

RISC-V GCC possible
inefficiencies

OPUS analysis (<celt_decode_lost>)

- The **function** is **big** and performs a lot of operations
- Hard to precisely make a comparison, but... we can note something!
- **ARM** and **GCC** code have approx the **same number** of **mem ops** (250)
- The **ratio** of **compressed/uncompressed memory ops** is the **same**
- **ARM** always **keeps** the **same register** as **base address** for **+65%** of **memory ops**. This **reg** is **never modified**
- **RISC-V** uses **multiple regs**, every time **modifying** them

OPUS analysis (<celt_decode_lost>)

```
14aca: 6f3b .....ldr r3, [r7, #112] ; 0x70
14acc: 6d7c .....ldr r4, [r7, #84] ; 0x54
14ad0: f107 019c ...add.w r1, r7, #156 ; 0x9c
```

ARM code: 173 mem ops on 256 use r7 as base reg.
r7 is loaded in the beginning and never modified.

```
1dfd8: 777d .....lui a4, 0xffffffff
1dfda: ec870793 .....addi a5, a4, -312
1dfde: 97a2 .....add a5, a5, s0
1dfe0: 438c .....lw a1, 0(a5)
1dfe2: ed470793 .....addi a5, a4, -300
1dfe6: ec070713 .....addi a4, a4, -320
1dfea: 97a2 .....add a5, a5, s0
1dfec: 9722 .....add a4, a4, s0
1dfee: 439c .....lw a5, 0(a5)
1dff0: 4318 .....lw a4, 0(a4)
```

HCC code: before a memory operation, the code loads a value into a reg and adds an offset.

OPUS analysis (<celt_decode_lost>)

- The way **GCC** schedules the **operations** seems **redundant**.
- In this fragment of code, the **net effect** is to **load** values in **a1, a4, a5**.

```
1dfd8: 777d ..... lui a4,0xffffffff
1dfda: ec870793 ..... addi a5,a4,-312
1dfde: 97a2 ..... add a5,a5,s0
1dfe0: 438c ..... lw a1,0(a5)
1dfe2: ed470793 ..... addi a5,a4,-300
1dfe6: ec070713 ..... addi a4,a4,-320
1dfea: 97a2 ..... add a5,a5,s0
1dfec: 9722 ..... add a4,a4,s0
1dfee: 439c ..... lw a5,0(a5)
1dff0: 4318 ..... lw a4,0(a4)
```

Fragment of <celt_decode_lost> function (compiled with GCC)

OPUS analysis (<celt_decode_lost>)

- The **addresses** are **similar**: we can use only one register as base address

```

1dfd8: 777d.....lui a4,0xffffffff
1dfda: ec870793.....addi a5,a4,-312
1dfde: 97a2.....add a5,a5,s0
1dfe0: 438c.....lw a1,0(a5)
1dfe2: ed470793.....addi a5,a4,-300
1dfe6: ec070713.....addi a4,a4,-320
1dfea: 97a2.....add a5,a5,s0
1dfec: 9722.....add a4,a4,s0
1dfee: 439c.....lw a5,0(a5)
1dff0: 4318.....lw a4,0(a4)

```

Original GCC code

```

1dfd8: 777d.....lui a4,0xffffffff
yyyyy: xxxxxxxx.....addi a4,a4,-320
yyyyy: xxxx.....add a4,a4,s0
yyyyy: xxxx.....lw a1,8(a4)
yyyyy: xxxx.....lw a5,20(a4)
yyyyy: xxxx.....lw a4,0(a4)

```

Manually optimized code

OPUS analysis (<celt_decode_lost>)

- Another portion of the code (GCC 9.2 does not solve the issue):

```

1dffc: 76fd.....lui a3,0xfffff
1dffe: ed468793.....addi a5,a3,-300 #
1e002: ec068713.....addi a4,a3,-320
1e006: 97a2.....add a5,a5,s0
1e008: 9722.....add a4,a4,s0
1e00a: 4318.....lw a4,0(a4)
1e00c: 439c.....lw a5,0(a5)
1e00e: 89b6.....mv s3,a3
1e010: e9c98613.....addi a2,s3,-356
1e014: 60f717db.....mulhiadd a5,a4,a5,96
1e018: eb068713.....addi a4,a3,-336
1e01c: 9722.....add a4,a4,s0
1e01e: ed868693.....addi a3,a3,-296
1e022: eb098593.....addi a1,s3,-336
1e026: eb498513.....addi a0,s3,-332
1e02a: c31c.....sw a5,0(a4)
1e02c: 96a2.....add a3,a3,s0
1e02e: 9622.....add a2,a2,s0
1e030: 95a2.....add a1,a1,s0
1e032: 9522.....add a0,a0,s0
1e034: 50dc.....lw a5,36(s1)
1e036: 4294.....lw a3,0(a3)
1e038: 4210.....lw a2,0(a2)
1e03a: 418c.....lw a1,0(a1)
1e03c: 4108.....lw a0,0(a0)
1e03e: 4761.....li a4,24

