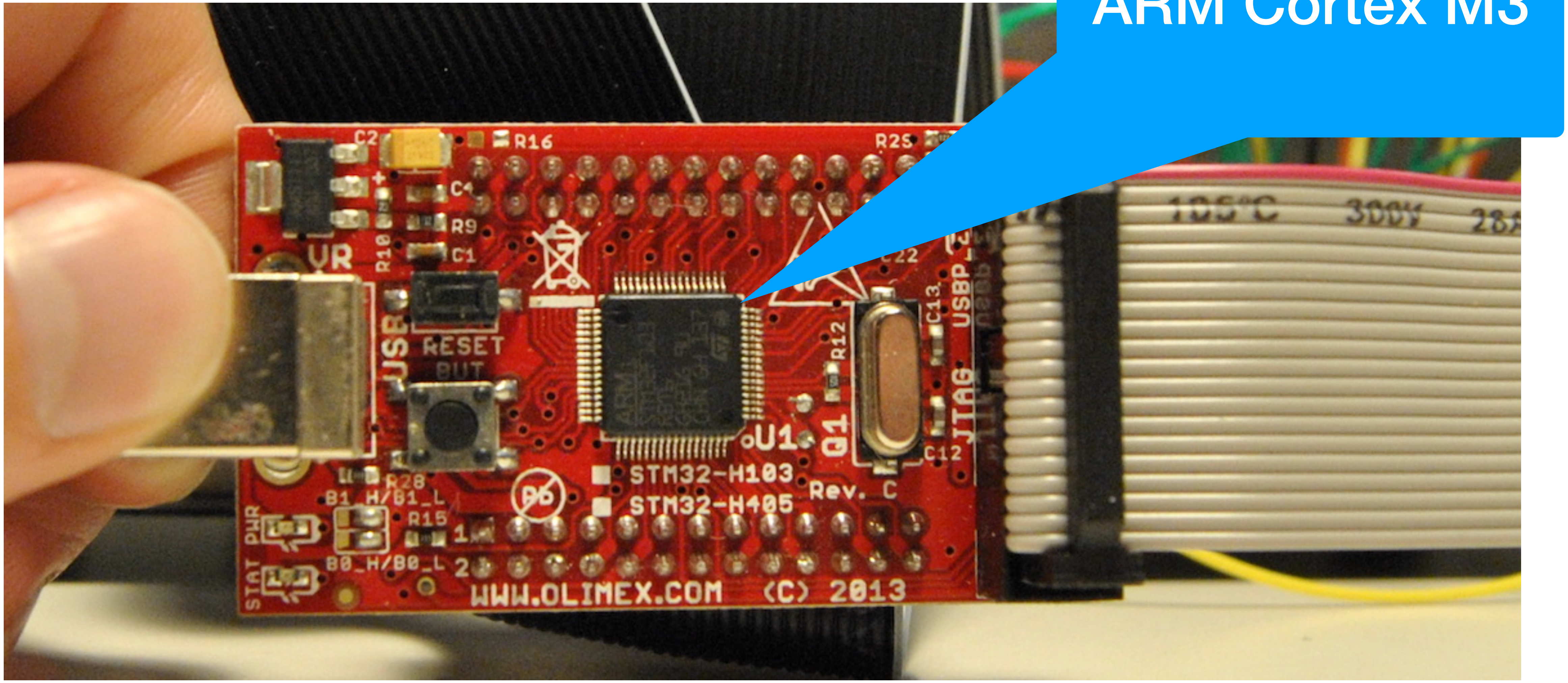
Running C programs bare metal on ARM using the GNU toolchain

foss-gbg 2018-09-26

Jacob Mossberg
https://www.jacobmossberg.se

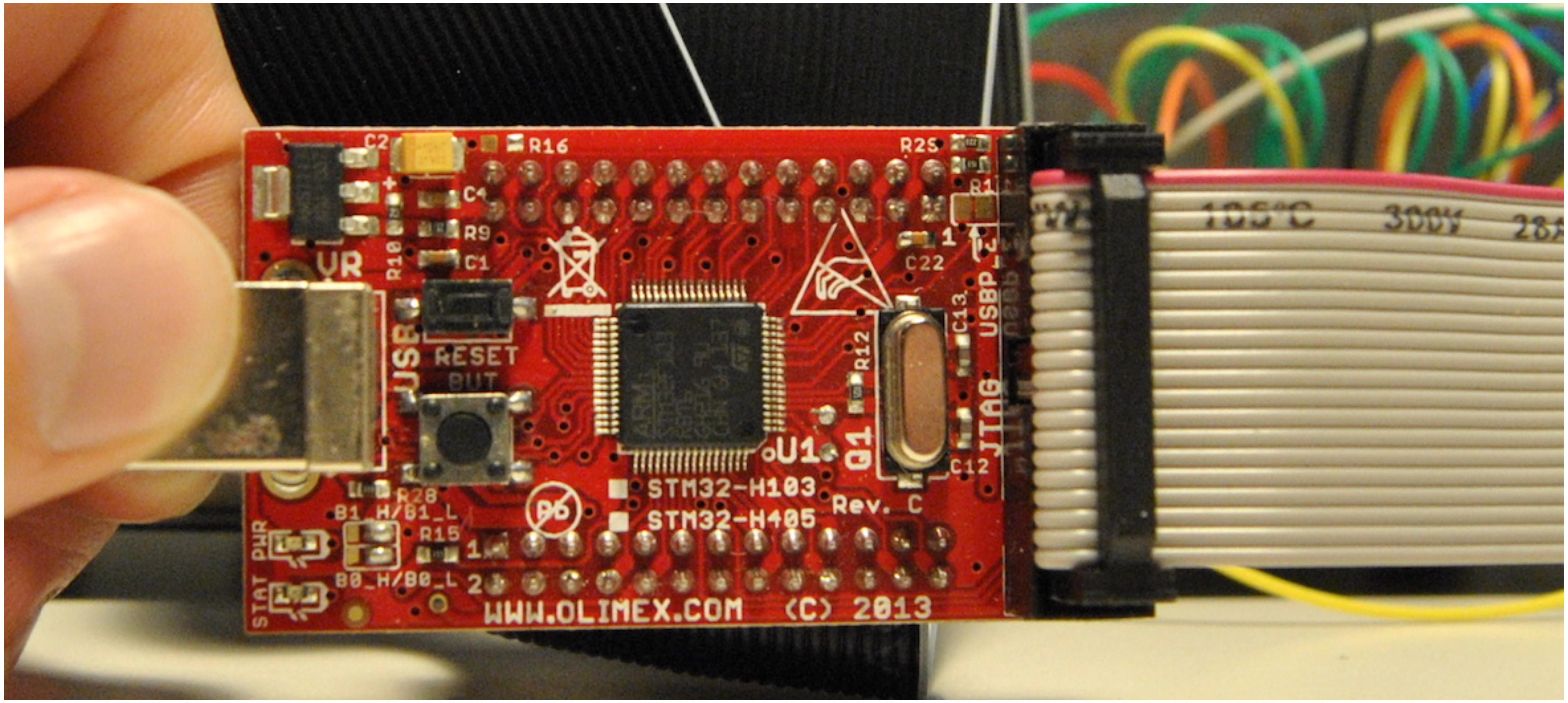
```
static const int a = 7;
static int b = 8;
static int sum;

void main()
{
  sum = a + b;
}
```



ARM Cortex M3

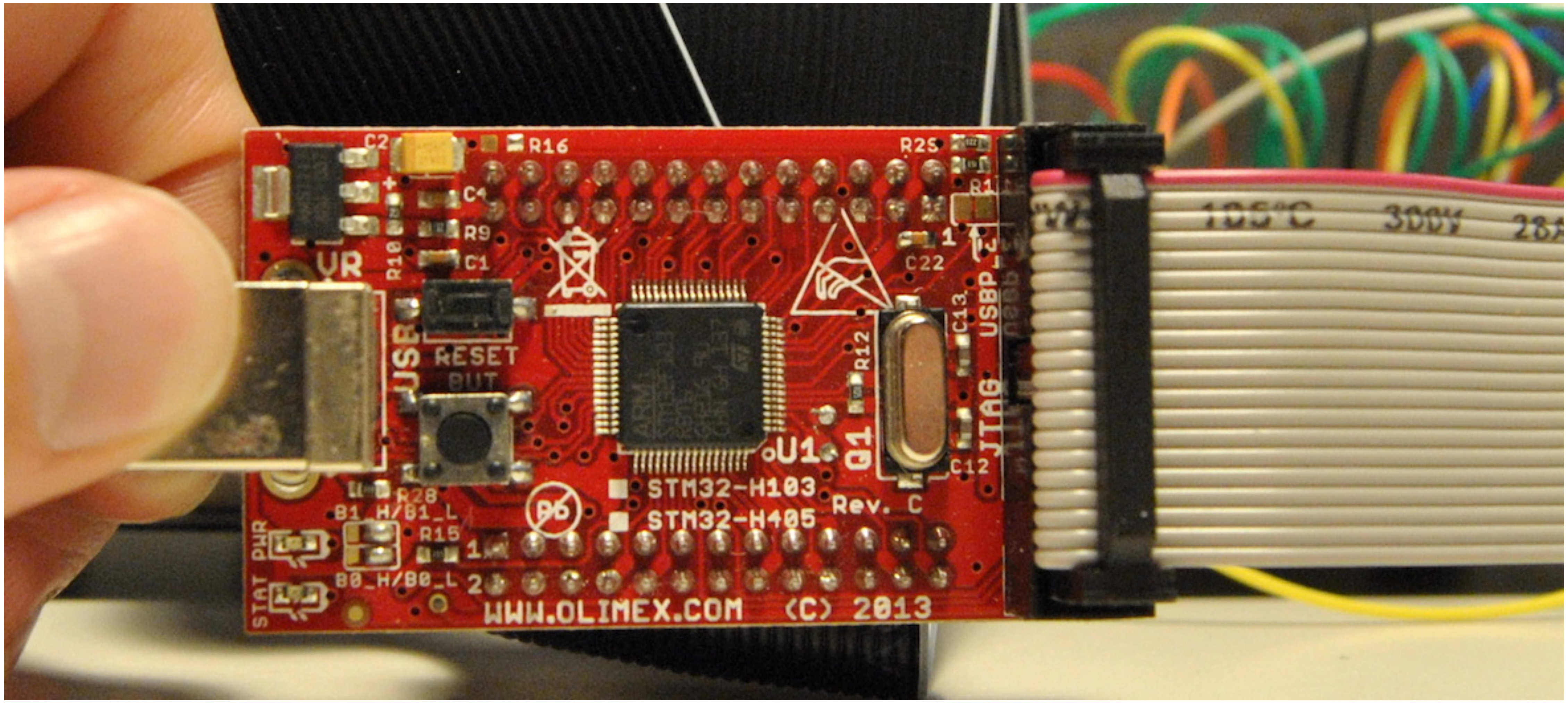
C prerequisites

```
mov r2, #3
mov r3, #4
add r4, r2, r3
```



r2 = 3
r3 = 4
r4 = r2 + r3 = 7

```
mov r2, #3
mov r3, #4
add r4, r2, r3
```

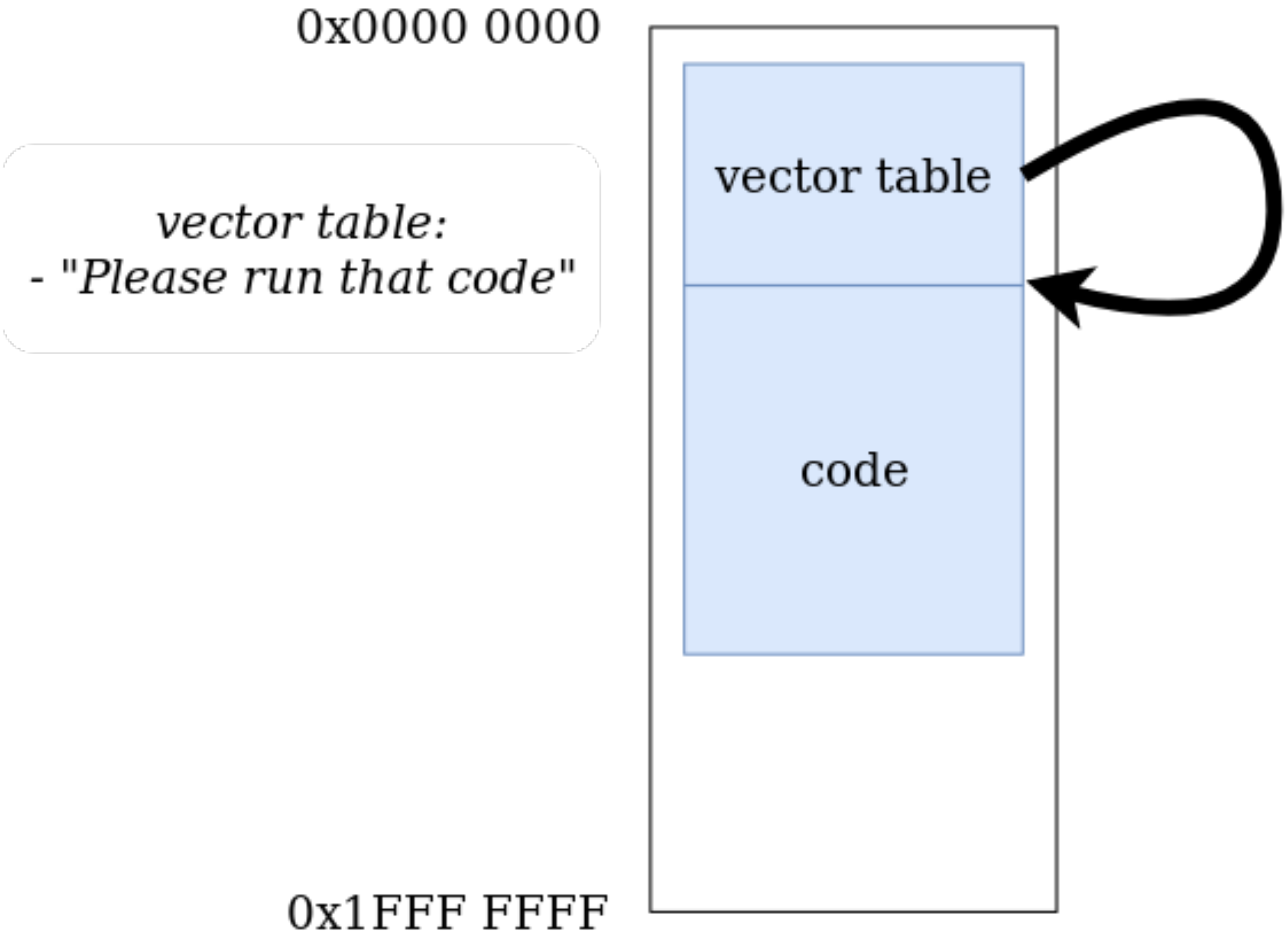


Assembler prerequisites

What happens at power on?

A reset exception happens

Flash memory



vector table:
- *"Please run that code"*

Vector table

Address	Description
0x0000 0000	Initial Stack Pointer (SP) value
0x0000 0004	Reset exception
...	Other exceptions...

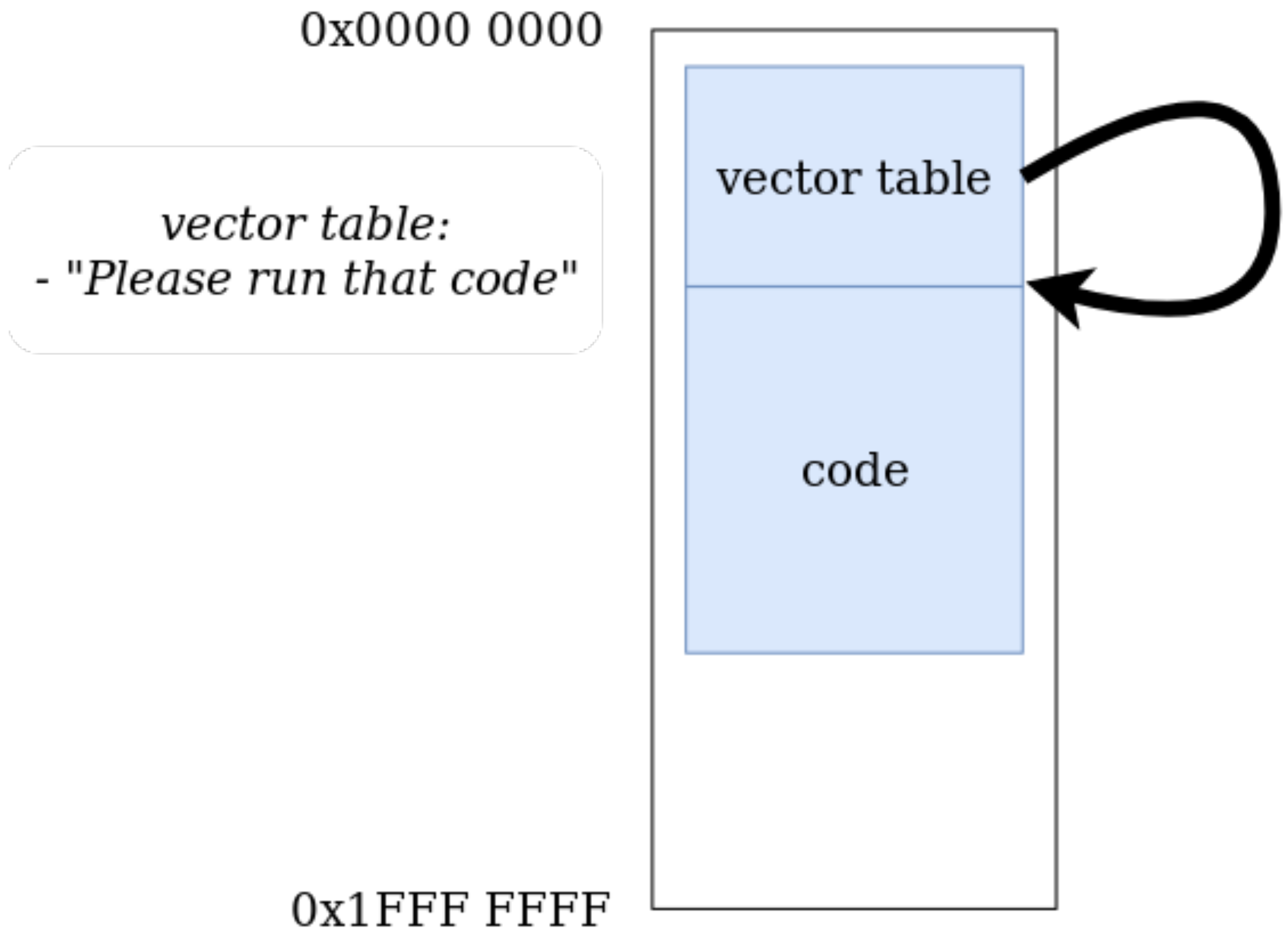
Vector table

```
.section        vectors
.word           0
.word           _start + 1
```

_start

```
   .text
_start:
    mov r2, #3
    mov r3, #4
    add r4, r2, r3

stop:
    b stop
```

Flash memory



vector table:
- "Please run that code"

```
.section     vectors
.word        0
.word        _start + 1
```

```
   .text
_start:
    mov r2, #3
    mov r3, #4
    add r4, r2, r3

stop:
    b stop
```

Assembler prerequisites

- ✓ A. Vector table with start address for reset exception handler
- ✓ B. Vector table don't need stack pointer initialization
- 🛑 C. Vector table at address 0x0
- 🛑 D. `.text` section after the vector table in flash

Assembler prerequisites

- ✓ A. Vector table with start address for reset exception handler
- ✓ B. Vector table don't need stack pointer initialization
- 🛑 C. Vector table at address 0x0
- 🛑 D. `.text` section after the vector table in flash

Linker

Linker script

```
SECTIONS
{
  .  = 0x0;
  .text :
  {
    *(vectors)
    *(.text)
  }
}
```

Assembler prerequisites

✓ A. Vector table with start address for reset exception handler

✓ B. Vector table don't need stack pointer initialization

✓ C. Vector table at address 0x0

✓ D. `.text` section after the vector table in flash

Compile

Use the thumb instruction set

```
-mcpu=cortex-m3 -mthumb: cpu type
-o <file>: output file
```

```
$ arm-none-eabi-as -mcpu=cortex-m3 \
                   -mthumb \
                   -o add.o add.s
```

The GNU Assembler (gas)

Link

```
-Tstm32.ld: use linker script stm32.ld
-o <file>: output file
```

```
$ arm-none-eabi-ld -Tstm32.ld \
                   -o add.elf \
                   add.o
```

Inspect elf file

```
$ xxd -c 4 add.elf | head -n4
00000000: 7f45 4c46  .ELF
00000004: 0101 0100  ....
00000008: 0000 0000  ....
0000000c: 0000 0000  ....
```

Inspect elf file

```
$ xxd -c 4 ad                          4
0000
00                         0   ....
000          0000 0000   ....
0000000c: 0000 0000   ....
```

Will not work....

