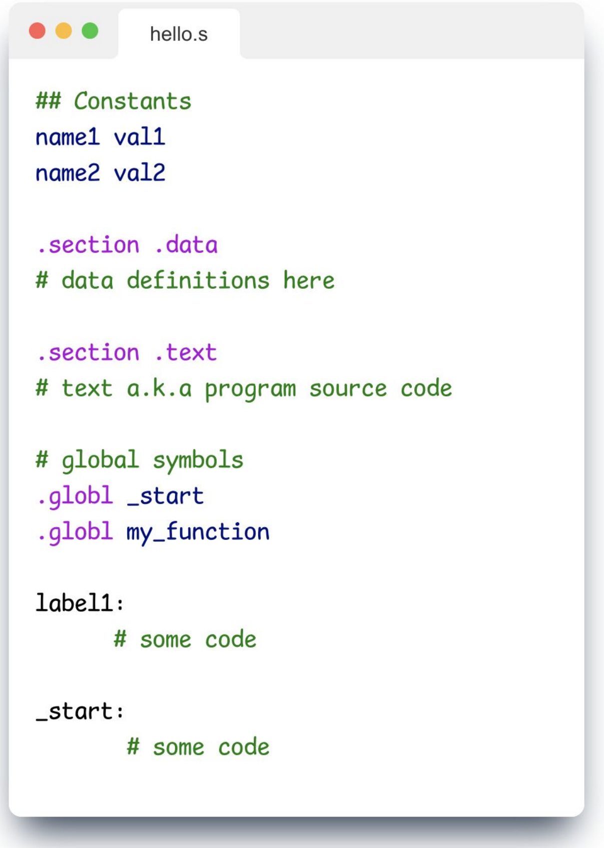
Computer Architecture

RISC-V Assembly Intro

Assembly program layout



```
## Constants
name1 val1
name2 val2

.section .data
# data definitions here

.section .text
# text a.k.a program source code

# global symbols
.globl _start
.globl my_function

label1:
      # some code

_start:
       # some code
```

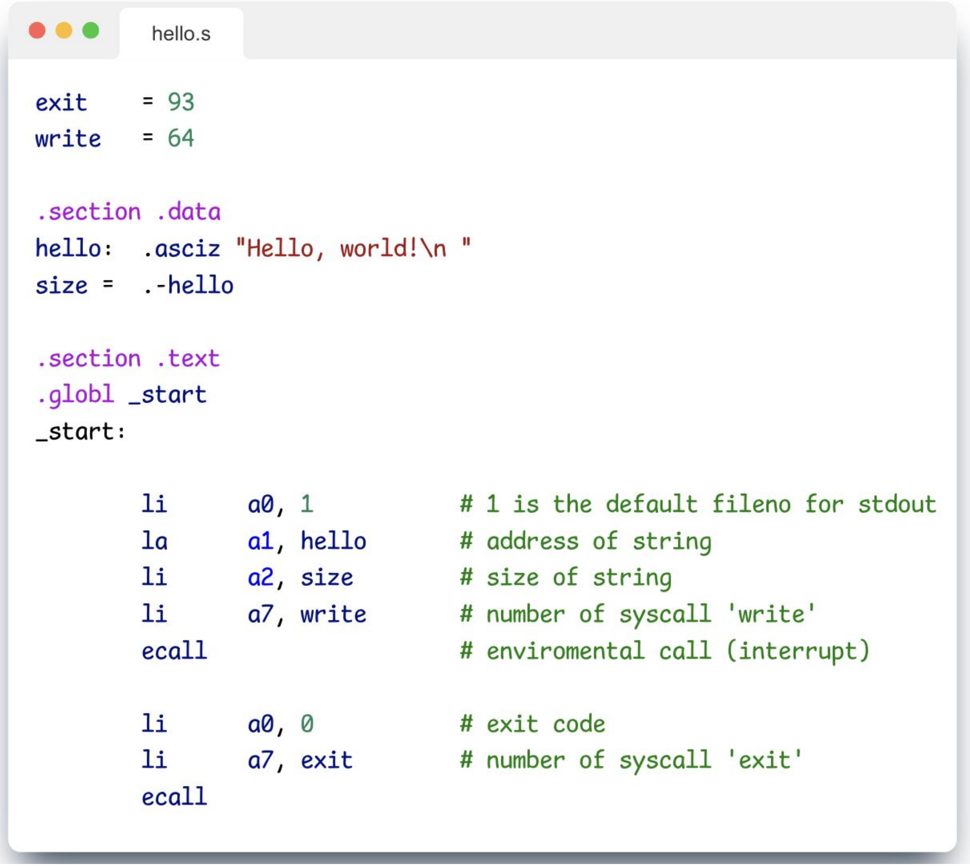
Values will be used by assembler in compilation time