```

Original GCC code

Manual reordering



```

1dffc: 76fd.....lui a3,0xfffff
1dffe: ed468793.....addi a5,a3,-300 #
1e002: ec068713.....addi a4,a3,-320
1e006: 97a2.....add a5,a5,s0
1e008: 9722.....add a4,a4,s0
1e00a: 4318.....lw a4,0(a4)
1e00c: 439c.....lw a5,0(a5)
1e014: 60f717db.....mulhiadd a5,a4,a5,96
1e01e: eb068713.....addi a4,s3,-336
1e018: eb068713.....addi a4,s3,-336
1e01e: ed868693.....addi a3,s3,-296
1e010: e9c98613.....addi a2,s3,-356
1e022: eb098593.....addi a1,s3,-336
1e026: eb498513.....addi a0,s3,-332
1e01c: 9722.....add a4,a4,s0
1e02c: 96a2.....add a3,a3,s0
1e02e: 9622.....add a2,a2,s0
1e030: 95a2.....add a1,a1,s0
1e032: 9522.....add a0,a0,s0
1e02a: c31c.....sw a5,0(a4)
1e034: 50dc.....lw a5,36(s1)
1e036: 4294.....lw a3,0(a3)
1e038: 4210.....lw a2,0(a2)
1e03a: 418c.....lw a1,0(a1)
1e03c: 4108.....lw a0,0(a0)
1e03e: 4761.....li a4,24

```

Equivalent code

OPUS analysis (<celt_decode_lost>)

```

1dff: 76fd ..... lui a3,0xffff
1dfe: ed468793 ..... addi a5,a3,-300 #
1e00: ec068713 ..... addi a4,a3,-320
1e06: 97a2 ..... add a5,a5,s0
1e08: 9722 ..... add a4,a4,s0
1e0a: 4318 ..... lw a4,0(a4)
1e0c: 439c ..... lw a5,0(a5)
1e14: 60f717db ..... muliadd a5,a4,a5,96
1e0e: 89b6 ..... mv s3,a3
1e18: eb068713 ..... addi a4,s3,-336
1e1e: ed868693 ..... addi a3,s3,-296
1e10: e9c98613 ..... addi a2,s3,-356
1e22: eb098593 ..... addi a1,s3,-336
1e26: eb498513 ..... addi a0,s3,-332
1e1c: 9722 ..... add a4,a4,s0
1e2c: 96a2 ..... add a3,a3,s0
1e2e: 9622 ..... add a2,a2,s0
1e30: 95a2 ..... add a1,a1,s0
1e32: 9522 ..... add a0,a0,s0
1e2a: c31c ..... sw a5,0(a4)
1e34: 50dc ..... lw a5,36(s1)
1e36: 4294 ..... lw a3,0(a3)
1e38: 4210 ..... lw a2,0(a2)
1e3a: 418c ..... lw a1,0(a1)
1e3c: 4108 ..... lw a0,0(a0)
1e3e: 4761 ..... li a4,24

```

Equivalent code

- The memory operations are:
 - `lw rd, 0(0xffff000 - off + s0)`
- **off** is:
 - 296
 - 300
 - 320
 - 332
 - 336
 - 336
 - 356
- **off** is always in the range [296, 356]
- The **sequence** of “addi + add” is **repeated** for **each register** (a0, a1, a2, a3, a4)
- But with the **load/store** operations we can **add** an **offset** **in place!!!**

OPUS analysis (<celt_decode_lost>:)

- Strange compiler choices (GCC 9.2 produces similar code)

```

1dfffc: 76fd.....lui a3,0xffffffff
1dfffe: ed468793.....addi a5,a3,-300 #
1e0002: ec068713.....addi a4,a3,-320
1e0006: 97a2.....add a5,a5,s0
1e0008: 9722.....add a4,a4,s0
1e000a: 4318.....lw a4,0(a4)
1e000c: 439c.....lw a5,0(a5)
1e0014: 60f717db.....muladd a5,a4,a5,96
1e001e: 89b6.....mv s3,a3
1e0118: eb068713.....addi a4,s3,-336
1e011e: ed868693.....addi a3,s3,-296
1e010: e9c98613.....addi a2,s3,-356
1e022: eb098593.....addi a1,s3,-336
1e026: eb498513.....addi a0,s3,-332
1e01c: 9722.....add a4,a4,s0
1e02c: 96a2.....add a3,a3,s0
1e02e: 9622.....add a2,a2,s0
1e030: 95a2.....add a1,a1,s0
1e032: 9522.....add a0,a0,s0
1e02a: c31c.....sw a5,0(a4)
1e034: 50dc.....lw a5,36(s1)
1e036: 4294.....lw a3,0(a3)
1e038: 4210.....lw a2,0(a2)
1e03a: 418c.....lw a1,0(a1)
1e03c: 4108.....lw a0,0(a0)
1e03e: 4761.....li a4,24