Convert to binary

GNU Binary Utilities documentation:

"*When* ***`objcopy`*** *generates a raw binary file, it will essentially produce a memory dump of the contents of the input object file.*

All symbols and relocation information will be ***discarded****. The memory dump will start at the load address of the lowest section copied into the output file.*"

Convert to binary

```
$ arm-none-eabi-objcopy -O binary \
                        add.elf \
                        add.bin
```

Inspect bin file

```
$ xxd -c 4 add.bin | head -n4
00000000: 0000 0000  ....
00000004: 0900 0000  ....
00000008: 0322 0423  .".#
0000000c: d418 fee7  ....
```

Hex	Instruction
0x0322	MOVS R2, #3
0x0423	MOVS R3, #4
0xD418	ADDS R4, R2, R3
0xFEE7	B #0

Inspect bin file

```
$ xxd -c 4
```

Look at section A7.7.75 in ARMv7-M Architecture Reference Manual

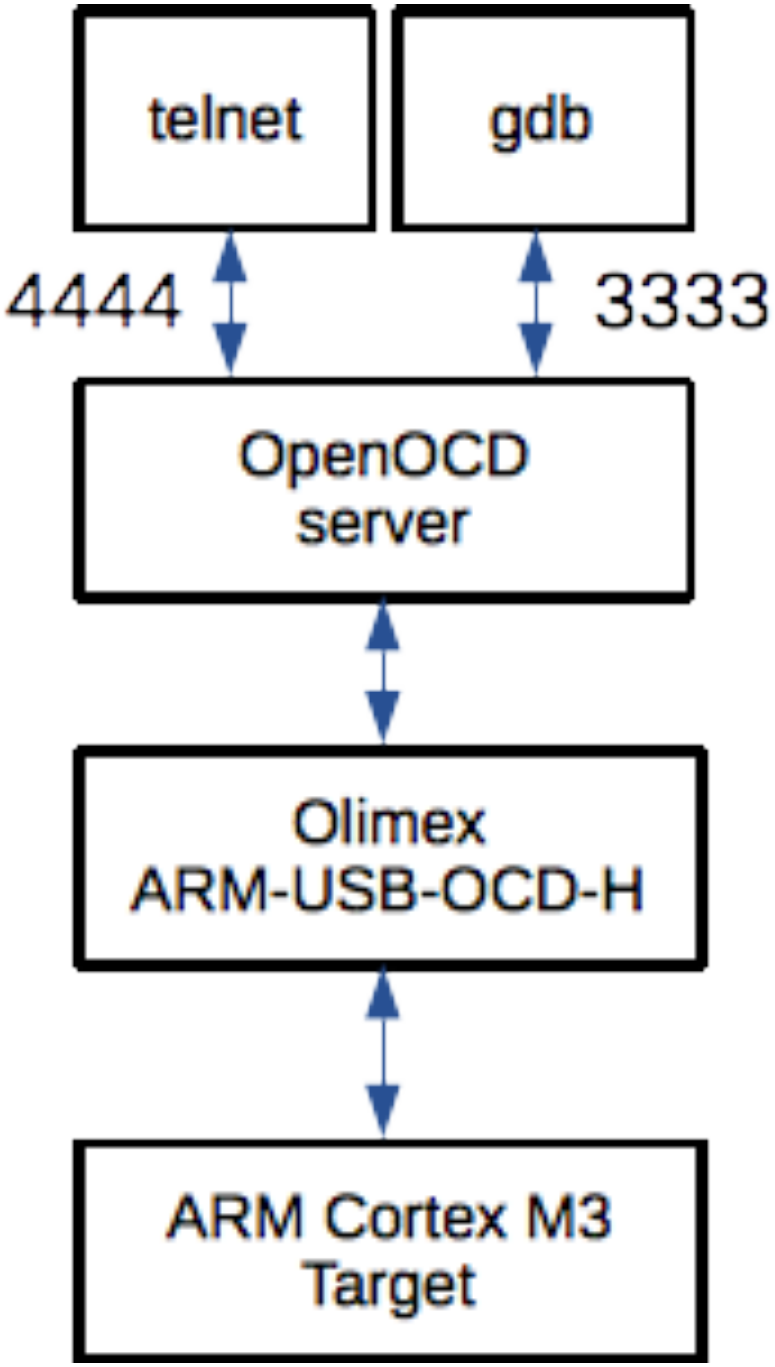
or

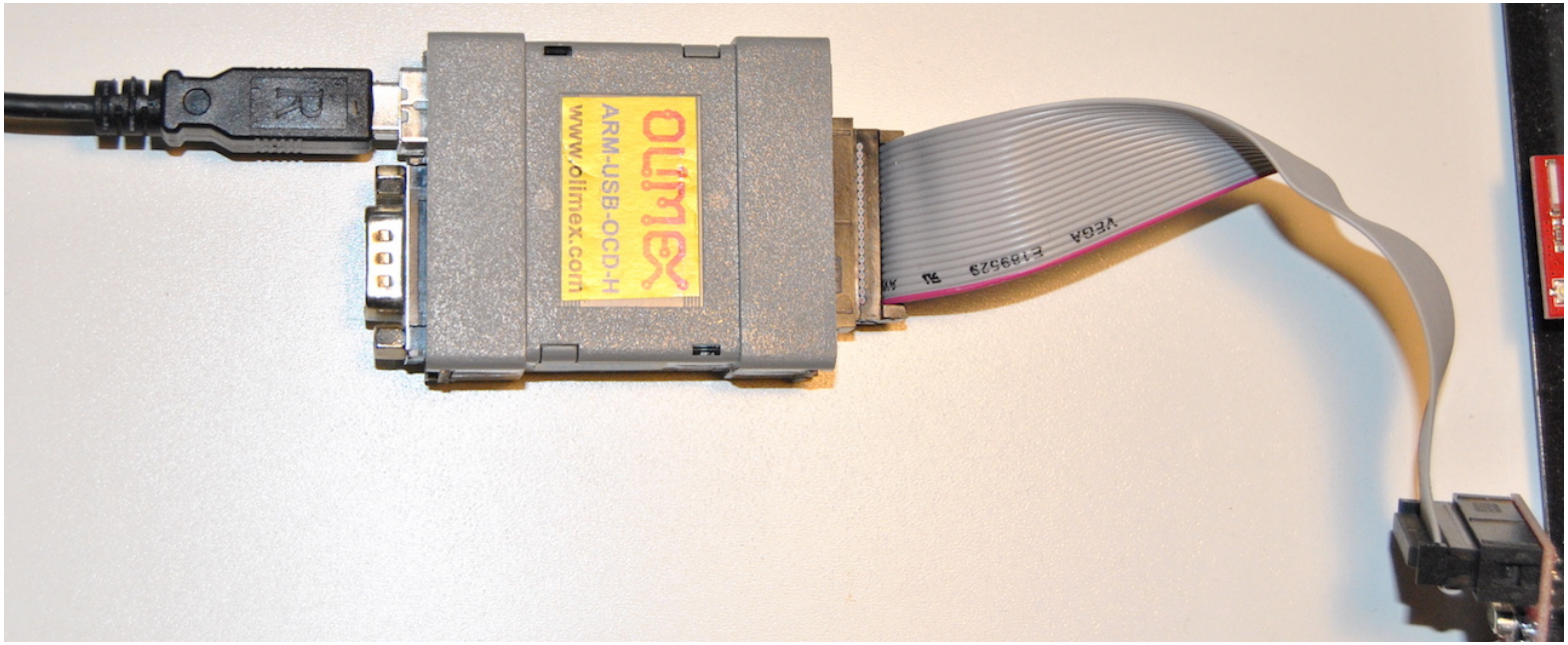
http://armconverter.com/hextoarm/

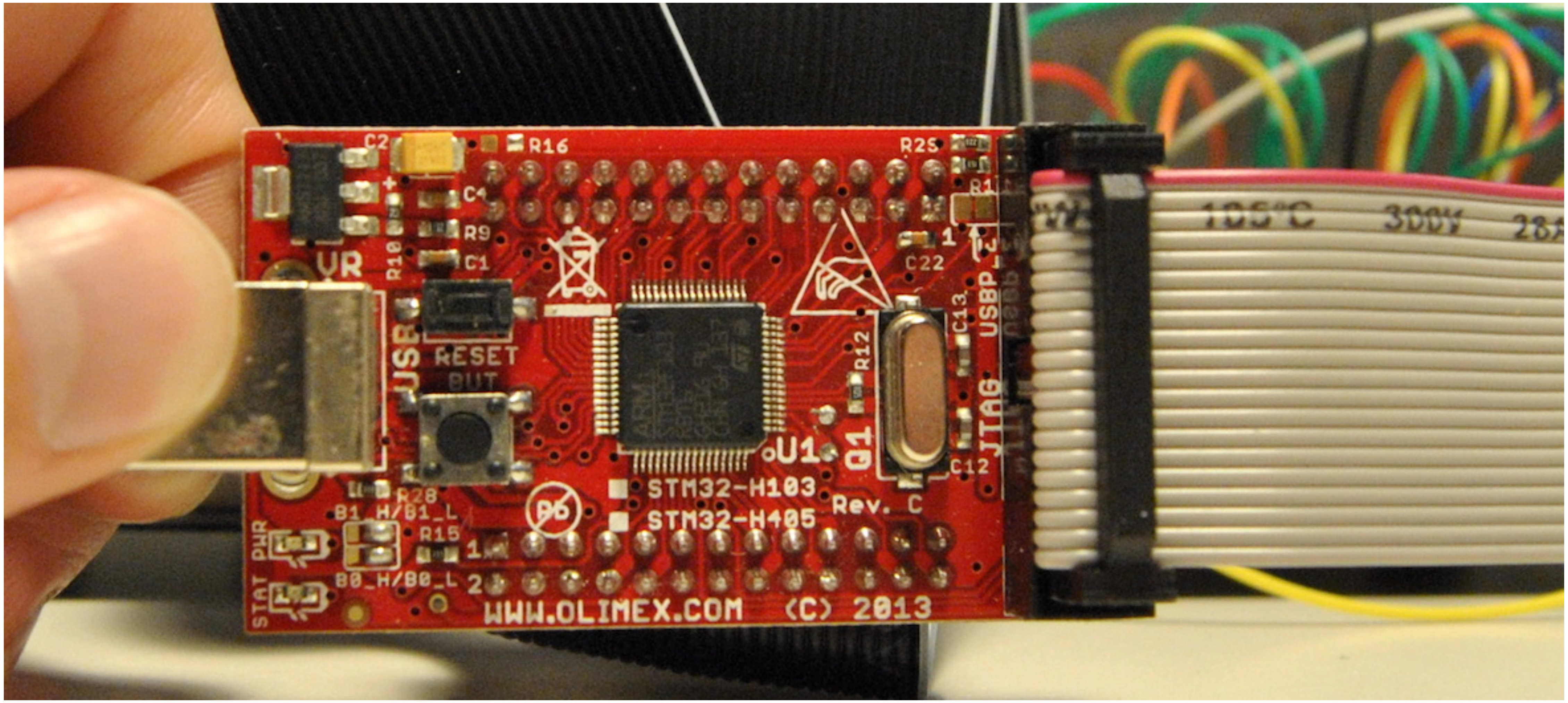
	MOVS R3, #4
0xD418	ADDS R4, R2, R3
0xFEE7	B #0