Data segment, will go to .data section in ELF

Code segment, will go to .text section in ELF

Functions and variables names are just labels for memory addresses. We need to explicitly declare (.globl) if we need them outside

Hello world - no libc



```
exit    = 93
write   = 64

.section .data
hello:  .asciz "Hello, world!\n "
size =  .-hello

.section .text
.globl _start
_start:

        li      a0, 1           # 1 is the default fileno for stdout
        la      a1, hello       # address of string
        li      a2, size        # size of string
        li      a7, write       # number of syscall 'write'
        ecall                   # enviromental call (interrupt)

        li      a0, 0           # exit code
        li      a7, exit        # number of syscall 'exit'
        ecall
```

syscall numbers (yes we still in linux)

strings and numbers

linker will look for _start

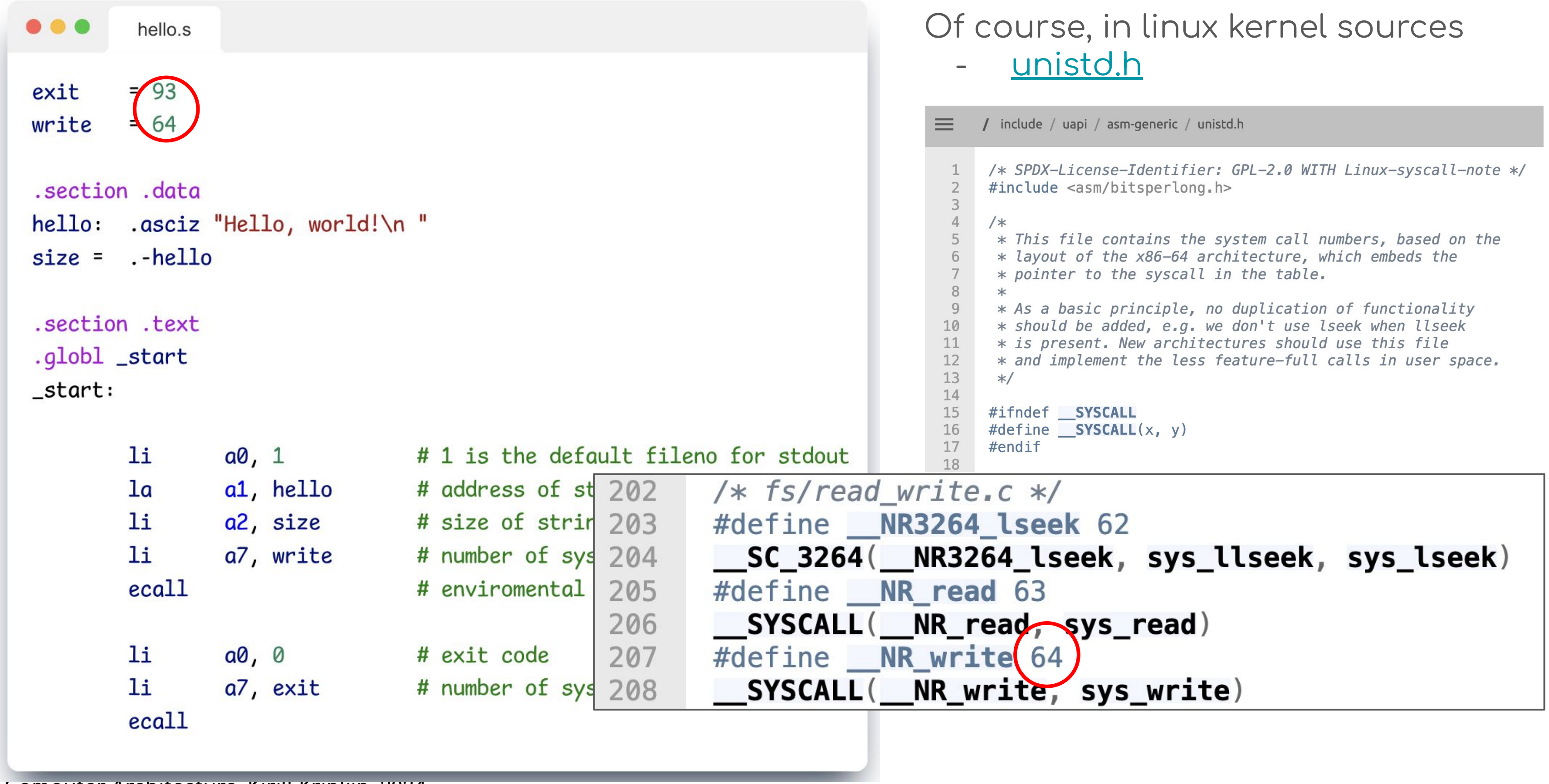
call write

call exit

Hello world - no libc - where to find syscalls?