```

Equivalent code

- ARM: 40 B
- HCC: **68 B**
- Manually optimized: **30 B**

Manual optimization



```

1dfffc: 76fd.....lui a0,0xffffffff
yyyyy: xxxx.....add a0,a0,s0
yyyyy: xxxxxxxx.....addi a0,a0,-356
1e00c: 439c.....lw a5,56(a0)
1e00a: 4318.....lw a4,36(a0)
1e014: 60f717db.....muladd a5,a4,a5,96
1e02a: c31c.....sw a5,20(a0)
1e034: 50dc.....lw a5,36(s1)
1e036: 4294.....lw a3,60(a0)
1e038: 4210.....lw a2,0(a0)
1e03a: 418c.....lw a1,20(a0)
1e03c: 4108.....lw a0,24(a0)
1e03e: 4761.....li a4,24

```

Equivalent code

OPUS analysis (<celt_decode_lost>:)

```

1dfd8: 777d ..... lui a4,0xffff
yyyyy: xxxxxxxx ..... addi a4, a4, -320
yyyyy: xxxx ..... add a4, a4, s0
yyyyy: xxxx ..... lw a1, 8(a4)
yyyyy: xxxx ..... lw a5, 20(a4)
yyyyy: xxxx ..... lw a4, 0(a4)

```

```

1dffc: 76fd ..... lui a0,0xffff
yyyyy: xxxx ..... add a0, a0, s0
yyyyy: xxxxxxxx ..... addi a0, a0, -356
1e00c: 439c ..... lw a5, 56(a0)
1e00a: 4318 ..... lw a4, 36(a0)
1e014: 60f717db ..... muliadd a5, a4, a5, 96
1e02a: c31c ..... sw a5, 20(a0)
1e034: 50dc ..... lw a5, 36(s1)
1e036: 4294 ..... lw a3, 60(a0)
1e038: 4210 ..... lw a2, 0(a0)
1e03a: 418c ..... lw a1, 20(a0)
1e03c: 4108 ..... lw a0, 24(a0)
1e03e: 4761 ..... li a4, 24

```

- These are the **two** manually **optimized fragments** of code.
- $s0 = sp + \text{fixed_offset}$
- After the initial assignment, **s0** is **never modified**.
- We always **load 0xfffff**, then **add s0** and a **variable offset**.
- This **pattern** is **frequent** into the function (it seems >60 times).
- The **variable offsets** are very **similar** also for **different fragments**.



- A **fixed register** for the **whole function** as a **base** for the memory operations (as ARM does) can be **convenient**.

Toy Examples

- Reproduced the problem with 2 different toy-examples

Toy Example 0: stack ping-pong

```
#include <stdio.h>

#define M 2000
#define N 1000

extern void foo(int *arr);

int main(int argc, char **argv) {
    int arr_0[N]; // Loaded on the stack
    int arr_1[M]; // Loaded on the stack

    foo(arr_0);
    foo(arr_1);

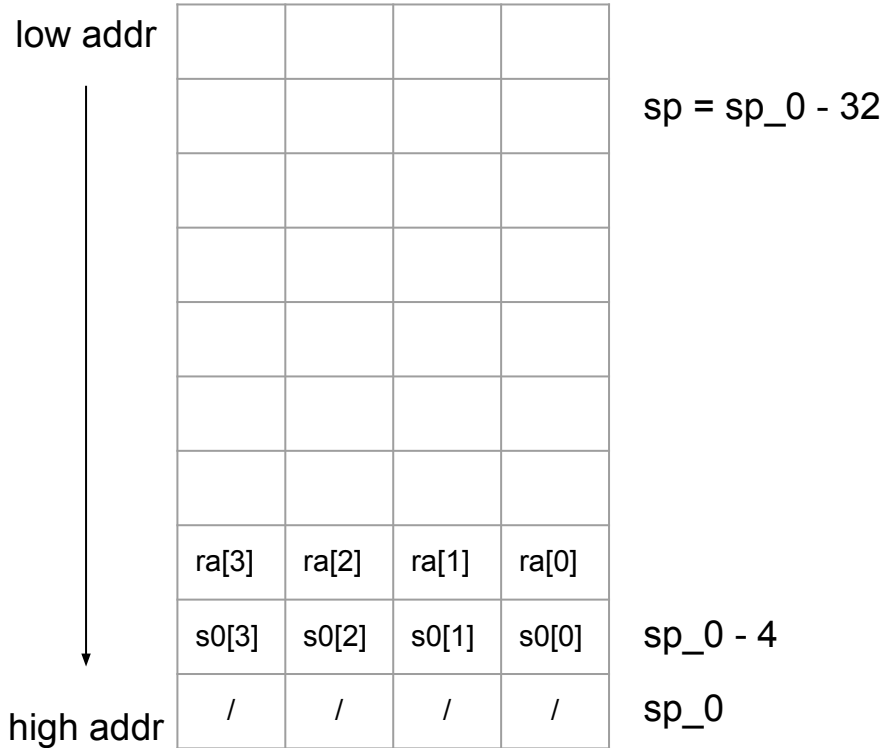
    return 0;
}
```