Thanks to http://armconverter.com/hextoarm/

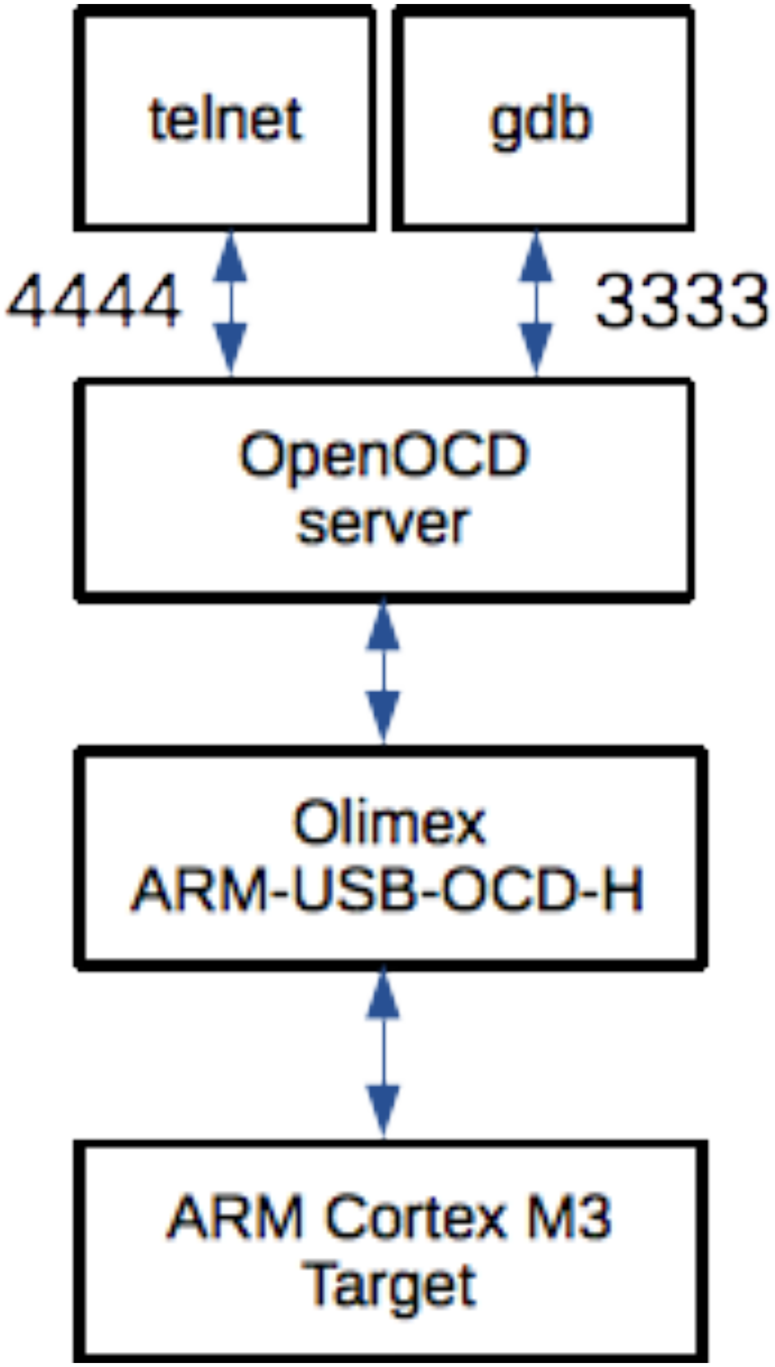
OpenOCD


telnet
gdb
4444
3333
OpenOCD server
Olimex ARM-USB-OCD-H
ARM Cortex M3 Target


OLIMEX
ARM-USB-OCD-H
www.olimex.com


STM32-H103
STM32-H405
Rev. C
JTAG
RESET BUT
USB
WWW.OLIMEX.COM (C) 2013

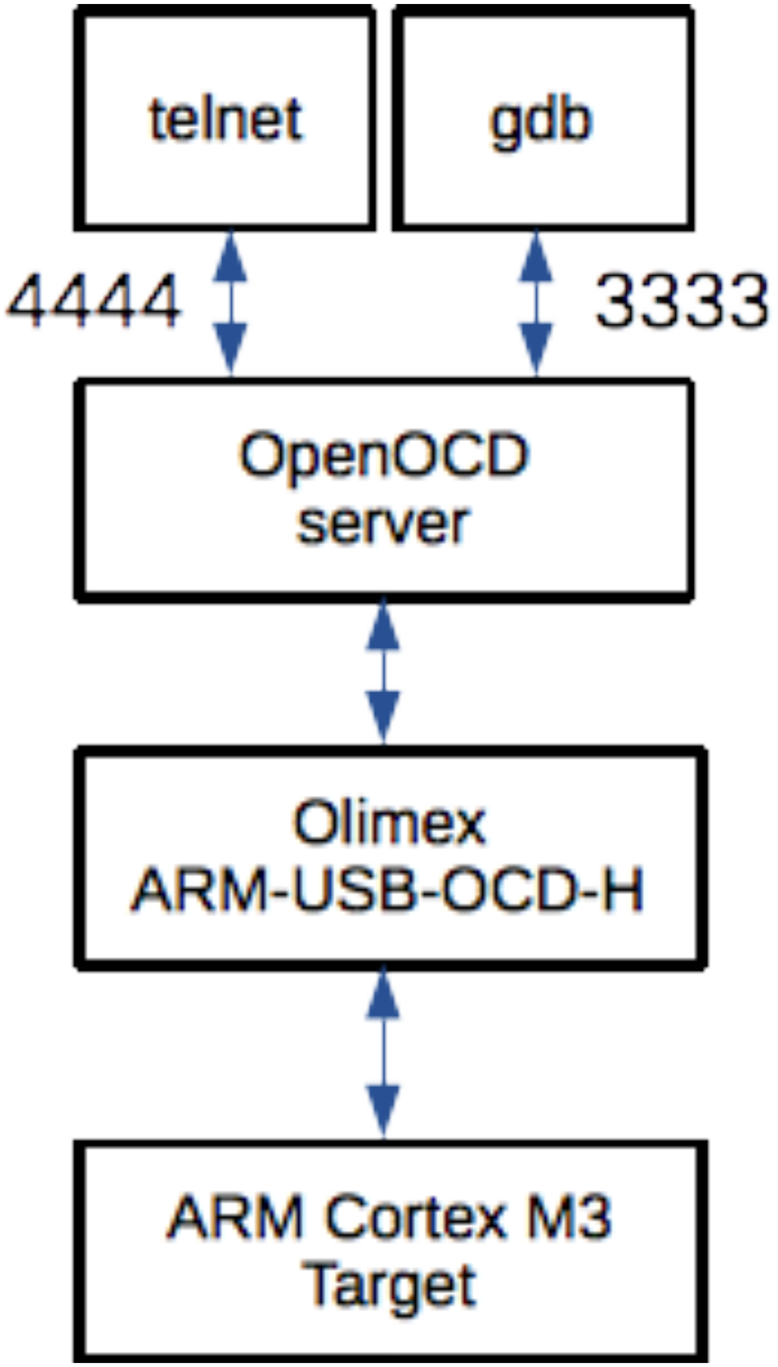
OpenOCD


telnet
gdb
4444
3333
OpenOCD server
Olimex ARM-USB-OCD-H
ARM Cortex M3 Target

Flash

```
$ openocd -f openocd.cfg
```

```
$ telnet localhost 4444
halt
stm32f1x mass_erase 0
flash write_bank 0 add.bin 0
reset run
```



Verify

```
halt
reg

==== arm v7m registers
(0) r0 (/32): 0x00000020
(1) r1 (/32): 0x00000000
(2) r2 (/32): 0x00000003
(3) r3 (/32): 0x00000004
(4) r4 (/32): 0x00000007
...
```

Assembler prerequisites

✓ A. Vector table with start address for reset exception handler

✓ B. Vector table don't need stack pointer initialization

✓ C. Vector table at address 0x0

✓ D. `.text` section after the vector table in flash

C program

```
static const int a = 7;
static int b = 8;
static int sum;

void main()
{
  sum = a + b;
}
```

Generate assembler code

```
$arm-none-eabi-gcc -S \
                   -mcpu=cortex-m3 \
                   -mthumb \
                   test_program.c
```

```
-S
    Stop after the stage of compilation
    proper; do not assemble. The output
    is in the form of an assembler
    code file for each non-assembler input file
    specified.
    .cpu cortex-m3
    .eabi_attribute 20, 1
    .eabi_attribute 21, 1
    .eabi_attribute 23, 3
    .eabi_attribute 24, 1
    .eabi_attribute 25, 1
    .eabi_attribute 26, 1
    .eabi_attribute 30, 6
    .eabi_attribute 34, 1
    .eabi_attribute 18, 4
    .file "test_program.c"
    .section  .rodata
    .align  2
    .type a, %object
    .size a, 4
a:
    .word 7
    .data
    .align  2
    .type b, %object
    .size b, 4
b:
    .word 8
    .bss
    .align  2
sum:
    .space  4
    .size sum, 4
    .text
    .align  1
    .global main
    .syntax unified
    .thumb
    .thumb_func
    .fpu softvfp
    .type main, %function
main:
    @ args = 0, pretend = 0, frame = 0
    @ frame_needed = 1, uses_anonymous_args = 0
    @ link register save eliminated.
    push  {r7}
    add r7, sp, #0
    movs  r2, #7
    ldr r3, .L2
    ldr r3, [r3]
    add r3, r3, r2
    ldr r2, .L2+4
    str r3, [r2]
    nop
    mov sp, r7
    @ sp needed
    pop {r7}
    bx  lr
.L3:
    .align  2
.L2:
    .word b
    .word sum
    .size main, .-main
    .ident  "GCC: (15:6.3.1+svn253039-1build1) 6.3.1 20170620"
    @ args = 0, pretend = 0, frame = 0
    @ frame_needed = 1, uses_anonymous_args = 0
    @ link register save eliminated.
    push  {r7}
    add r7, sp, #0
    movs  r2, #7
    ldr r3, .L2
    ldr r3, [r3]
    add r3, r3, r2
    ldr r2, .L2+4
    str r3, [r2]
    nop
    mov sp, r7
    @ sp needed
    pop {r7}
    bx  lr
.L3:
    .align  2
.L2:
    .word b
    .word sum
    .size main, .-main
    .ident  "GCC: (15:6.3.1+svn253039-1build1) 6.3.1 20170620"
```

C prerequisites

```
    .cpu cortex-m3
    .eabi_attribute 20, 1
    .eabi_attribute 21, 1
    .eabi_attribute 23, 3
    .eabi_attribute 24, 1
    .eabi_attribute 25, 1
    .eabi_attribute 26, 1
    .eabi_attribute 30, 6
    .eabi_attribute 34, 1
    .eabi_attribute 18, 4
    .file "test_program.c"
    .section  .rodata
    .align  2
    .type a, %object
    .size a, 4
a:
    .word 7
    .data
    .align  2
    .type b, %object
    .size b, 4
b:
    .word 8
    .bss
    .align  2
sum:
    .space  4
    .size sum, 4
    .text
    .align  1
    .global main
    .syntax unified
    .thumb
    .thumb_func
    .fpu softvfp
    .type main, %function
main:
    @ args = 0, pretend = 0, frame = 0
    @ frame_needed = 1, uses_anonymous_args = 0
    @ link register save eliminated.
    push  {r7}
    add r7, sp, #0
    movs  r2, #7
    ldr r3, .L2
    ldr r3, [r3]
    add r3, r3, r2
    ldr r2, .L2+4
    str r3, [r2]
    nop
    mov sp, r7
    @ sp needed
    pop {r7}
    bx  lr
.L3:
    .align  2
.L2:
    .word b
    .word sum
    .size main, .-main
    .ident  "GCC: (15:6.3.1+svn253039-1build1) 6.3.1
20170620"
```

```
static const int a = 7;
static int b = 8;
static int sum;
```

```
void main()
{
  sum = a + b;
}
```

```
    .section  .rodata
    .align  2
    .type a, %object
    .size a, 4
a:
    .word 7
    .data
    .align  2
    .type b, %object
    .size b, 4
b:
    .word 8
    .bss
    .align  2
sum:
    .space  4
    .size sum, 4
```

```
static const int a = 7;
static int b = 8;
static int sum;
```

```
    .section  .rodata
    .align  2
    .type a, %object
    .size a, 4
a:
    .word 7
```

```
static const int a = 7;
```

```
    .data
    .align  2
    .type b, %object
    .size b, 4
b:
    .word 8
```

```
static int b = 8;
```

```
    .bss
    .align  2
sum:
    .space  4
    .size sum, 4
```

```
static int sum;
```

```
        .section  .rodata
        .align  2
        .type a, %object
        .size a, 4
a:
        .word 7
```

```
static const int a = 7;
```

section .rodata

C prerequisites

A. Make the immutable data in the .rodata section available in the read only memory

```
        .section  .rodata
        .align  2
        .type a, %object
        .size a, 4
a:
        .word 7
```

```
static const int a = 7;
```

```
        .data
        .align  2
        .type b, %object
        .size b, 4
b:
        .word 8
```

```
static int b = 8;
```

```
        .bss
        .align  2
sum:
        .space  4
        .size sum, 4
```

```
static int sum;
```

```
    .data
    .align  2
    .type b, %object
    .size b, 4
b:
    .word 8
```

```
static int b = 8;
```

.data

C prerequisites

A. Make the immutable data in the **.rodata** section available in the read only memory

B. Make the mutable data in the **.data** section available in the read/write memory

```
    .section  .rodata
    .align  2
    .type a, %object
    .size a, 4
a:
    .word 7
```

```c
static const int a = 7;
```

```
    .data
    .align  2
    .type b, %object
    .size b, 4
b:
    .word 8
```

```c
static int b = 8;
```

```
    .bss
    .align  2
sum:
    .space  4
    .size sum, 4
```

```c
static int sum;
```

```
     .bss
     .align  2
sum:
     .space  4
     .size sum, 4
```

```
static int sum;
```

.bss

C prerequisites

A. Make the immutable data in the **.rodata** section available in the read only memory

B. Make the mutable data in the **.data** section available in the read/write memory

C. Make the **.bss** section available in the read/write memory too. Also make sure all memory in the .bss section is initialized to zero.

http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf (page 138), i.e the C99 ISO C standard:

"10
If an object that has automatic storage duration is not initialized explicitly, its value is indeterminate. If an object that has static storage duration is not initialized explicitly, then:
—if it has pointer type, it is initialized to a null pointer;
—if it has arithmetic type, it is initialized to (positive or unsigned) zero;"

le

C. Make the **.bss** ction available in the read/write memory too. **Also make sure all memory in the .bss section is initialized to zero.**

```
    .cpu cortex-m3
    .eabi_attribute 20, 1
    .eabi_attribute 21, 1
    .eabi_attribute 23, 3
    .eabi_attribute 24, 1
    .eabi_attribute 25, 1
    .eabi_attribute 26, 1
    .eabi_attribute 30, 6
    .eabi_attribute 34, 1
    .eabi_attribute 18, 4
    .file "test_program.c"
    .section  .rodata
    .align  2
    .type a, %object
    .size a, 4
a:
    .word 7
    .data
    .align  2
    .type b, %object
    .size b, 4
b:
    .word 8
    .bss
    .align  2
sum:
    .space  4
    .size sum, 4
    .text
    .align  1
    .global main
    .syntax unified
    .thumb
    .thumb_func
    .fpu softvfp
    .type main, %function
main:
    @ args = 0, pretend = 0, frame = 0
    @ frame_needed = 1, uses_anonymous_args = 0
    @ link register save eliminated.
    push  {r7}
    add r7, sp, #0
    movs  r2, #7
    ldr r3, .L2
    ldr r3, [r3]
    add r3, r3, r2
    ldr r2, .L2+4
    str r3, [r2]
    nop
    mov sp, r7
    @ sp needed
    pop {r7}
    bx  lr
.L3:
    .align  2
.L2:
    .word b
    .word sum
    .size main, .-main
    .ident  "GCC: (15:6.3.1+svn253039-1build1) 6.3.1
20170620"
```

```
static const int a = 7;
static int b = 8;
static int sum;
```

```
void main()
{
  sum = a + b;
}
```

```
    .text
    .align  1
    .global main
    .syntax unified
    .thumb
    .thumb_func
    .fpu softvfp
    .type main, %function
main:
    push  {r7}
    add r7, sp, #0
    movs  r2, #7
    ldr r3, .L2
    ldr r3, [r3]
    add r3, r3, r2
    ldr r2, .L2+4
    str r3, [r2]
    nop
    mov sp, r7
    @ sp needed
    pop {r7}
    bx  lr
```

```
void main()
{
  sum = a + b;
}
```

```
push  {r7}
add r7, sp, #0
```

```
{
```

```
movs  r2, #7
ldr r3, .L2
ldr r3, [r3]
add r3, r3, r2
ldr r2, .L2+4
str r3, [r2]
nop
```

```
sum = a + b;
```

```
mov sp, r7
@ sp needed
pop {r7}
bx  lr
```

```
}
```

```
push  {r7}                   {
add r7, sp, #0
```

```
movs  r2, #7
ldr r3, .L2
ldr r3, [r3]
add r3, r3, r2               sum = a + b;
ldr r2, .L2+4
str r3, [r2]
nop
```

```
mov sp, r7
@ sp needed
pop {r7}                     }
bx  lr
```

```
push  {r7}
add r7, sp, #0
```

```
{
```

```
movs  r2, #7
ldr r3, .L2
ldr r3, [r3]
add r3, r3, r2
ldr r2, .L2+4
str r3, [r2]
nop
```

```
sum = a + b;
```

```
mov sp, r7
@ sp needed
pop {r7}
bx  lr
```

```
}
```

3.4.7 PUSH and POP

Push registers onto, and pop registers off a full-descending **stack**

C prerequisites

A. Make the immutable data in the **.rodata** section available in the read only memory

B. Make the mutable data in the **.data** section available in the read/write memory

C. Make the **.bss** section available in the read/write memory too. Also make sure all memory in the .bss section is initialized to zero.

D. Initialize stack pointer

Assembler prerequisites

A. Vector table with start address for reset exception handler

B. Vector table don't need stack pointer initialization

C. Vector table at address 0x0

D. `.text` section after the vector table in flash

Assembler and C prerequisites

A. Vector table with start address for reset exception handler

B. Vector table at address 0x0

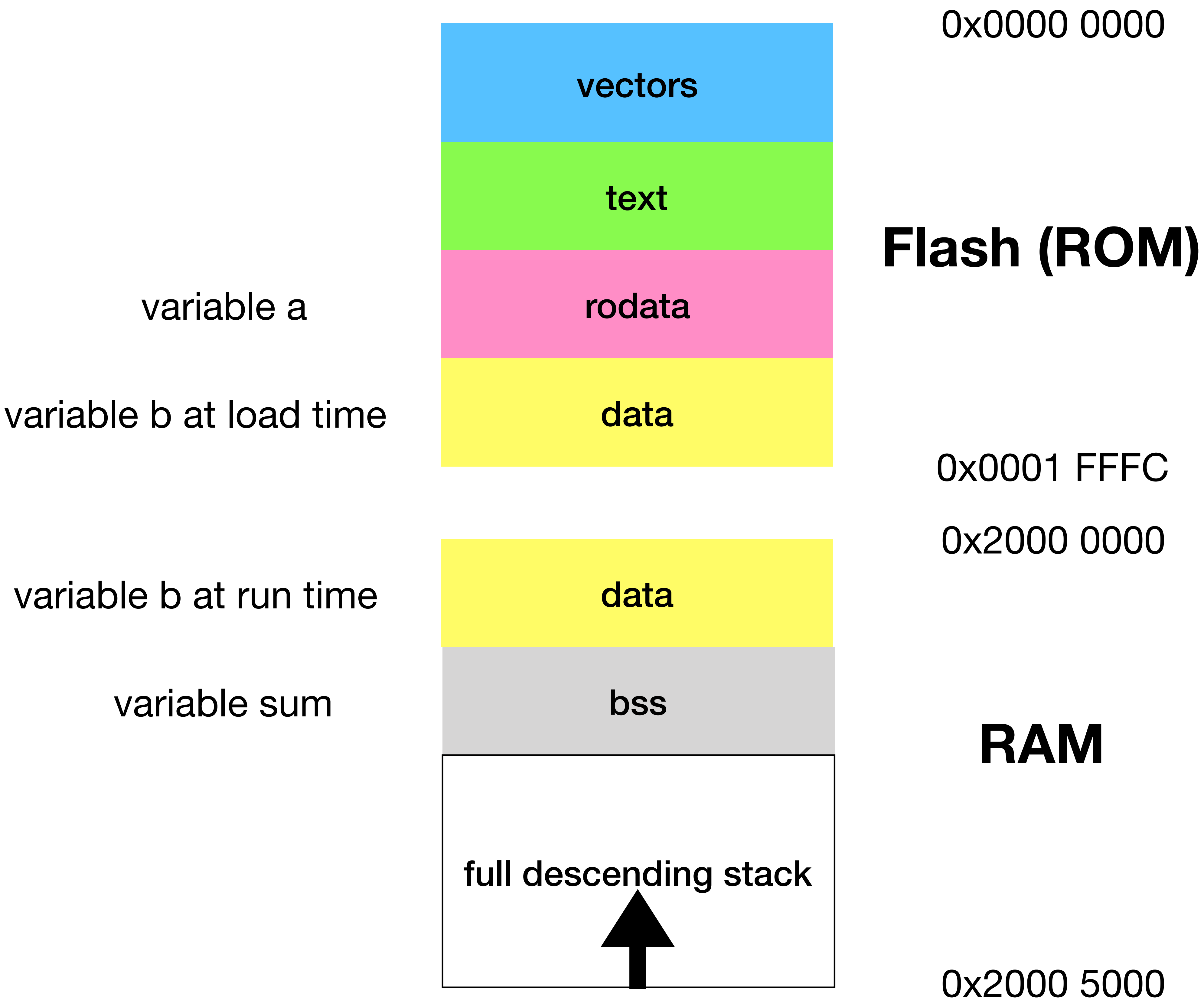
C. `.text` section after the vector table in flash

D. Make the immutable data in the **.rodata** section available in the read only memory

E. Make the mutable data in the **.data** section available in the read/write memory

F. Make the **.bss** section available in the read/write memory too. Also make sure all memory in the .bss section is initialized to zero.

