1_is_match & mask2_is_match & mask3_is_match & mask4_is_match;
    if (is_all_not_match) {
        len -= 4;
        cstr += 4;
        continue;
    }

    if (mask1_is_match)
        return (char *)cstr;
    else if (mask2_is_match)
        return (char *)(cstr + 1);
    else if (mask3_is_match)
        return (char *)(cstr + 2);
    return (char *)(cstr + 3);
}
```

```
while (len >= 4) {
    uint32_t inp = *(uint32_t *)cstr;
    uint32_t mask = inp ^ (c * 0x01010101);

    const uint8_t &mask1_is_match = (!!(mask & 0x000000ff));
    const uint8_t &mask2_is_match = (!!(mask & 0x0000ff00));
    const uint8_t &mask3_is_match = (!!(mask & 0x00ff0000));
    const uint8_t &mask4_is_match = (!!(mask & 0xff000000));
    const auto &is_all_not_match = mask1_is_match & mask2_is_match & mask3_is_match & mask4_is_match;
    if (is_all_not_match) {
        len -= 4;
        cstr += 4;
        continue;
    }

    if (mask1_is_match)
        return (char *)cstr;
    else if (mask2_is_match)
        return (char *)(cstr + 1);
    else if (mask3_is_match)
        return (char *)(cstr + 2);
    return (char *)(cstr + 3);
}
```

Why if-else here?

I have no idea currently, it ran faster on my laptop and PC.

Maybe it can be calculated parallelly.



https://quick-bench.com/q/6JaCKIyOQ_-AF7WGFmCDLP7WXLg

Thank you for your listening.

scc@teamt5.org

TEAMT5
杜 浦 數 位 安 全
Persistent Cyber Threat Hunters