toy_example_0.c

```
00000000 <main>:
0: 1101 ..... addi sp, sp, -32
4: cc22 ..... sw s0, 24(sp)
c: ce06 ..... sw ra, 28(sp)
2: 7375 ..... lui t1, 0xffffd
6: 13030313 ..... addi t1, t1, 304 # fffffd130 <main+0xffffd130>
12: 911a ..... add sp, sp, t1
a: 640d ..... lui s0, 0x3
e: ee040793 ..... addi a5, s0, -288 # 2ee0 <main+0x2ee0>
16: 978a ..... add a5, a5, sp
14: 7575 ..... lui a0, 0xffffd
18: 12050513 ..... addi a0, a0, 288 # fffffd120 <main+0xffffd120>
1c: 953e ..... add a0, a0, a5
1e: 00000097 ..... auipc ra, 0x0
    1e: R_RISCV_CALL foo
    1e: R_RISCV_RELAX *ABS*
22: 000080e7 ..... jalr ra # 1e <main+0x1e>
26: ee040793 ..... addi a5, s0, -288
2c: 978a ..... add a5, a5, sp
2a: 7579 ..... lui a0, 0xfffffe
2e: 0c050513 ..... addi a0, a0, 192 # fffff0c0 <main+0xfffffe0c0>
32: 953e ..... add a0, a0, a5
34: 00000097 ..... auipc ra, 0x0
    34: R_RISCV_CALL foo
    34: R_RISCV_RELAX *ABS*
38: 000080e7 ..... jalr ra # 34 <main+0x34>
3c: 630d ..... lui t1, 0x3
3e: ed030313 ..... addi t1, t1, -304 # 2ed0 <main+0x2ed0>
42: 911a ..... add sp, sp, t1
44: 40f2 ..... lw ra, 28(sp)
46: 4462 ..... lw s0, 24(sp)
48: 4501 ..... li a0, 0
4a: 6105 ..... addi sp, sp, 32
4c: 8082 ..... ret
```

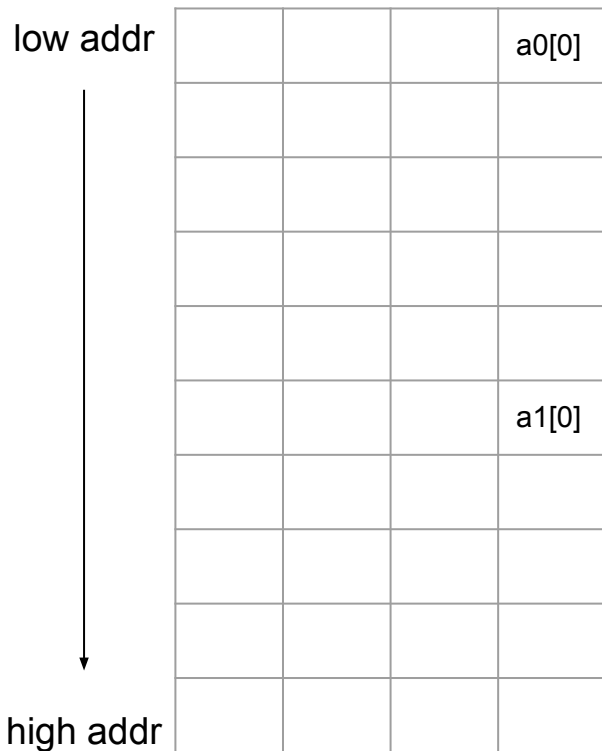
toy_example_0.s

Toy Example 0: stack ping-pong



- Push saved regs on the stack
- Align to 128-bit boundary, i.e. 32 B (ABI)

Toy Example 0: stack ping-pong



$sp = sp_0 - 12016$


- Push saved regs on the stack
- Align to 128-bit boundary, i.e. 32 B (ABI)
- Allocate stack space for both the arrays (12000 B)

$sp - 4000$

sp_0

Toy Example 0: stack ping-pong

low addr



high addr

			a0[0]
			a1[0]