G. Initialize stack pointer



0x0000 0000

vectors

text

Flash (ROM)

variable a

rodata

variable b at load time

data

0x0001 FFFC

0x2000 0000

variable b at run time

data

variable sum

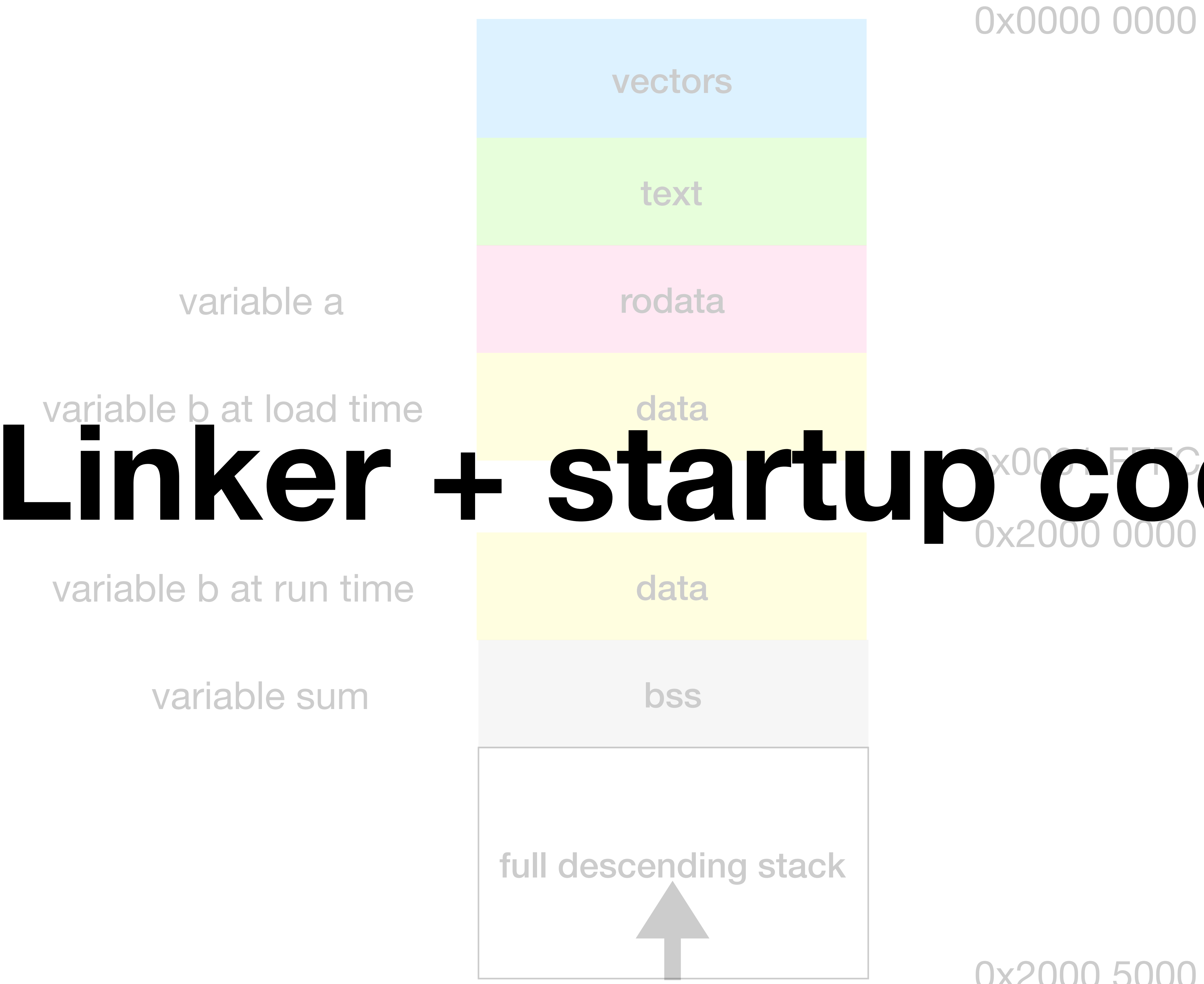
bss

RAM

full descending stack

0x2000 5000

Linker + startup code



0x0000 0000

vectors

text

variable a

rodata

variable b at load time

data

0x0004 FFFC

0x2000 0000

variable b at run time

data

variable sum

bss

full descending stack

0x2000 5000

Vector table

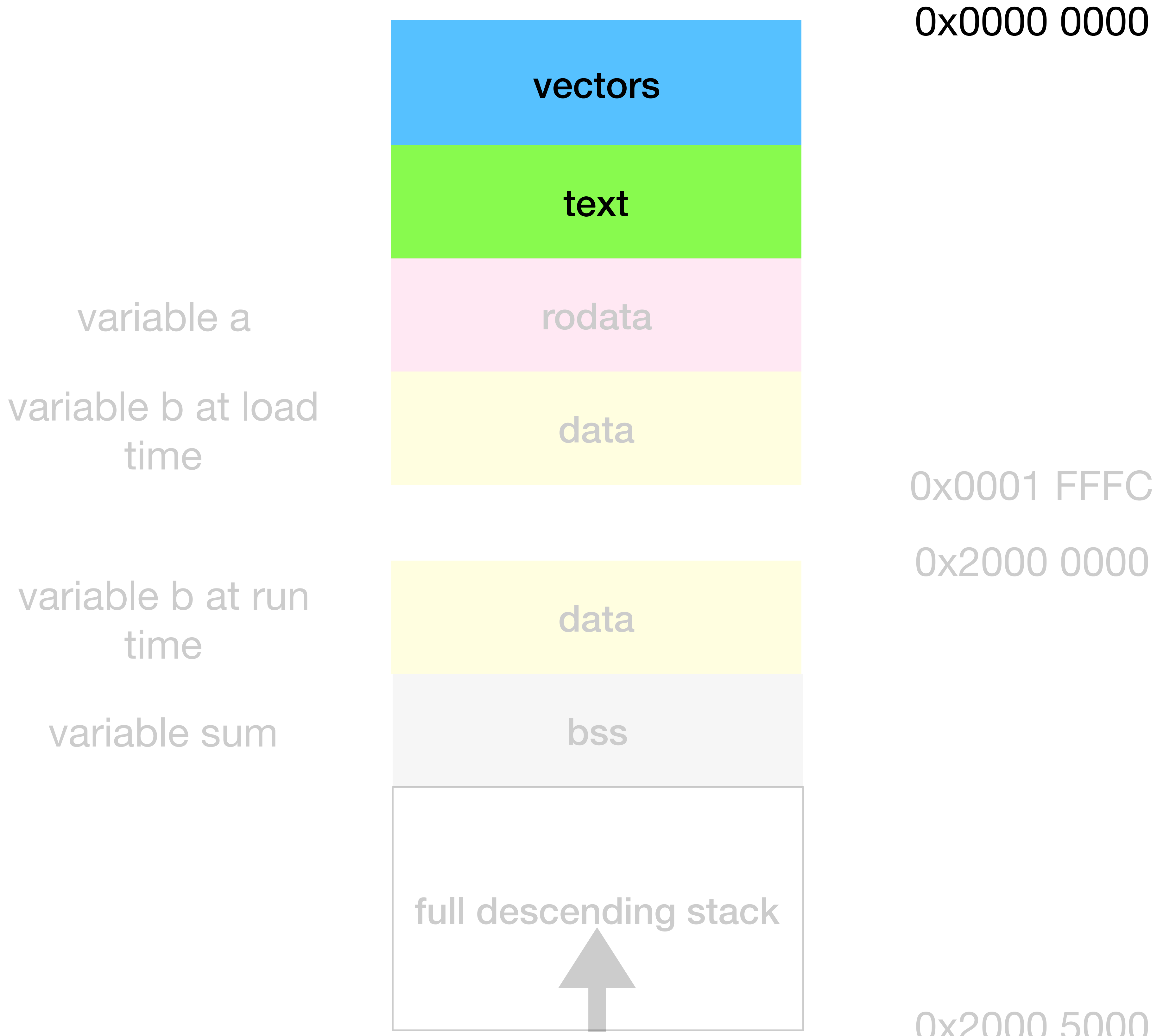
```
.section    vectors
.word       0
.word       _start + 1
```

Vector table

```
#define STACK_TOP 0x20005000
void startup();

unsigned int * myvectors[2]
__attribute__ ((section("vectors")))= {
    (unsigned int *)    STACK_TOP,
    (unsigned int *)    startup
};
```

Linker script

```
SECTIONS
{
  .  = 0x0;
  .text :
  {
    *(vectors)
    *(.text)
  }
}
```



0x0000 0000

vectors

text

variable a

rodata

variable b at load time

data

0x0001 FFFC

0x2000 0000

variable b at run time

data

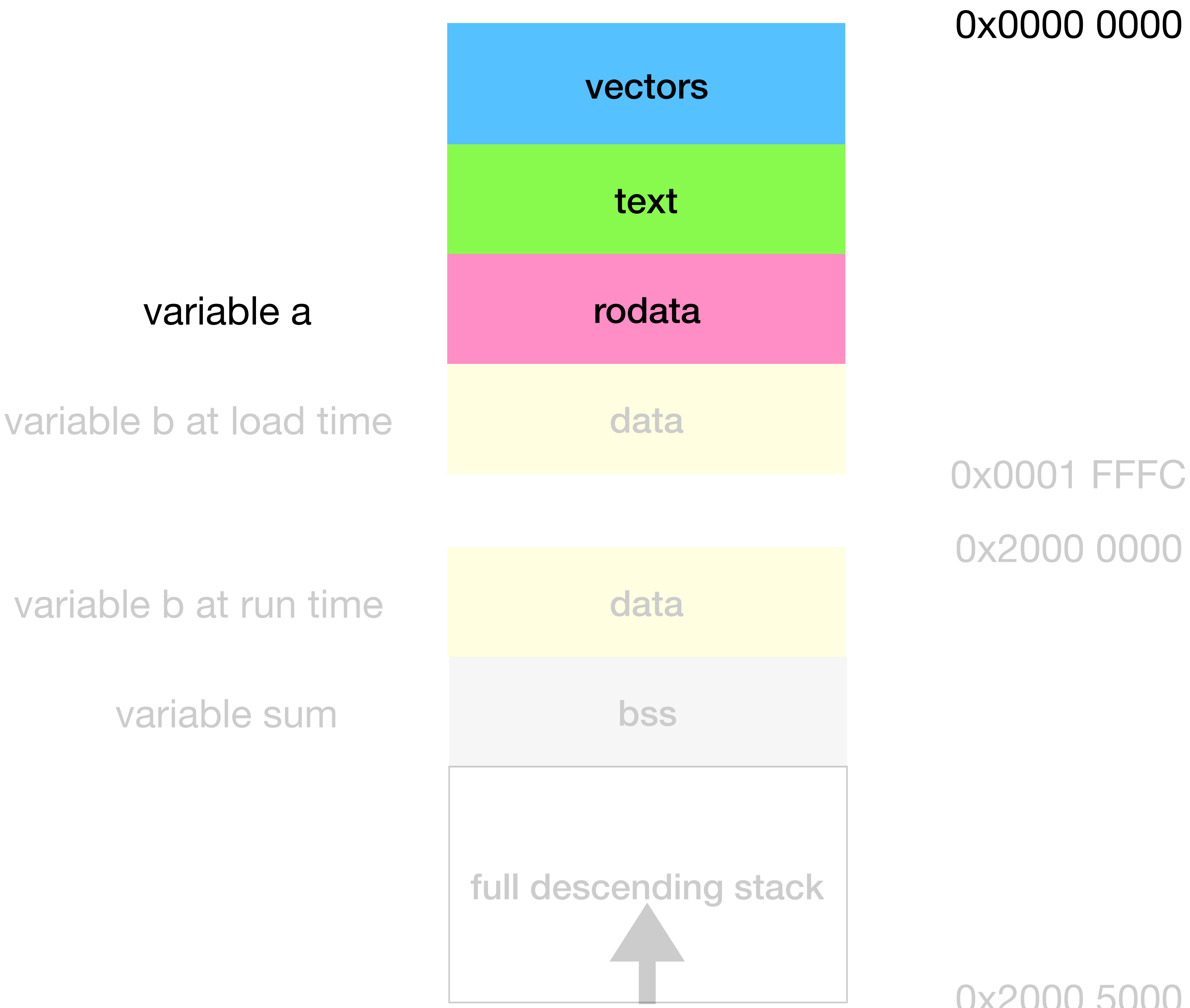
variable sum

bss

full descending stack

0x2000 5000

```
SECTIONS
{
  .  = 0x0;
  .text :
  {
    *(vectors)
    *(.text)
  }
  .rodata :
  {
    *(.rodata)
  }
}
```



0x0000 0000

vectors

text

variable a

rodata

variable b at load time

data

0x0001 FFFC

0x2000 0000

variable b at run time

data

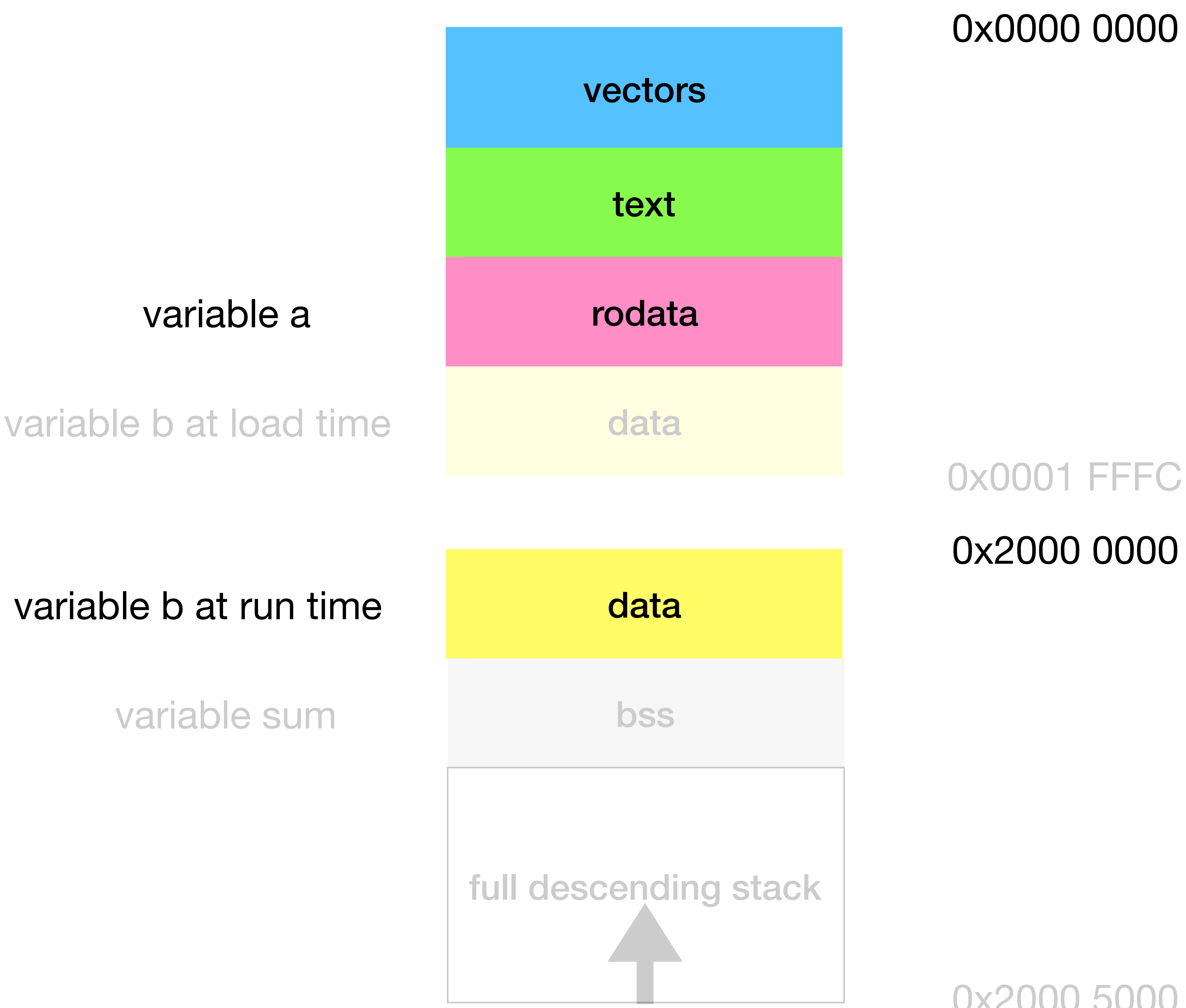
variable sum

bss

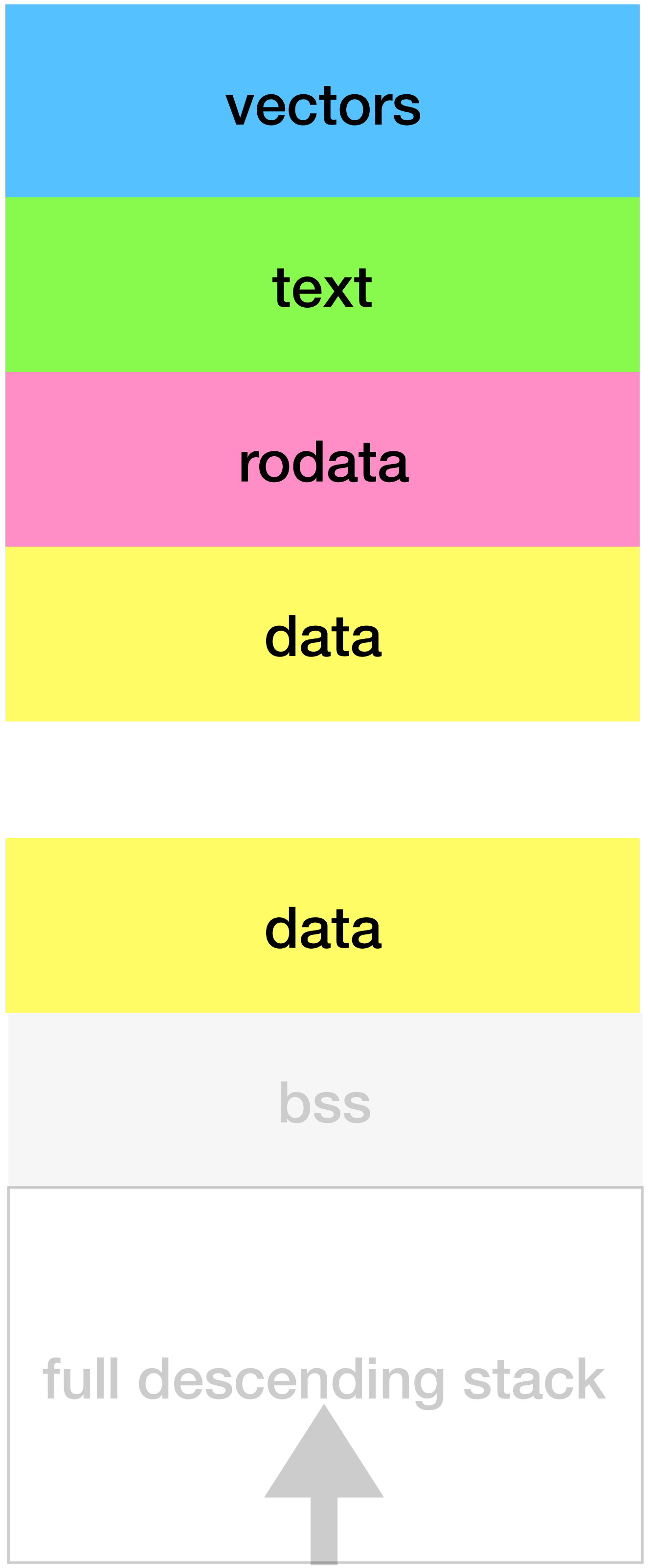
full descending stack

0x2000 5000

```
SECTIONS
{
  .  = 0x0;
  .text :
  {
    *(vectors)
    *(.text)
  }
  .rodata :
  {
    *(.rodata)
  }
  .  = 0x20000000;
  .data :
  {
    *(.data)
  }
}
```


0x0000 0000
vectors
text
variable a
rodata
variable b at load time
data
0x0001 FFFC
0x2000 0000
variable b at run time
data
variable sum
bss
full descending stack
0x2000 5000

```
SECTIONS
{
  .  = 0x0;
  .text :
  {
    *(vectors)
    *(.text)
  }
  .rodata :
  {
    *(.rodata)
  }
  _DATA_ROM_START = .;
  .  = 0x20000000;
  .data : AT(_DATA_ROM_START)
  {
    *(.data)
  }
}
```



0x0000 0000

vectors

text

variable a

rodata

variable b at load time

data

0x0001 FFFC

0x2000 0000

variable b at run time

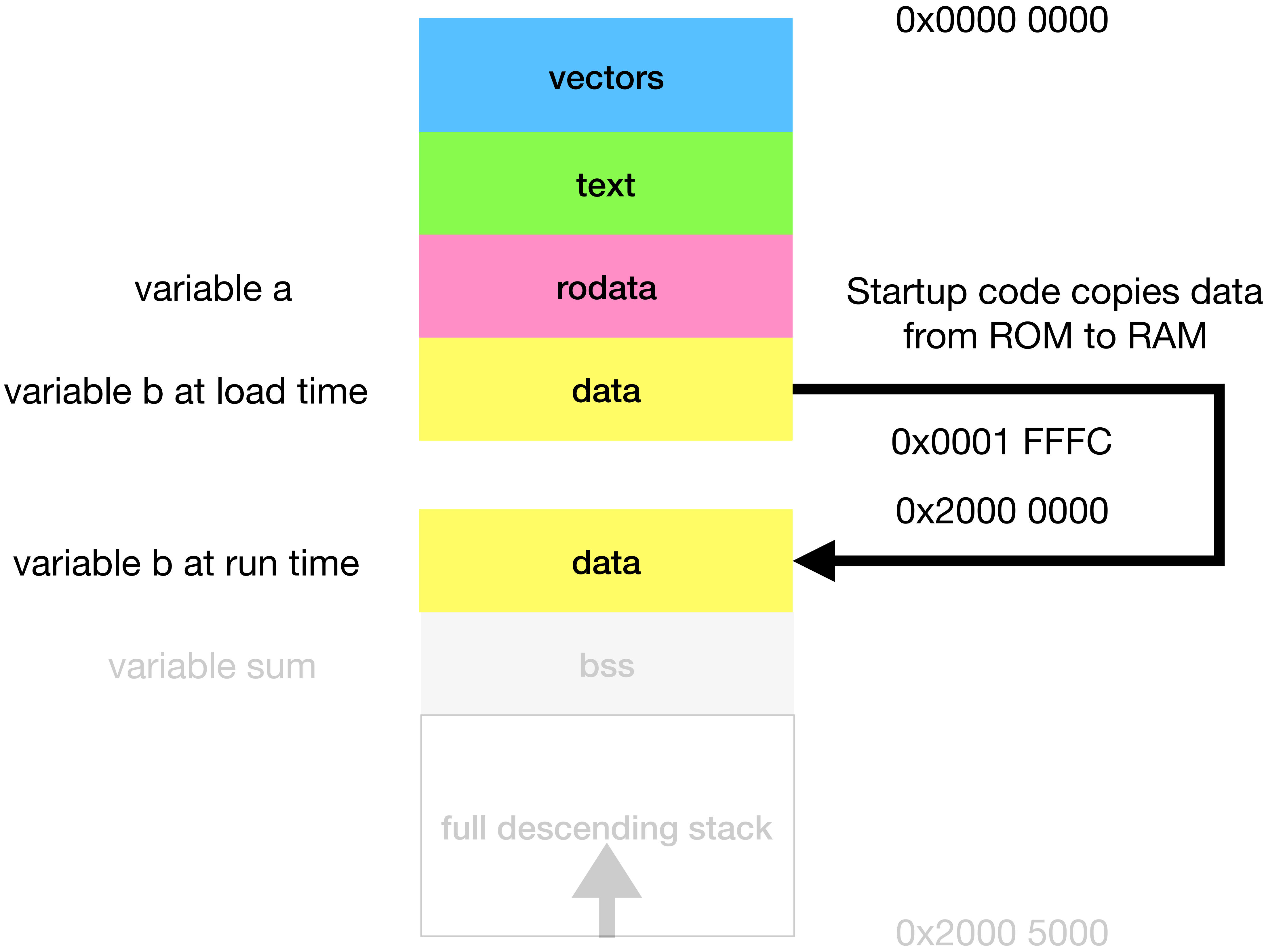
data

variable sum

bss

full descending stack

0x2000 5000



0x0000 0000

vectors

text

variable a

rodata

Startup code copies data
from ROM to RAM

variable b at load time

data

0x0001 FFFC

0x2000 0000

variable b at run time

data

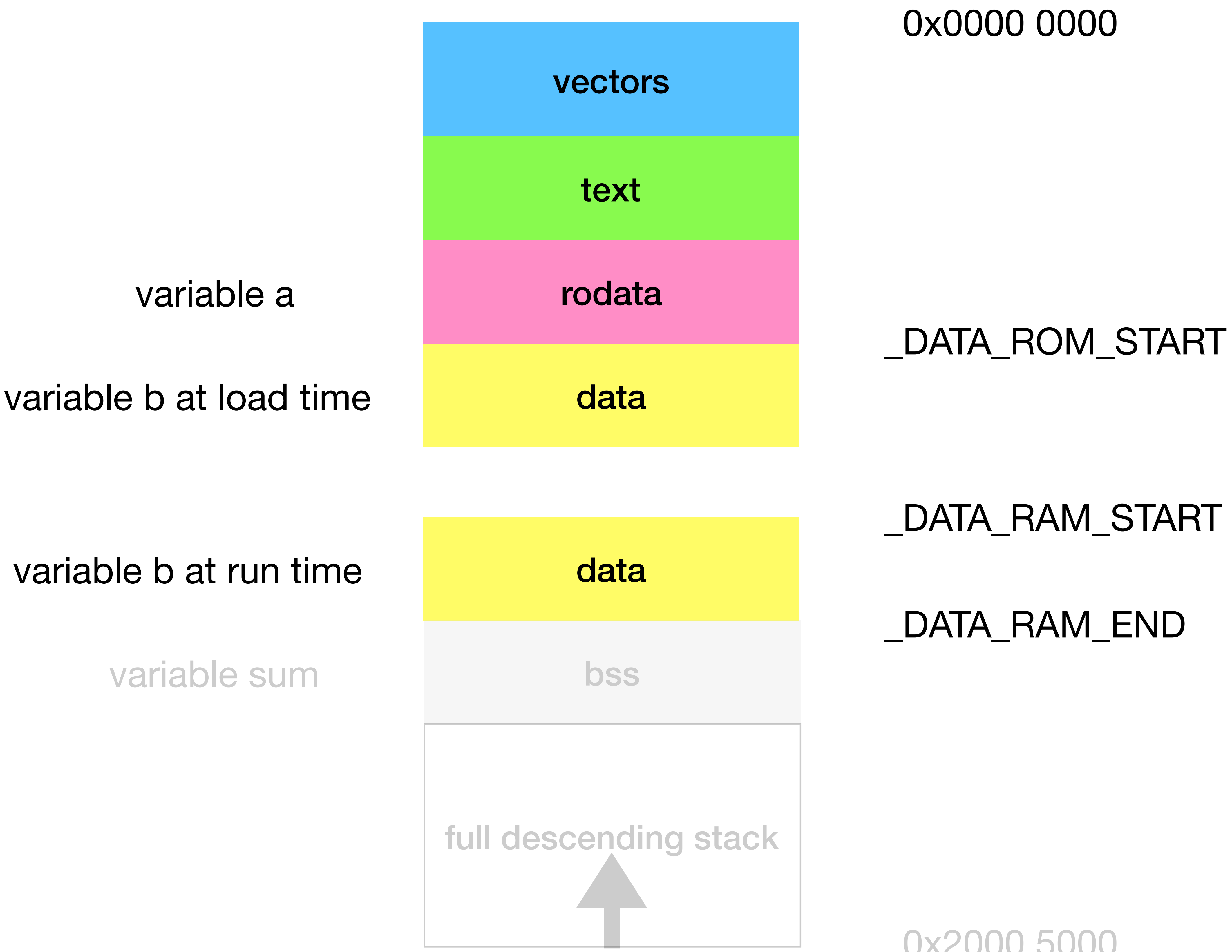
variable sum

bss

full descending stack

0x2000 5000

```
SECTIONS
{
  .  = 0x0;
  .text :
  {
    *(vectors)
    *(.text)
  }
  .rodata :
  {
    *(.rodata)
  }
  _DATA_ROM_START = .;
  .  = 0x20000000;
  _DATA_RAM_START = .;
  .data : AT(_DATA_ROM_START)
  {
    *(.data)
  }
  _DATA_RAM_END = .;
}
```



0x0000 0000

vectors

text

variable a

rodata

_DATA_ROM_START

variable b at load time

data

_DATA_RAM_START

variable b at run time

data

_DATA_RAM_END

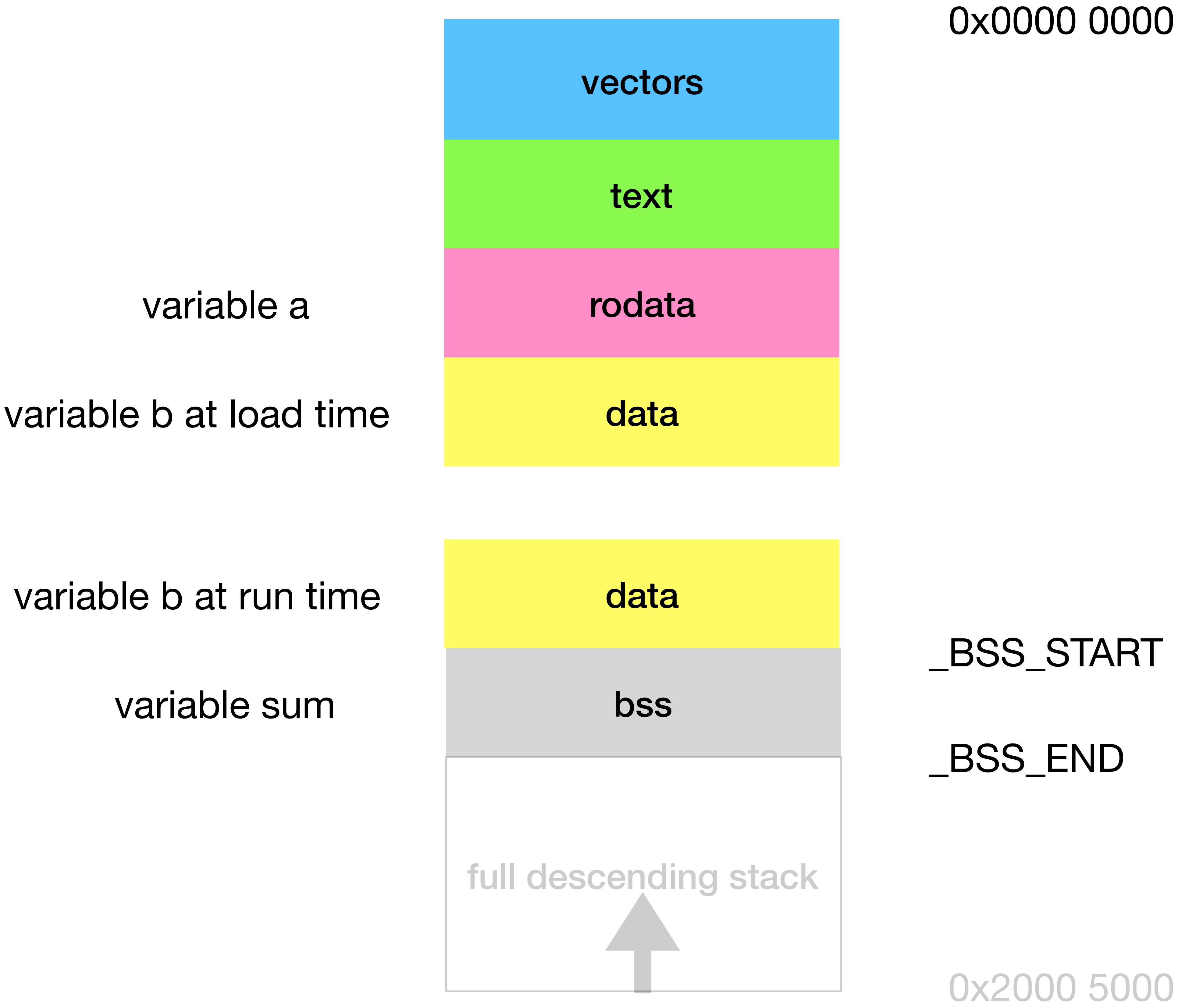
variable sum

bss

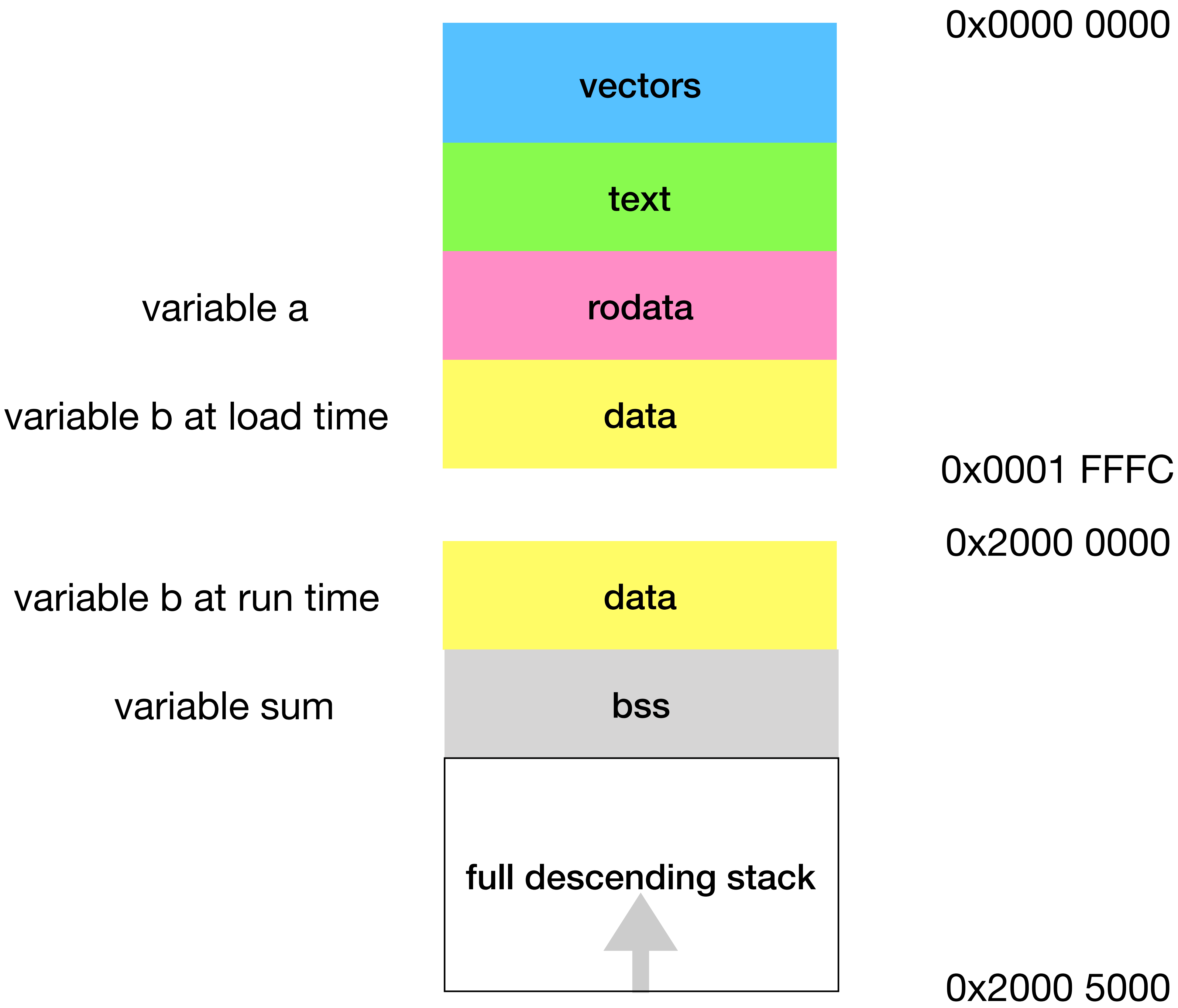
full descending stack

0x2000 5000

```
SECTIONS
{
  .  = 0x0;
  .text :
  {
    *(vectors)
    *(.text)
  }
  .rodata :
  {
    *(.rodata)
  }
  _DATA_ROM_START = .;
  .  = 0x20000000;
  _DATA_RAM_START = .;
  .data : AT(_DATA_ROM_START)
  {
    *(.data)      /* Data memory */
  }
  _DATA_RAM_END = .;
  _BSS_START = .;
  .bss :
  {
    *(.bss)
  }
  _BSS_END = .;
}
```