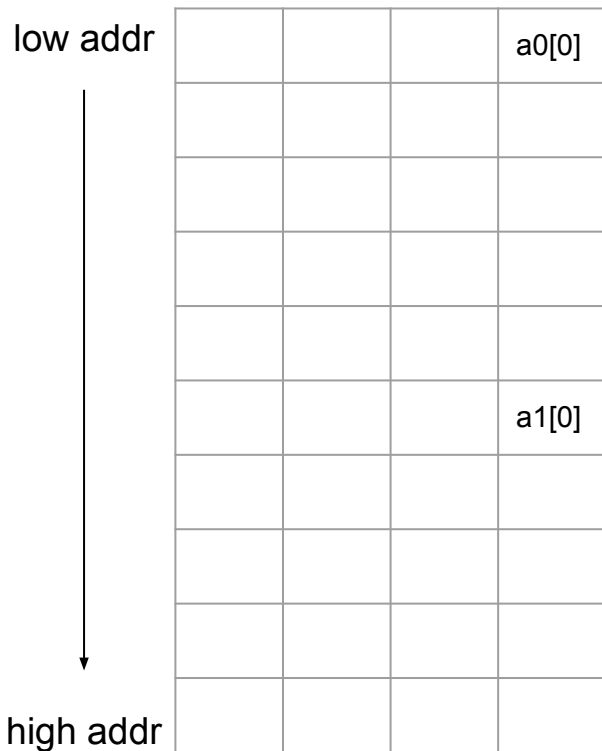
$sp = sp_0 - 12016$

- Push saved regs on the stack
- Align to 128-bit boundary, i.e. 32 B (ABI)
- Allocate stack space for both the arrays (12000 B)
- Put in a0 the pointer to the first vector:
 $a0 = sp + 12288 - 288 - 12288 + 288 = sp$

$sp - 4000$

sp_0

Toy Example 0: stack ping-pong

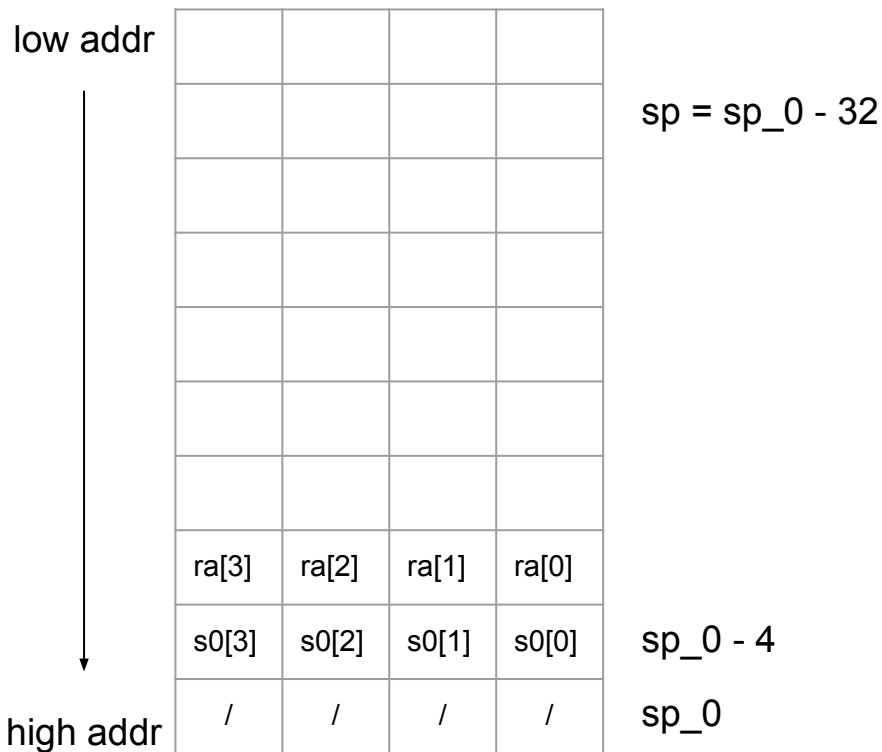


$sp = sp_0 - 12016$

- Push saved regs on the stack
- Align to 128-bit boundary, i.e. 32 B (ABI)
- Allocate stack space for both the arrays (12000 B)
- Put in a0 the pointer to the first vector:
 $a0 = sp + 12288 - 288 - 12288 + 288 = sp$
- Jump to routine
- Put in a0 the pointer to the second vector:
 $a0 = sp + 12288 - 288 - 8192 + 192 = sp - 4000$

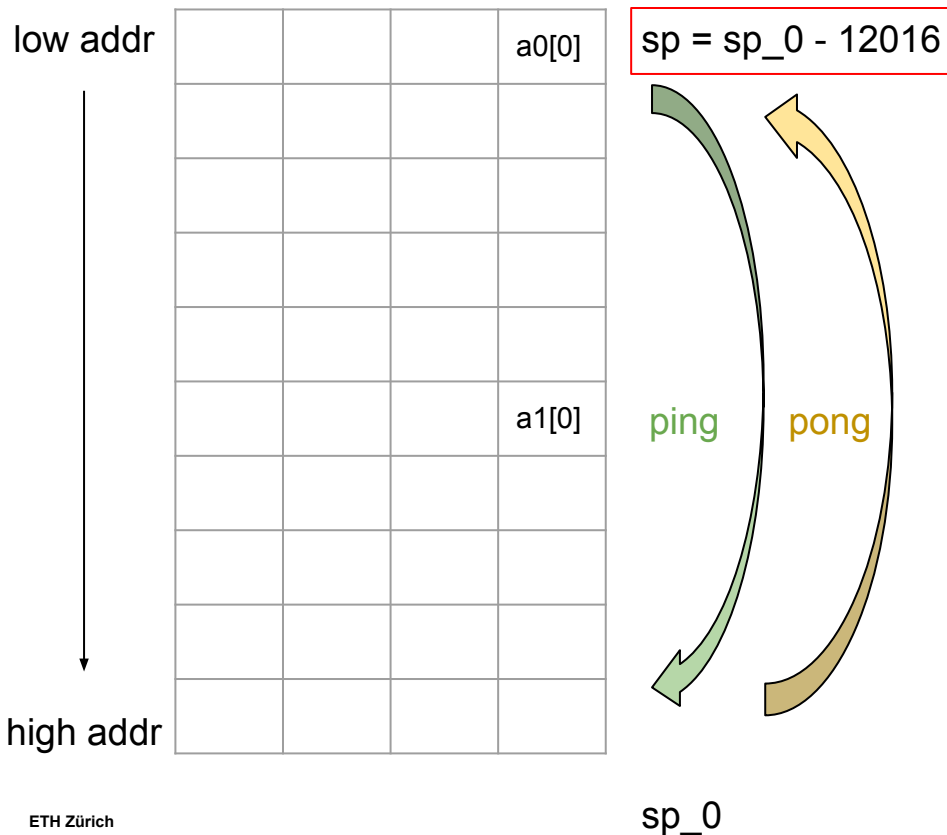
$sp - 4000$

Toy Example 0: stack ping-pong



- Push saved regs on the stack
- Align to 128-bit boundary, i.e. 32 B (ABI)
- Allocate stack space for both the arrays (12000 B)
- Put in a0 the pointer to the first vector:
 $a0 = sp + 12288 - 288 - 12288 + 288 = sp$
- Jump to routine
- Put in a0 the pointer to the second vector:
 $a0 = sp + 12288 - 288 - 8192 + 192 = sp - 4000$
- Jump to routine
- Pop variables from the stack and return

Toy Example 0: stack ping-pong



- Push saved regs on the stack
- Align to 128-bit boundary, i.e. 32 B (ABI)
- Allocate stack space for both the arrays (12000 B)
- Put in a0 the pointer to the first vector:

$$a0 = sp + 12288 - 288 - 12288 + 288 = sp$$

$$a0 = sp + (12288 - 288) - (12288 - 288) = sp$$

$$a0 = sp + (12000) - (12000) = sp$$

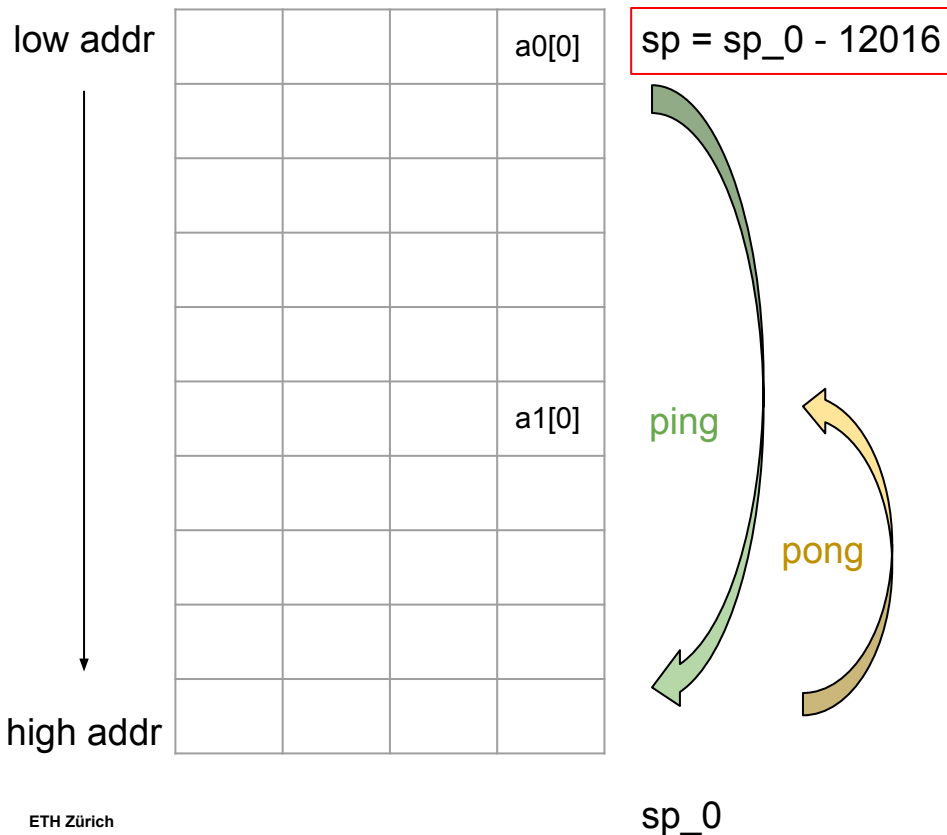
Remember that:

- int a0[1000]
- int a1[2000]



Total: $3000 * 4 \text{ B} = 12000 \text{ B}$

Toy Example 0: stack ping-pong



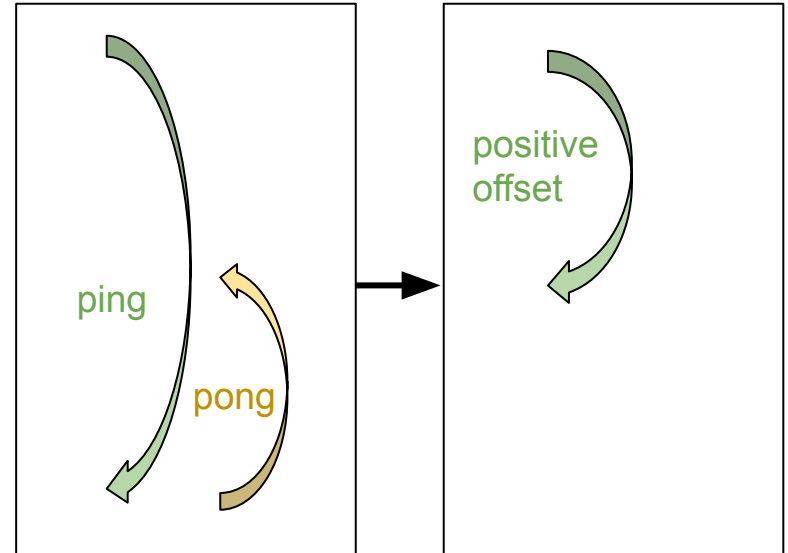
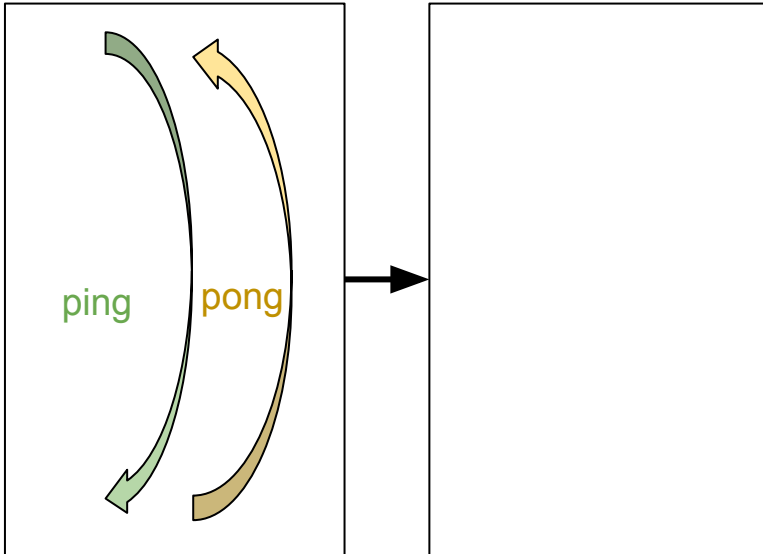
- Push saved regs on the stack
- Align to 128-bit boundary, i.e. 32 B (ABI)
- Allocate stack space for both the arrays (12000 B)
- Put in a0 the pointer to the first vector:
 $a0 = sp + 12288 - 288 - 12288 + 288 = sp$
- Jump to routine
- Put in a0 the pointer to the second vector:
 $a0 = sp + 12288 - 288 - 8192 + 192 = sp - 4000$
 $a0 = sp + (12288 - 288) - (8192 - 192) = sp - 4000$
 $a0 = sp + (12000) - (8000) = sp - 4000$

Remember that:

- `int a1[2000]` → Size: $2000 * 4B = 8000$

Toy Example 0: stack ping-pong

- The ping-pong is redundant



Toy Example 0: stack ping-pong

- The ping-pong is redundant
- When the array is large, each “ping” and “pong” value is at least 6 B
- Need to understand why this is triggered
- In the OPUS function (`celt_decode_lost`), the ping-pong seems to be repeated for each stack-related memory operation