0x0000 0000
vectors
text
variable a
rodata
variable b at load time
data
variable b at run time
data
_BSS_START
variable sum
bss
_BSS_END
full descending stack
0x2000 5000

```
SECTIONS
{
  .  = 0x0;
  .text :
  {
    *(vectors)
    *(.text)
  }
  .rodata :
  {
    *(.rodata)
  }
  _DATA_ROM_START = .;
  .  = 0x20000000;
  _DATA_RAM_START = .;
  .data : AT(_DATA_ROM_START)
  {
    *(.data)      /* Data memory */
  }
  _DATA_RAM_END = .;
  _BSS_START = .;
  .bss :
  {
    *(.bss)
  }
  _BSS_END = .;
}
```



0x0000 0000

vectors

text

variable a

rodata

variable b at load time

data

0x0001 FFFC

0x2000 0000

variable b at run time

data

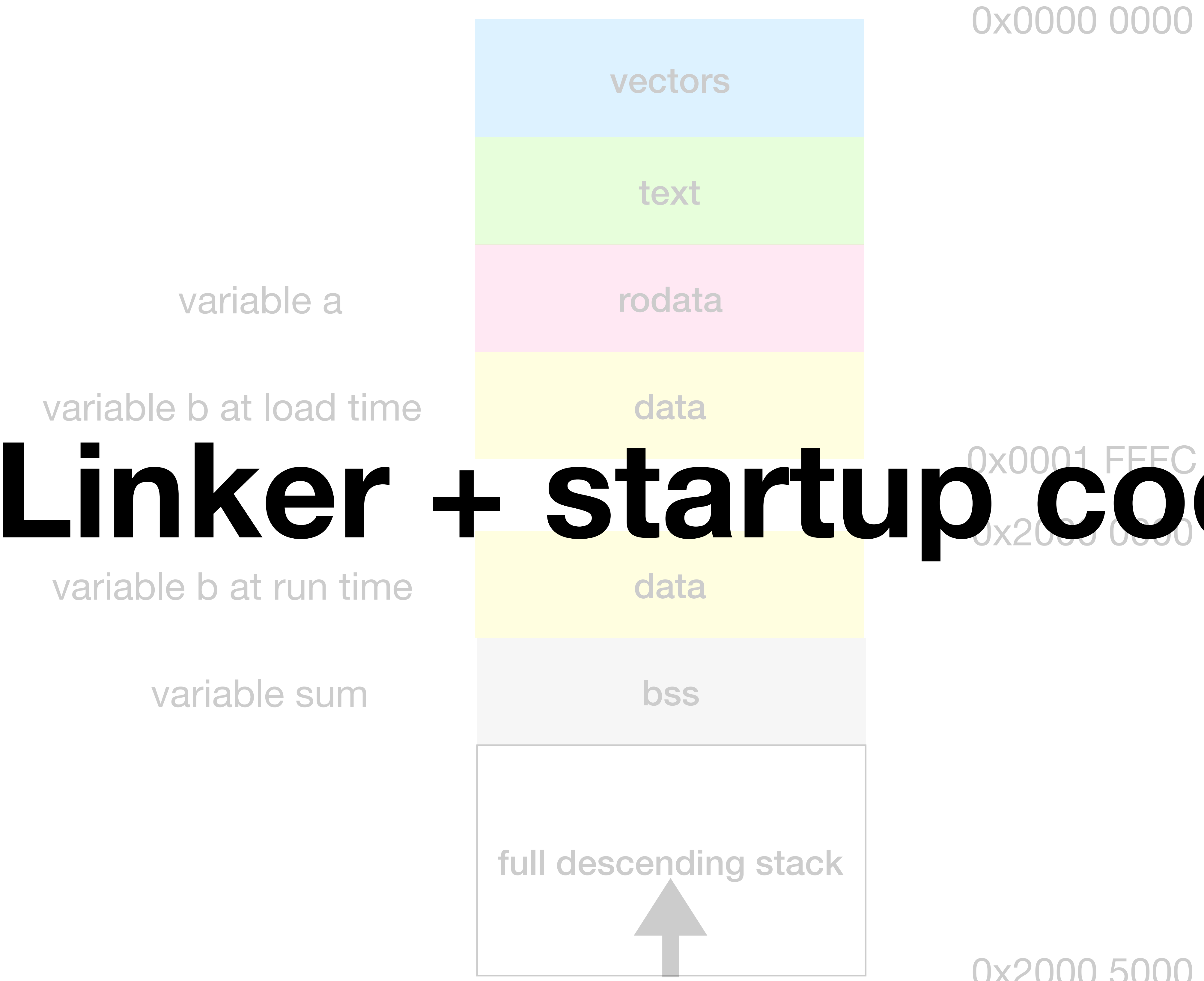
variable sum

bss

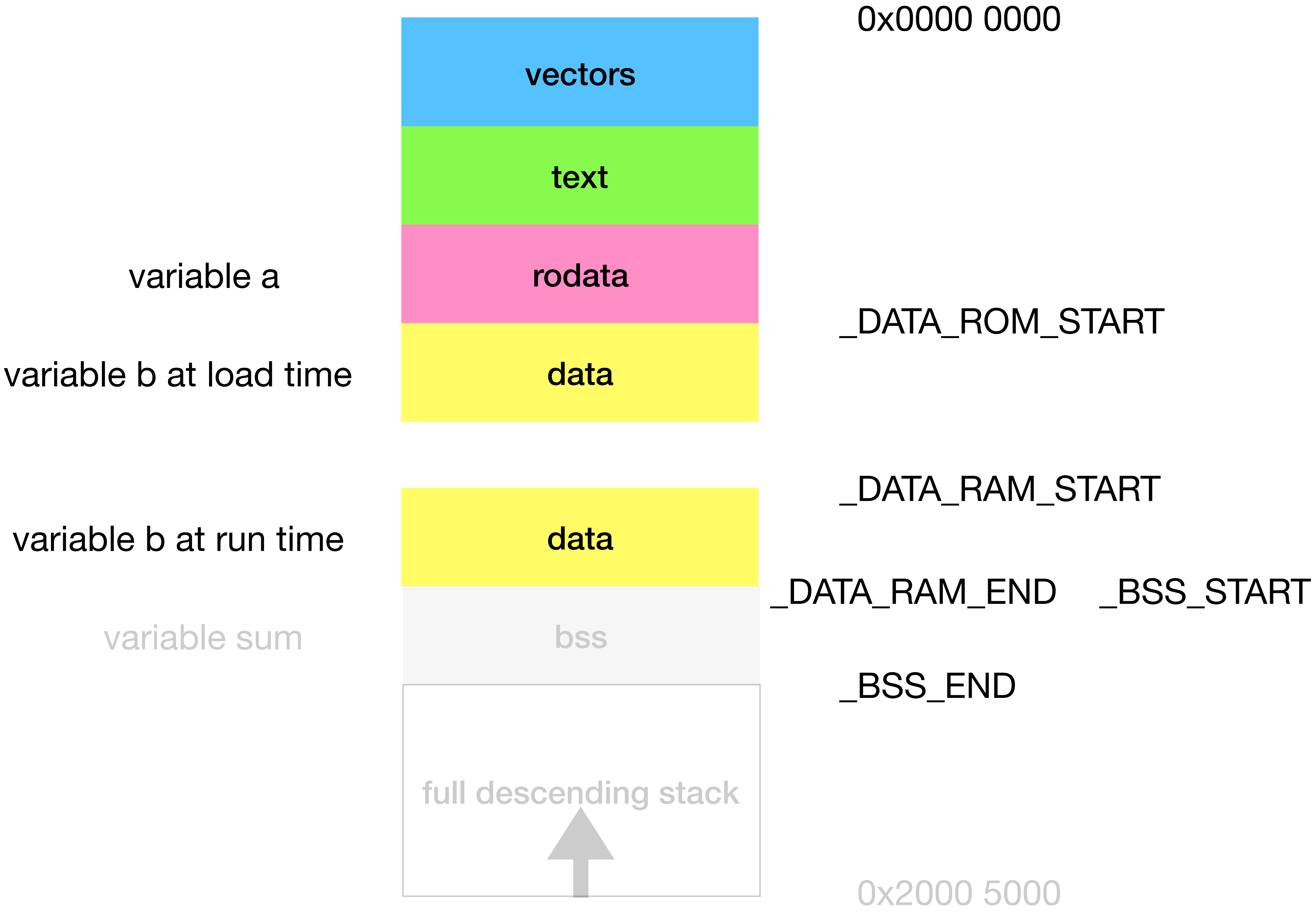
full descending stack

0x2000 5000

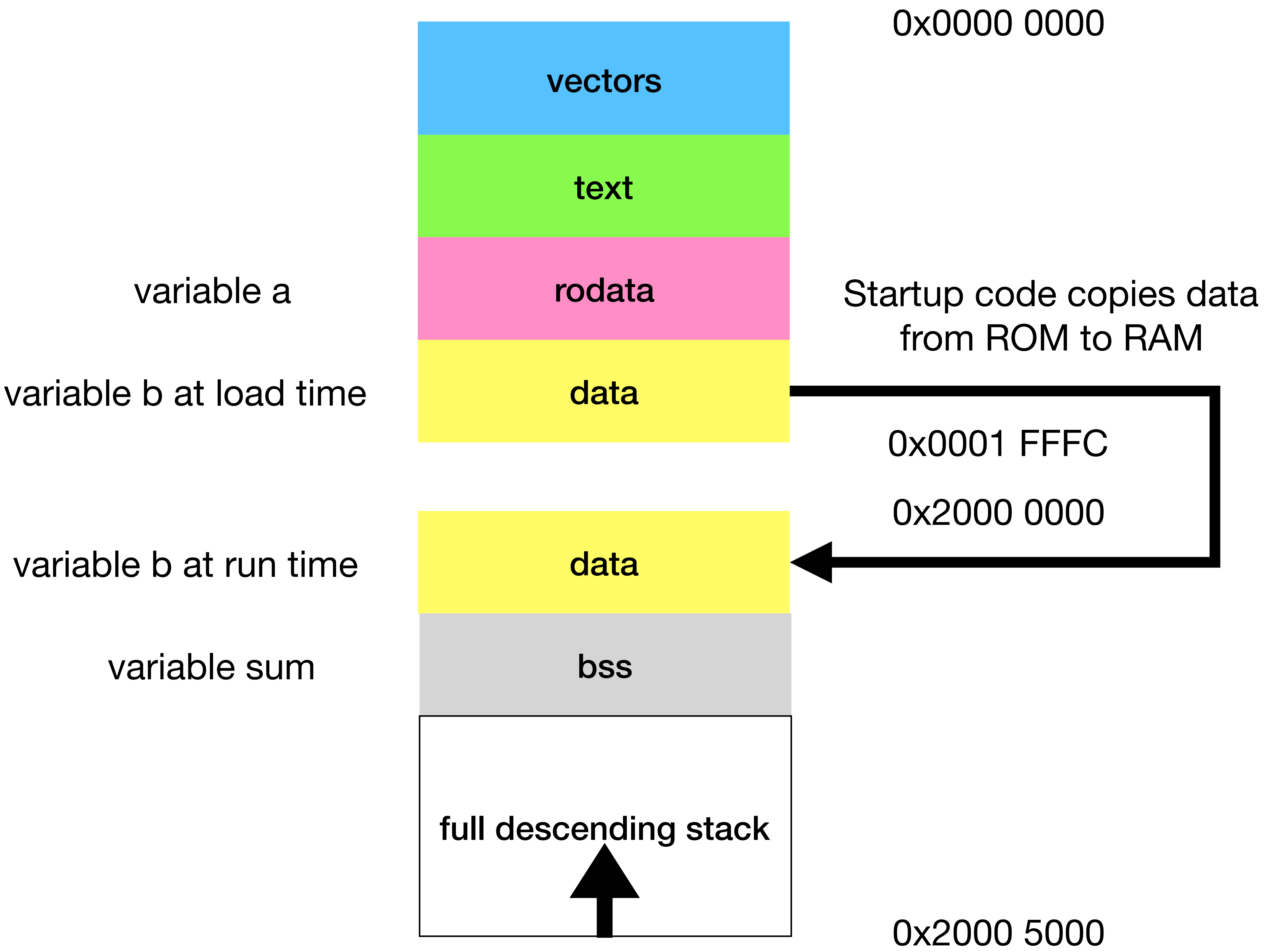
Linker + startup code



```
#define STACK_TOP 0x20005000
void startup();

unsigned int * myvectors[2]
__attribute__ ((section("vectors")))= {
    (unsigned int *)    STACK_TOP,
    (unsigned int *)    startup
};
```


0x0000 0000
vectors
text
variable a
rodata
_DATA_ROM_START
variable b at load time
data
_DATA_RAM_START
variable b at run time
data
_DATA_RAM_END
_BSS_START
variable sum
bss
_BSS_END
full descending stack
0x2000 5000

```
#define STACK_TOP 0x20005000
void startup();

unsigned int * myvectors[2]
__attribute__ ((section("vectors")))= {
    (unsigned int *)    STACK_TOP,
    (unsigned int *)    startup
};

extern unsigned int _DATA_ROM_START;
extern unsigned int _DATA_RAM_START;
extern unsigned int _DATA_RAM_END;
extern unsigned int _BSS_START;
extern unsigned int _BSS_END;
```



0x0000 0000

vectors

text

variable a

rodata

Startup code copies data from ROM to RAM

variable b at load time

data

0x0001 FFFC

0x2000 0000

variable b at run time

data

variable sum

bss

full descending stack

0x2000 5000

```c
void startup()
{
    /* Copy data belonging to the `.data` section from its
     * load time position on flash (ROM) to its run time
     * position in SRAM.
     */
    unsigned int * data_rom_start_p = &_DATA_ROM_START;
    unsigned int * data_ram_start_p = &_DATA_RAM_START;
    unsigned int * data_ram_end_p = &_DATA_RAM_END;

    while(data_ram_start_p != data_ram_end_p)
    {
        *data_ram_start_p = *data_rom_start_p;
        data_ram_start_p++;
        data_rom_start_p++;
    }
}
```

Assembler and C prerequsites

✓ A. Provide reset vector with start address for reset exception

✓ B. Initialize stack pointer

✓ C. Put the reset vector at address 0x0

✓ D. Put the .text section after the reset vector in flash

✓ E. Make the immutable data in the **.rodata** section available in the read only memory

✓ F. Make the mutable data in the **.data** section available in the read/write memory

G. Make the **.bss** section available in the read/write memory too. Also make sure all memory in the .bss section is initialized to zero.

Assembler and C prerequsites

✓ A. Provide reset vector with start address for reset exception

✓ B. Initialize stack pointer

✓ C. Put the reset vector at address 0x0

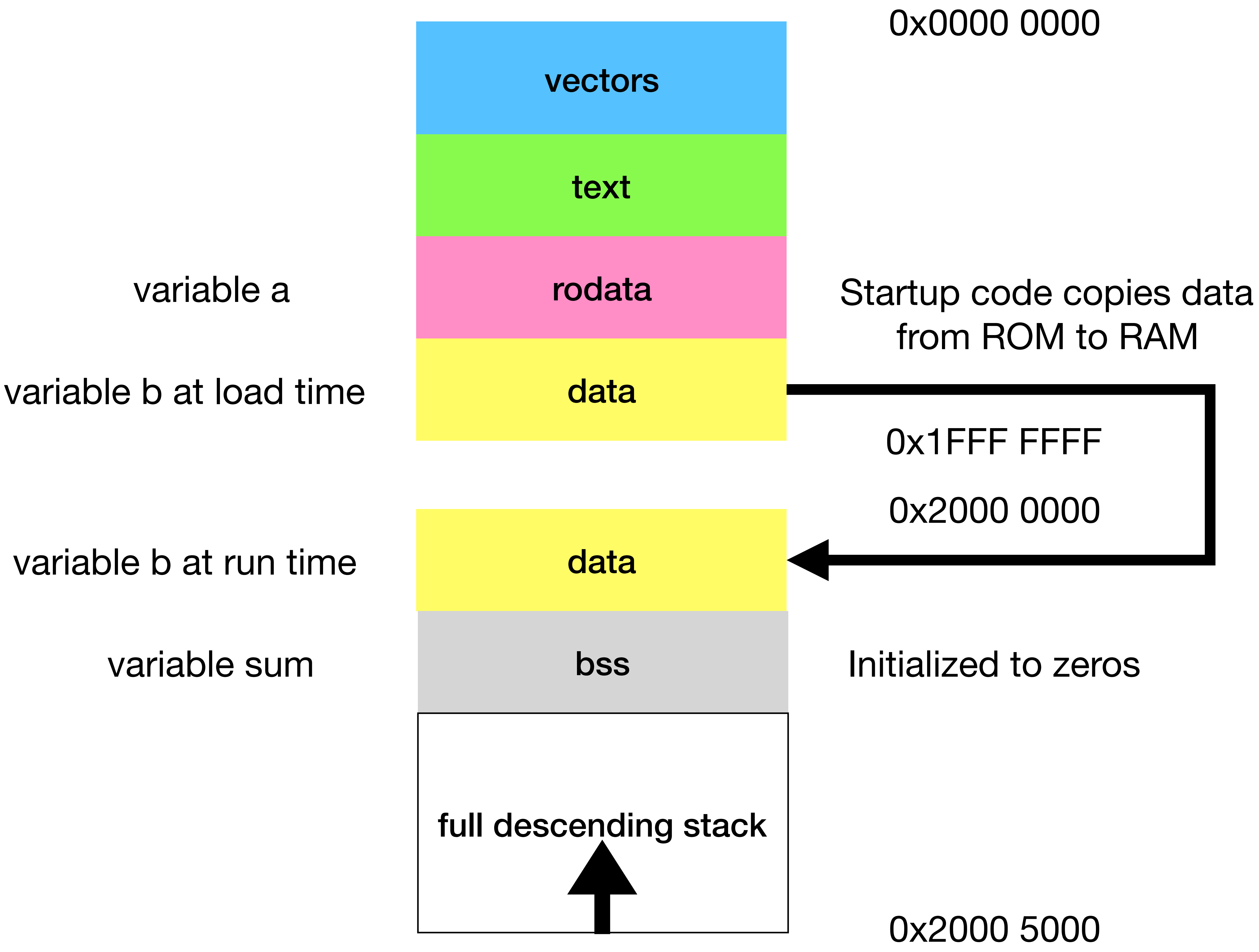
✓ D. Put the .text section after the reset vector in flash

✓ E. Make the immutable data in the **.rodata** section available in the read only memory

✓ F. Make the mutable data in the **.data** section available in the read/write memory

G. Make the **.bss** section available in the read/write memory too. **Also make sure all memory in the .bss section is initialized to zero.**

```c
void startup()
{
    /* Copy data belonging to the `.data` section from its
     * load time position on flash (ROM) to its run time
     * position in SRAM.
     */
    unsigned int * data_rom_start_p = &_DATA_ROM_START;
    unsigned int * data_ram_start_p = &_DATA_RAM_START;
    unsigned int * data_ram_end_p = &_DATA_RAM_END;

    while(data_ram_start_p != data_ram_end_p)
    {
        *data_ram_start_p = *data_rom_start_p;
        data_ram_start_p++;
        data_rom_start_p++;
    }

    /* Initialize data in the `.bss` section to zeros.
     */
    unsigned int * bss_start_p = &_BSS_START;
    unsigned int * bss_end_p = &_BSS_END;

    while(bss_start_p != bss_end_p)
    {
        *bss_start_p = 0;
        bss_start_p++;
    }

}
```


0x0000 0000
vectors
text
variable a
rodata
Startup code copies data from ROM to RAM
variable b at load time
data
0x1FFF FFFF
0x2000 0000
variable b at run time
data
variable sum
bss
Initialized to zeros
full descending stack
0x2000 5000

```
#define STACK_TOP 0x20005000
void startup();

unsigned int * myvectors[2]
__attribute__ ((section("vectors")))= {
    (unsigned int *)    STACK_TOP,
    (unsigned int *)    startup
};

extern unsigned int _DATA_ROM_START;
extern unsigned int _DATA_RAM_START;
extern unsigned int _DATA_RAM_END;
extern unsigned int _BSS_START;
extern unsigned int _BSS_END;

void startup()
{
    /* Copy data belonging to the `.data` section from its
     * load time position on flash (ROM) to its run time
     * position in SRAM.
     */
    unsigned int * data_rom_start_p = &_DATA_ROM_START;
    unsigned int * data_ram_start_p = &_DATA_RAM_START;
    unsigned int * data_ram_end_p = &_DATA_RAM_END;

    while(data_ram_start_p != data_ram_end_p)
    {
        *data_ram_start_p = *data_rom_start_p;
        data_ram_start_p++;
        data_rom_start_p++;
    }

    /* Initialize data in the `.bss` section to zeros.
     */
    unsigned int * bss_start_p = &_BSS_START;
    unsigned int * bss_end_p = &_BSS_END;

    while(bss_start_p != bss_end_p)
    {
        *bss_start_p = 0;
        bss_start_p++;
    }

}
```

```c
#define STACK_TOP 0x20005000
void startup();

unsigned int * myvectors[2]
__attribute__ ((section("vectors")))= {
    (unsigned int *)    STACK_TOP,
    (unsigned int *)    startup
};
```

vector table

```c
extern unsigned int _DATA_ROM_START;
extern unsigned int _DATA_RAM_START;
extern unsigned int _DATA_RAM_END;
extern unsigned int _BSS_START;
extern unsigned int _BSS_END;
```

symbols from linker script

```c
void startup()
{
    /* Copy data belonging to the `.data` section from its
     * load time position on flash (ROM) to its run time
     * position in SRAM.
     */
    unsigned int * data_rom_start_p = &_DATA_ROM_START;
    unsigned int * data_ram_start_p = &_DATA_RAM_START;
    unsigned int * data_ram_end_p = &_DATA_RAM_END;

    while(data_ram_start_p != data_ram_end_p)
    {
        *data_ram_start_p = *data_rom_start_p;
        data_ram_start_p++;
        data_rom_start_p++;
    }
```

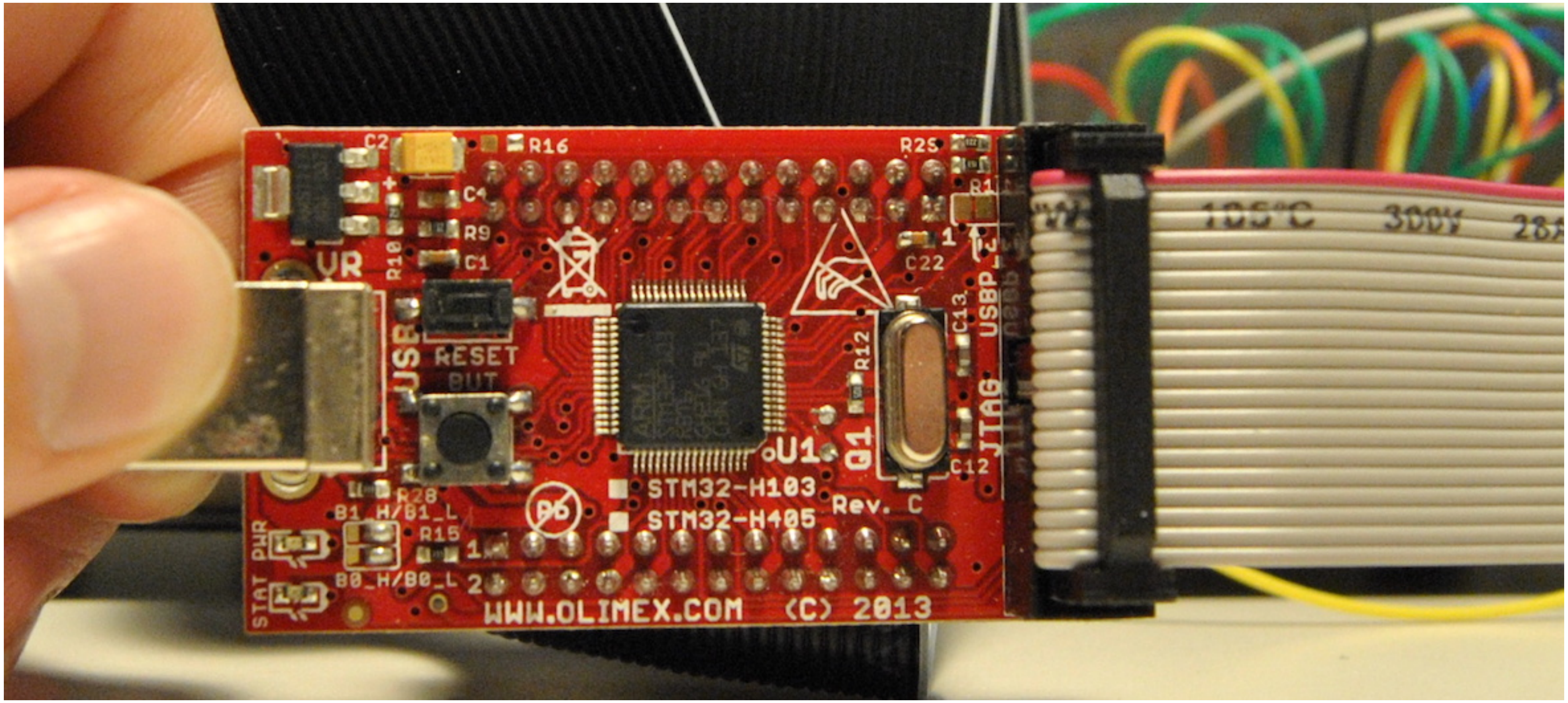
copy data from ROM to RAM

```c
    /* Initialize data in the `.bss` section to zeros.
     */
    unsigned int * bss_start_p = &_BSS_START;
    unsigned int * bss_end_p = &_BSS_END;

    while(bss_start_p != bss_end_p)
    {
        *bss_start_p = 0;
        bss_start_p++;
    }

}
```

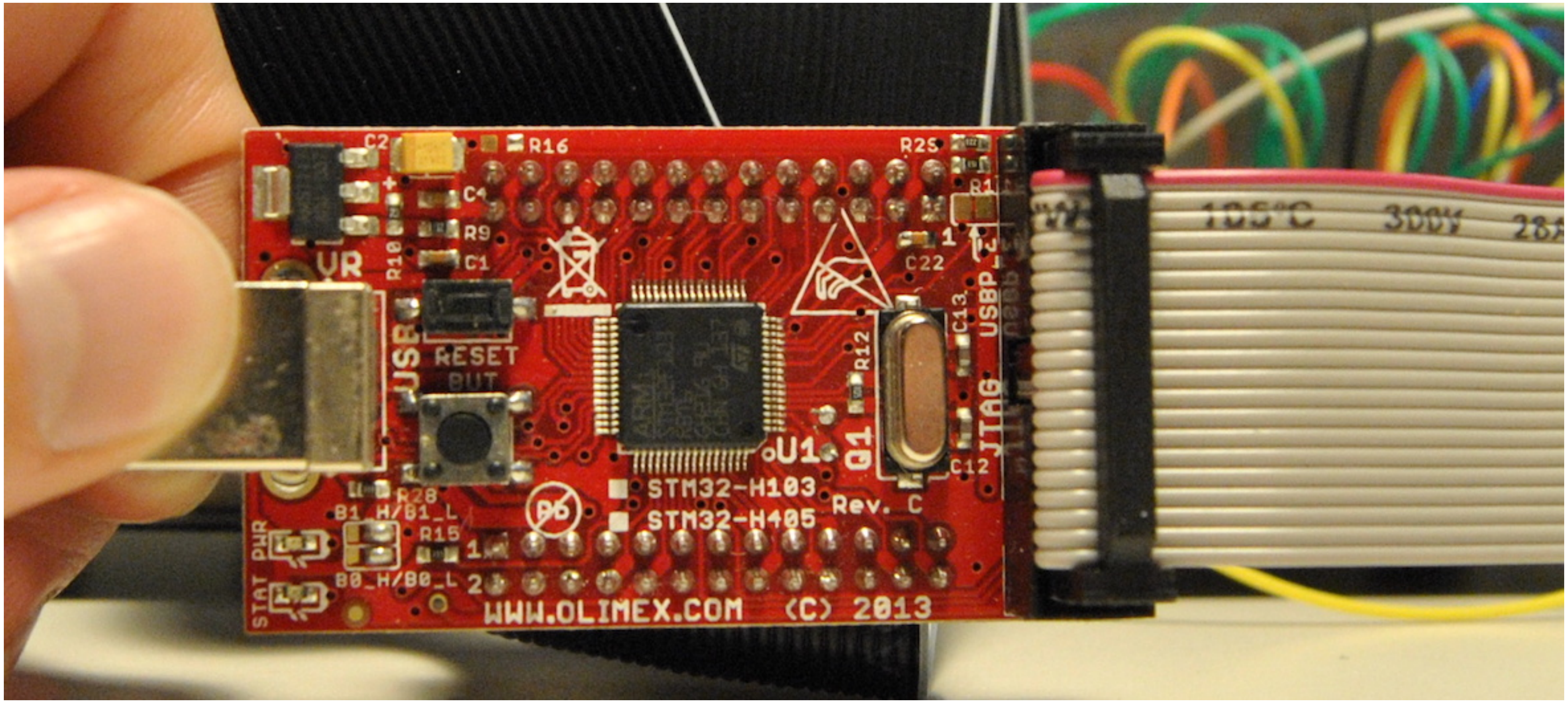
initialize bss to zeros

But there is one more thing...

```
static const int a = 7;
static int b = 8;
static int sum;

void main()
{
  sum = a + b;
}
```



```
static const int a = 7;
static int b = 8;
static int sum;

void main()
{
  sum = a + b;
}
```



```
void startup()
{
    /* Copy data belonging to the `.data` section from its
     * load time position on flash (ROM) to its run time
     * position in SRAM.
     */
    unsigned int * data_rom_start_p = &_DATA_ROM_START;
    unsigned int * data_ram_start_p = &_DATA_RAM_START;
    unsigned int * data_ram_end_p = &_DATA_RAM_END;

    while(data_ram_start_p != data_ram_end_p)
    {
        *data_ram_start_p = *data_rom_start_p;
        data_ram_start_p++;
        data_rom_start_p++;
    }

    /* Initialize data in the `.bss` section to zeros.
     */
    unsigned int * bss_start_p = &_BSS_START;
    unsigned int * bss_end_p = &_BSS_END;

    while(bss_start_p != bss_end_p)
    {
        *bss_start_p = 0;
        bss_start_p++;
    }

    main();

}
```

Compile

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o test_program.o test_program.c
```

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o startup.o startup.c
```

Compile

-O0: no optimization

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o test_program.o test_program.c
```

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o startup.o startup.c
```

Compile

-O0: no optimization

-c: compile, but do not link

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o test_program.o test_program.c
```

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o startup.o startup.c
```

Compile

-O0: no optimization

-g: debugging info

-c: compile, but do not link

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o test_program.o test_program.c
```

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o startup.o startup.c
```

Compile

-O0: no optimization

-g: debugging info

-c: compile, but do not link

-mcpu=cortex-m3 -mthumb: cpu type

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o test_program.o test_program.c
```

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o startup.o startup.c
```

Compile

-O0: no optimization

-g: debugging info

-c: compile, but do not link

-mcpu=cortex-m3 -mthumb: cpu type

-o <file>: output file

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o test_program.o test_program.c
```

```
arm-none-eabi-gcc -O0 -c -g -mcpu=cortex-m3 -mthumb \
                  -o startup.o startup.c
```

Link

```
arm-none-eabi-ld -Tstm32.ld \
                 -o test_program.elf
                 startup.o test_program.o
```

Link

-Tstm32.ld: use linker script stm32.ld

```
arm-none-eabi-ld -Tstm32.ld \
                 -o test_program.elf
                 startup.o test_program.o
```

Link

`-Tstm32.ld: use linker script stm32.ld`

`-o <file>: output file`

```
arm-none-eabi-ld -Tstm32.ld \
                 -o test_program.elf
                 startup.o test_program.o
```

Convert to binary

```
arm-none-eabi-objcopy \
          -O binary \
          test_program.elf test_program.bin
```

Convert to binary

```
-O binary: set object format of output
           file to binary
```

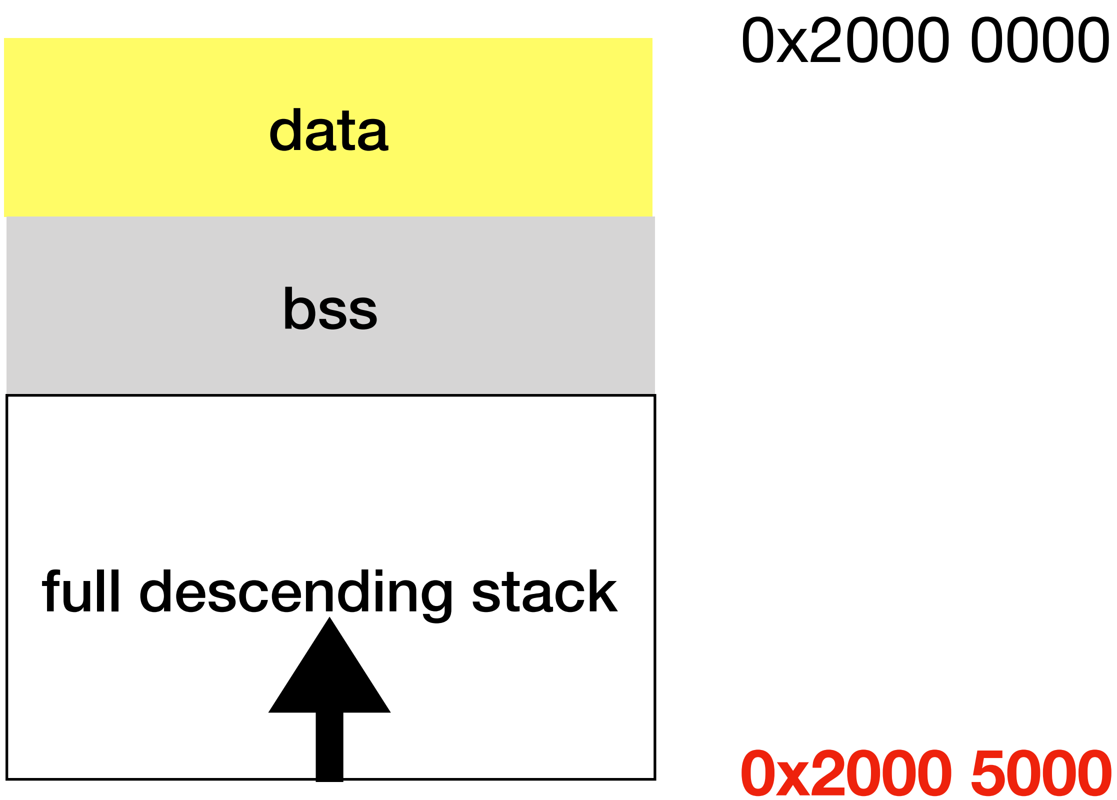
```
arm-none-eabi-objcopy \
          -O binary \
          test_program.elf test_program.bin
```

Inspect binary

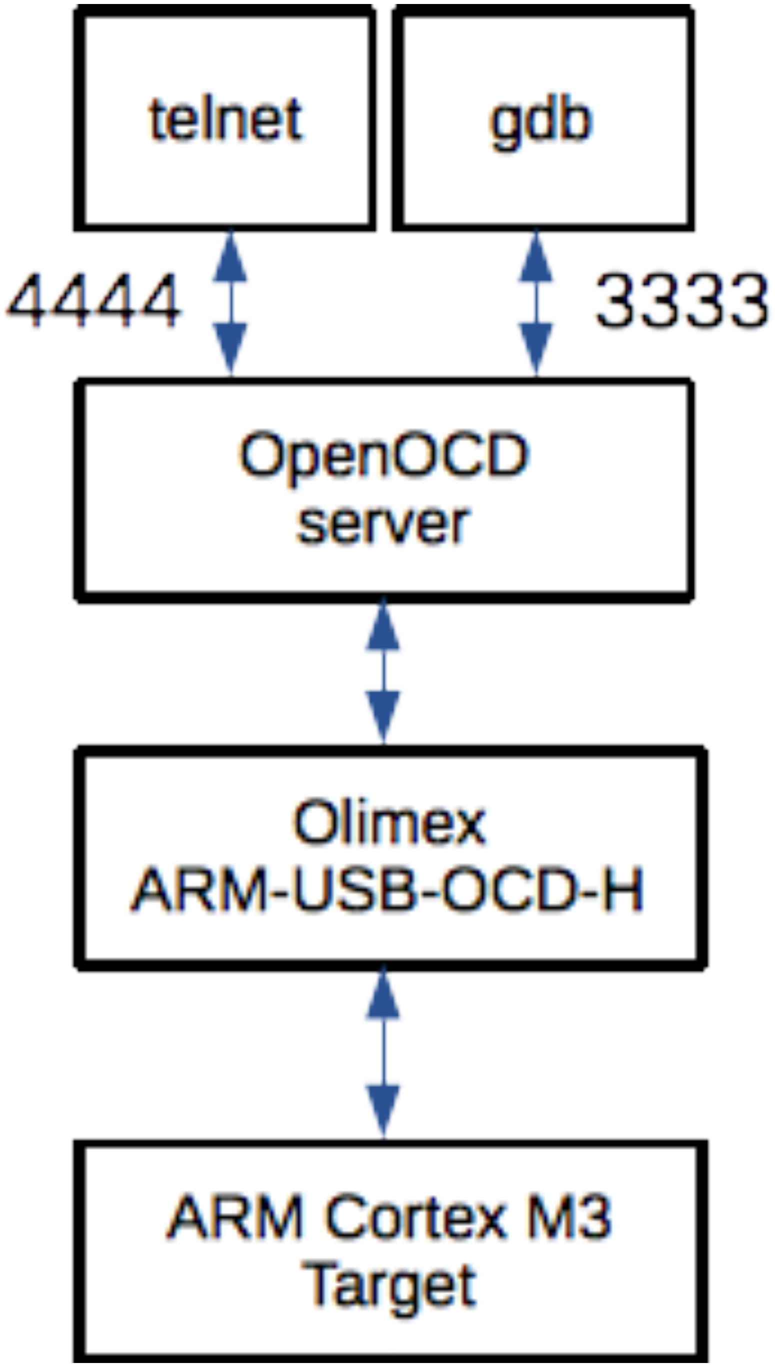
```
$ xxd -c 4 test_program.bin | head -n2
00000000: 0050 0020  .P.
00000004: 0900 0000  ....
```

Little Endian

Address	Content
0x03	**0x20**
0x02	**0x00**
0x01	**0x50**
0x00	**0x00**



Debugging



telnet

gdb

4444

3333

OpenOCD
server

Olimex
ARM-USB-OCD-H

ARM Cortex M3
Target

Debugging