Toy Example 1: inefficiencies + redundant epilogue

```
#include <stdio.h>

#define M 2000
#define N 1000

extern void foo(int *arr);

int main(int argc, char **argv) {
    int arr_0[argc*N]; // Loaded on the stack
    int arr_1[M]; // Loaded on the stack

    foo(arr_0);
    foo(arr_1);

    return 0;
}
```

toy_example_1.c

VLA

```
00000000 <main>:
0: 3e800793      li    a5,1000
4: 02f50533      mul   a0,a0,a5
8: 1101         addi   sp,sp,-32
a: 7379         lui    t1,0xffffe
c: ce06         sw     ra,28(sp)
e: cc22         sw     s0,24(sp)
10: 0d030313     addi   t1,t1,208 # fffffe0d <main+0xfffffe0d>
14: 1000         addi   s0,sp,32
16: 911a         add    sp,sp,t1
18: 050a         slli   a0,a0,0x2
1a: 40a10133     sub    sp,sp,a0
1e: 850a         mv     a0,sp
20: 00000097     auipc  ra,0x0
20: R_RISCV_CALL foo
20: R_RISCV_RELAX *ABS*
24: 000080e7     jalr   ra # 20 <main+0x20>
28: 7579         lui    a0,0xfffffe
2a: ff040793     addi   a5,s0,-16
2e: 0c050513     addi   a0,a0,192 # fffffe0c <main+0xfffffe0c>
32: 953e         add    a0,a0,a5
34: 00000097     auipc  ra,0x0
34: R_RISCV_CALL foo
34: R_RISCV_RELAX *ABS*
38: 000080e7     jalr   ra # 34 <main+0x34>
3c: 7379         lui    t1,0xfffffe
3e: 0b030313     addi   t1,t1,176 # fffffe0b <main+0xfffffe0b>
42: 00640133     add    sp,s0,t1
46: 6309         lui    t1,0x2
48: f3030313     addi   t1,t1,-208 # 1f30 <main+0x1f30>
4c: 911a         add    sp,sp,t1
4e: 40f2         lw     ra,28(sp)
50: 4462         lw     s0,24(sp)
52: 4501         li     a0,0
54: 6105         addi   sp,sp,32
56: 8082         ret
```

toy_example_1.s

Toy Example 1: inefficiencies

- No ping-pong
- The base address is saved inside s0 (short circuited the “ping” part)
- Other inefficiencies: address formation and stack manipulation
- At the second routine call, the a0 is formed from s0 with redundant operations
- We can derive a0 from sp instead

Toy Example 1: redundant epilogue

```
#include <stdio.h>

#define M 2000
#define N 1000

extern void foo(int *arr);

int main(int argc, char **argv) {
    int arr_0[argc*N]; // Loaded on the stack
    int arr_1[M]; // Loaded on the stack

    foo(arr_0);
    foo(arr_1);

    return 0;
}
```

toy_example_1.c

```
00000000 <main>:
0: 3e800793 ..... li a5,1000
4: 02f50533 ..... mul a0,a0,a5
8: 1101 ..... addi sp,sp,-32
a: 7379 ..... lui t1,0xffffe
c: ce06 ..... sw ra,28(sp)
e: cc22 ..... sw s0,24(sp)
10: 0d030313 ..... addi t1,t1,208 # fffffe0d <main+0xfffffe0d>
14: 1000 ..... addi s0,sp,32
16: 911a ..... add sp,sp,t1
18: 050a ..... slli a0,a0,0x2
1a: 40a10133 ..... sub sp,sp,a0
1e: 850a ..... mv a0,sp
20: 00000097 ..... auipc ra,0x0
20: R_RISCV_CALL foo
20: R_RISCV_RELAX *ABS*
24: 000080e7 ..... jalr ra,#20 <main+0x20>
28: 7579 ..... lui a0,0xffffe
2a: ff040793 ..... addi a5,s0,-16
2e: 0c050513 ..... addi a0,a0,192 # fffffe0c <main+0xfffffe0c>
32: 953e ..... add a0,a0,a5
34: 00000097 ..... auipc ra,0x0
34: R_RISCV_CALL foo
34: R_RISCV_RELAX *ABS*
38: 000080e7 ..... jalr ra,#34 <main+0x34>
3c: 7379 ..... lui t1,0xffffe
3e: 0b030313 ..... addi t1,t1,176 # fffffe0b <main+0xfffffe0b>
42: 00640133 ..... add sp,s0,t1
46: 6309 ..... lui t1,0x2
48: f3030313 ..... addi t1,t1,-208 # 1f30 <main+0x1f30>
4c: 911a ..... add sp,sp,t1
4e: 40f2 ..... lw ra,28(sp)
50: 4462 ..... lw s0,24(sp)
52: 4501 ..... li a0,0
54: 6105 ..... addi sp,sp,32
56: 8082 ..... ret
```

toy_example_1.s

Toy Example 1: redundant epilogue

7379	lui	t1,0xffffe
0b030313	addi	t1,t1,176
00640133	add	sp,s0,t1
6309	lui	t1,0x2
f3030313	addi	t1,t1,-208
911a	add	sp,sp,t1

18 B

xxxxxxxxx	addi	sp,s0,-32
-----------	------	-----------

4 B