```
$ openocd -f openocd.cfg
```

```
$ telnet localhost 4444
```

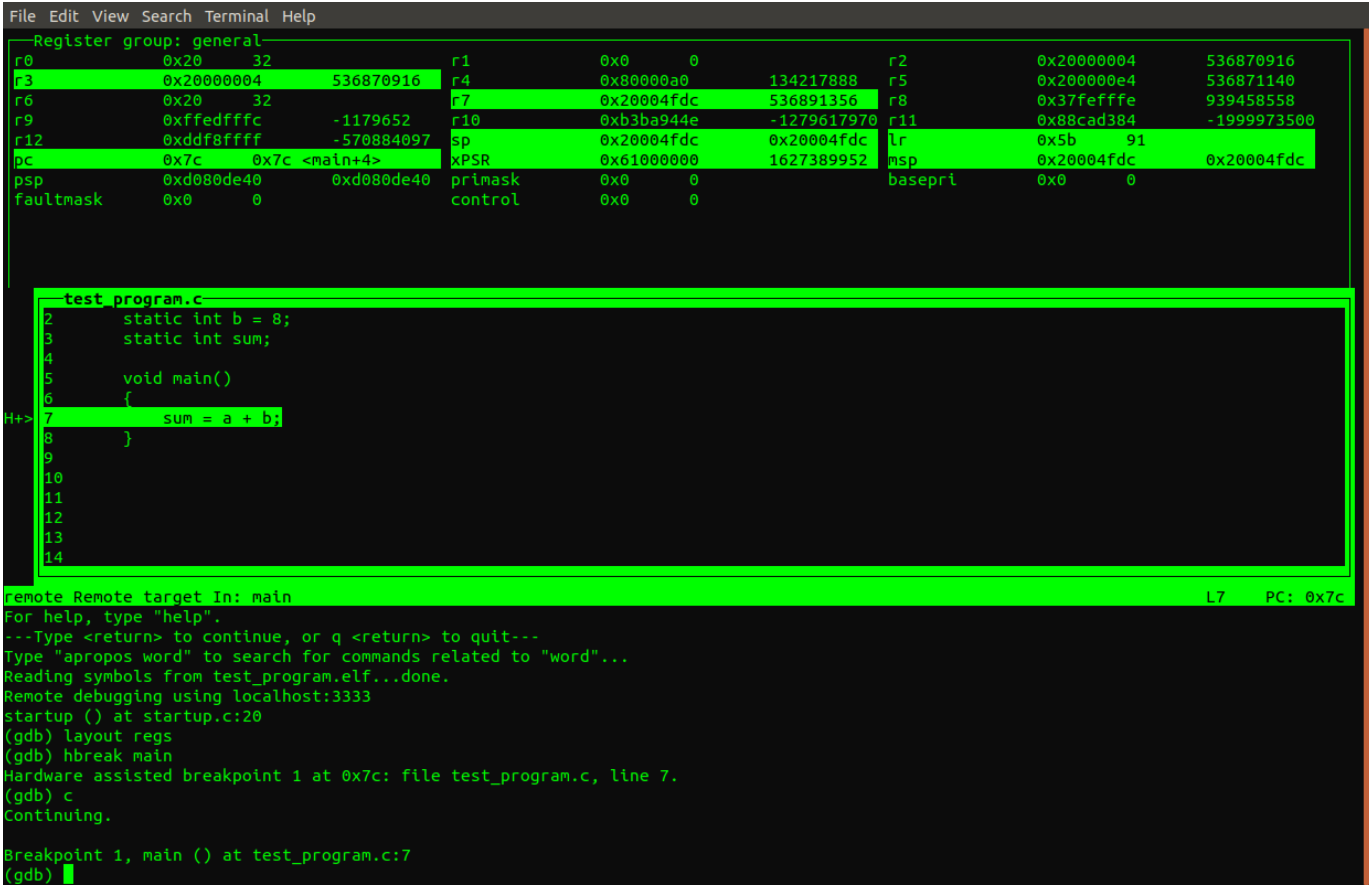
```
$ gdb-multiarch -tui --eval-command="target \
  remote localhost:3333" test_program.elf
```

Flashing

```
$ telnet localhost 4444
reset halt
stm32f1x mass_erase 0
flash write_bank 0 test_program.bin 0
reset halt
```

GDB

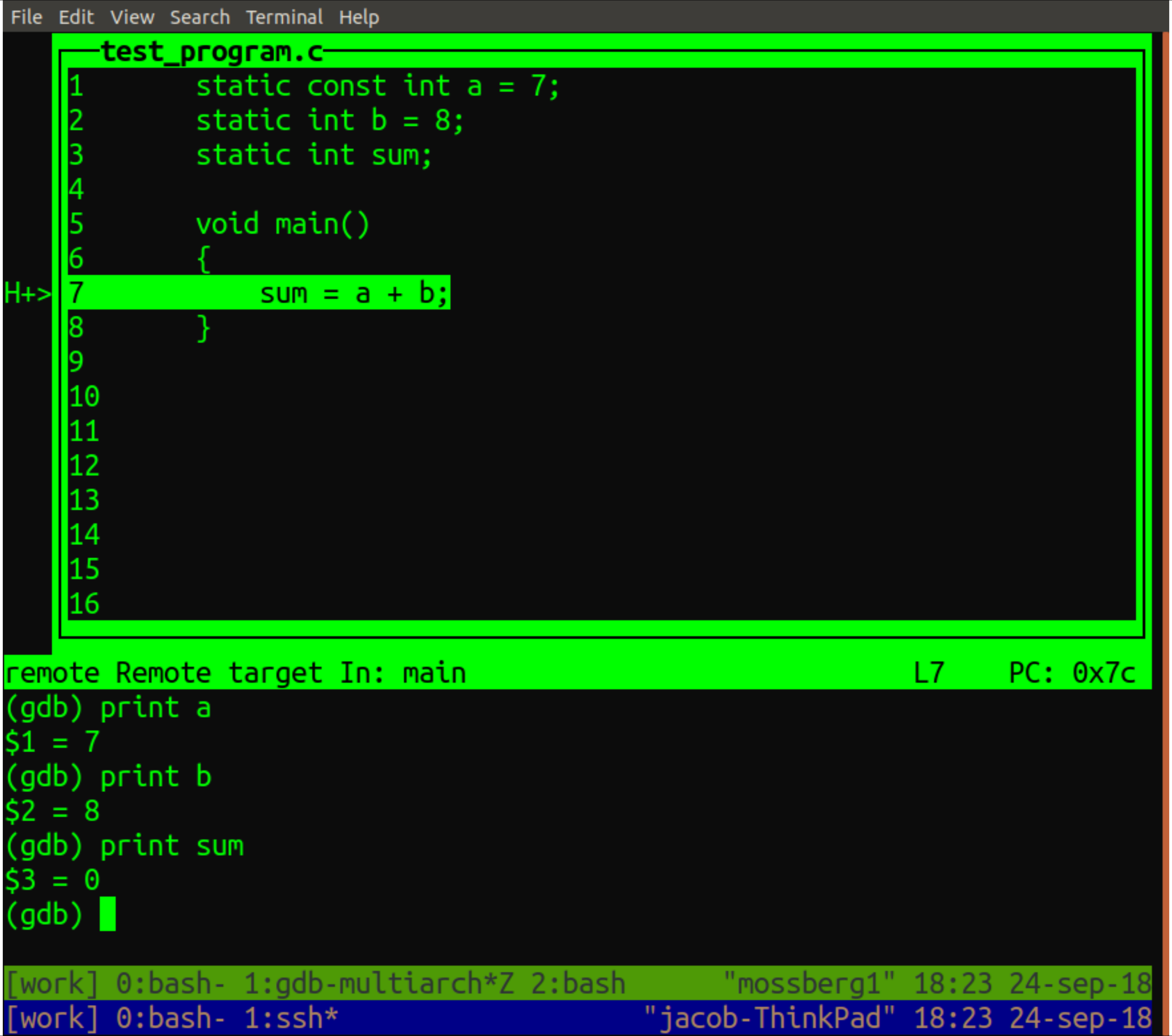
```
$ gdb-multiarch -tui --eval-command="target \
  remote localhost:3333" test_program.elf

(gdb) hbreak main
Hardware assisted breakpoint 1 at 0x7c: file
test_program.c, line 7.

(gdb) c
Continuing.

Breakpoint 1, main () at test_program.c:7
```



```
┌─Register group: general─────────────────────────────────────────────────────────────────────────────────┐
│r0             0x20     32                 r1             0x0      0                 r2             0x20000004       536870916  │
│r3             0x20000004       536870916  r4             0x80000a0        134217888 r5             0x200000e4       536871140  │
│r6             0x20     32                 r7             0x20004fdc       536891356 r8             0x37fefffe       939458558  │
│r9             0xffedfffc       -1179652   r10            0xb3ba944e       -1279617970 r11          0x88cad384       -1999973500│
│r12            0xddf8ffff       -570884097 sp             0x20004fdc       0x20004fdc lr            0x5b     91                │
│pc             0x7c     0x7c <main+4>      xPSR           0x61000000       1627389952 msp           0x20004fdc       0x20004fdc │
│psp            0xd080de40       0xd080de40 primask        0x0      0                 basepri        0x0      0                 │
│faultmask      0x0      0                  control        0x0      0                                                          │
│                                                                                                                              │
│                                                                                                                              │
  ┌─test_program.c──────────────────────────────────────────────────────────────────────────────────────┐
  │2       static int b = 8;                                                                             │
  │3       static int sum;                                                                               │
  │4                                                                                                     │
  │5       void main()                                                                                   │
  │6       {                                                                                             │
H+>│7           sum = a + b;                                                                             │
  │8       }                                                                                             │
  │9                                                                                                     │
  │10                                                                                                    │
  │11                                                                                                    │
  │12                                                                                                    │
  │13                                                                                                    │
  │14                                                                                                    │
  └──────────────────────────────────────────────────────────────────────────────────────────────────────┘
remote Remote target In: main                                                              L7    PC: 0x7c
For help, type "help".
---Type <return> to continue, or q <return> to quit---
Type "apropos word" to search for commands related to "word"...
Reading symbols from test_program.elf...done.
Remote debugging using localhost:3333
startup () at startup.c:20
(gdb) layout regs
(gdb) hbreak main
Hardware assisted breakpoint 1 at 0x7c: file test_program.c, line 7.
(gdb) c
Continuing.

Breakpoint 1, main () at test_program.c:7
(gdb) 
```

GDB

```
(gdb) print a
$1 = 7
(gdb) print b
$2 = 8
(gdb) print sum
$3 = 0
(gdb)
```



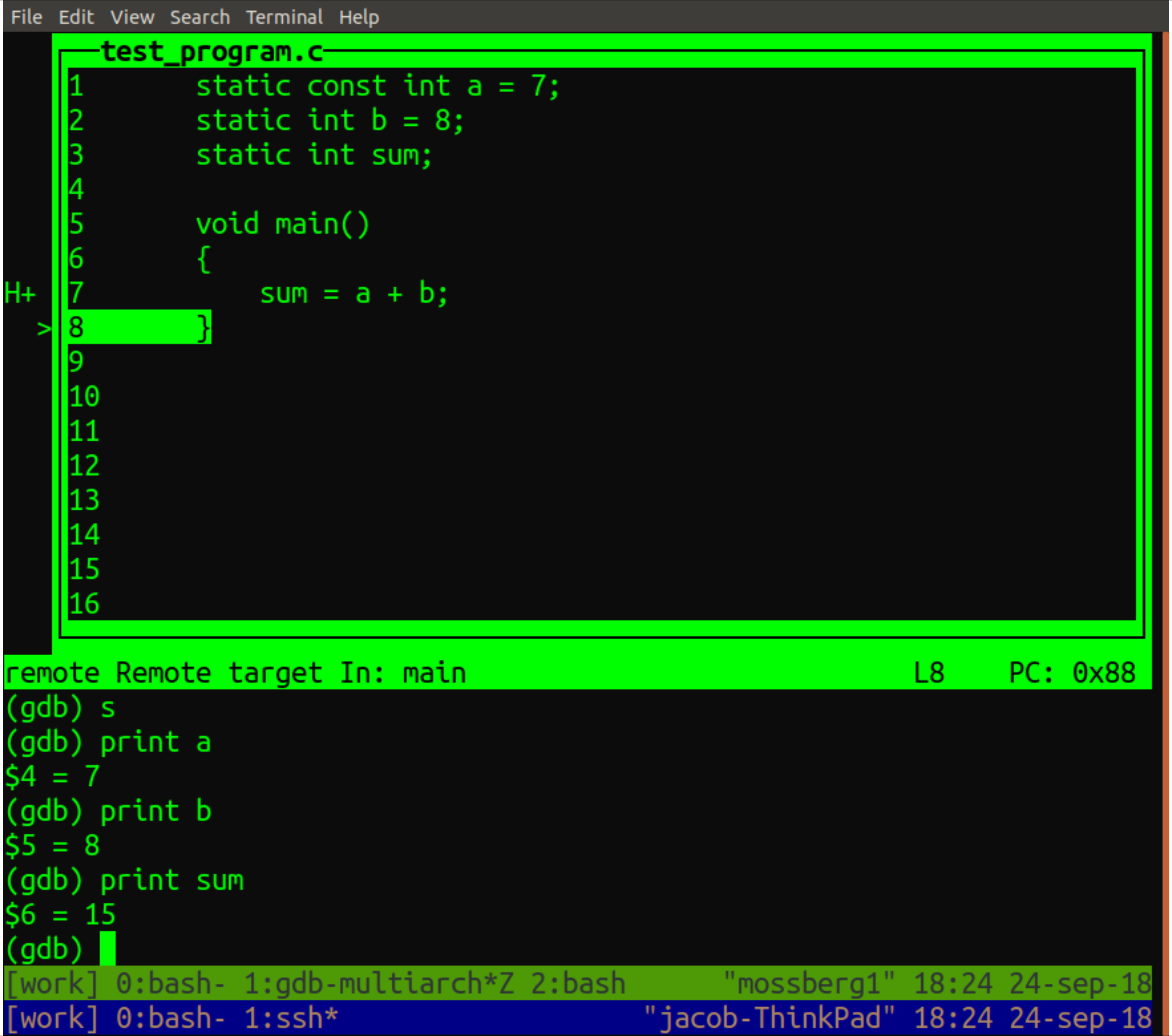
```
test_program.c
    static const int a = 7;
    static int b = 8;
    static int sum;

    void main()
    {
H+>     sum = a + b;
    }
```

```
remote Remote target In: main                          L7    PC: 0x7c
(gdb) print a
$1 = 7
(gdb) print b
$2 = 8
(gdb) print sum
$3 = 0
(gdb) 
```

```
[work] 0:bash- 1:gdb-multiarch*Z 2:bash     "mossberg1" 18:23 24-sep-18
[work] 0:bash- 1:ssh*                   "jacob-ThinkPad" 18:23 24-sep-18
```

GDB

```
(gdb) s
(gdb) print a
$4 = 7
(gdb) print b
$5 = 8
(gdb) print sum
$6 = 15
(gdb)
```



```
static const int a = 7;
static int b = 8;
static int sum;

void main()
{
    sum = a + b;
}
```

```
remote Remote target In: main                              L8     PC: 0x88
(gdb) s
(gdb) print a
$4 = 7
(gdb) print b
$5 = 8
(gdb) print sum
$6 = 15
(gdb)
[work] 0:bash- 1:gdb-multiarch*Z 2:bash      "mossberg1" 18:24 24-sep-18
[work] 0:bash- 1:ssh*                   "jacob-ThinkPad" 18:24 24-sep-18
```

But...

What about malloc, printf etc?

You need a standard library, e.g sourceware.org/newlib

Newlib prerequisites

- Implement low level system calls e.g. **sbrk** used by malloc
- Define location of heap memory in the linker script. Needed by sbrk.