Of course, in linux kernel sources

- unistd.h



hello.s

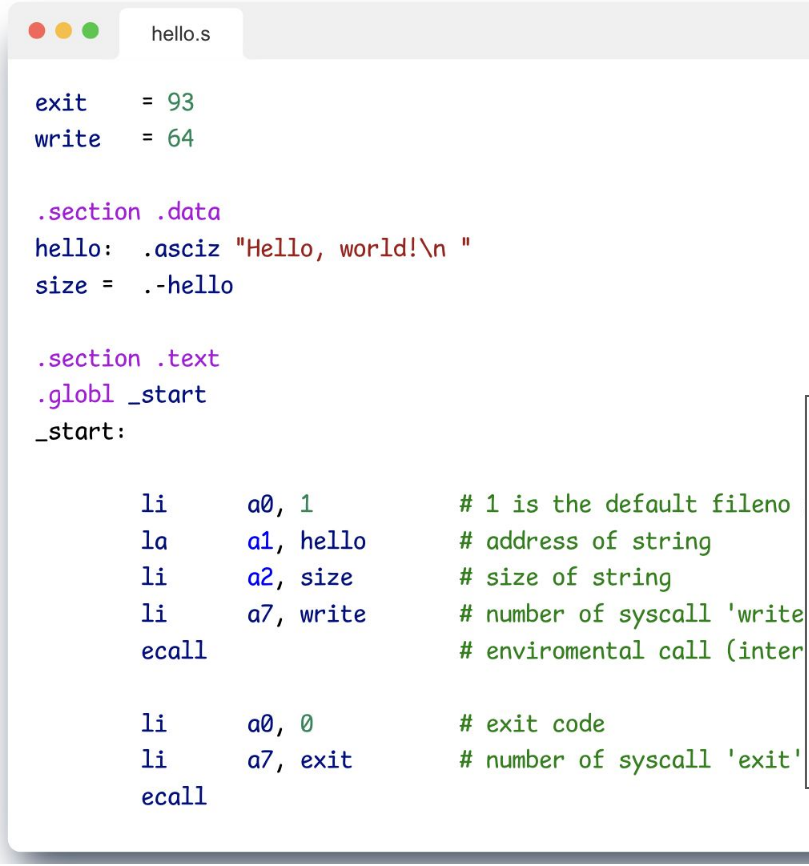
```
exit    = 93
write   = 64

.section .data
hello:  .asciz "Hello, world!\n "
size =  .-hello

.section .text
.globl _start
_start:

        li      a0, 1           # 1 is the default fileno for stdout
        la      a1, hello       # address of st
        li      a2, size        # size of strin
        li      a7, write       # number of sys
        ecall                   # enviromental

        li      a0, 0           # exit code
        li      a7, exit        # number of sys
        ecall
```

/ include / uapi / asm-generic / unistd.h

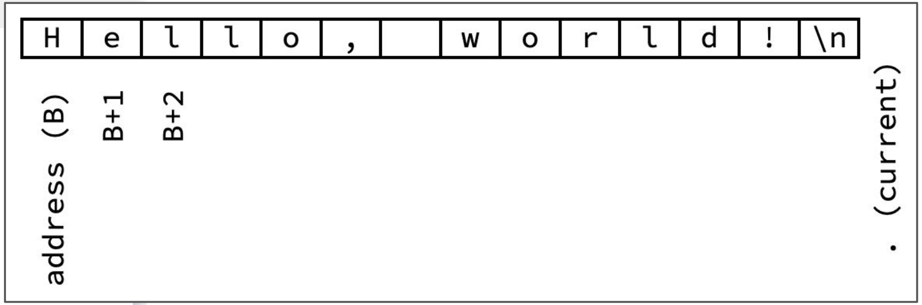
```
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
#include <asm/bitsperlong.h>

/*
 * This file contains the system call numbers, based on the
 * layout of the x86-64 architecture, which embeds the
 * pointer to the syscall in the table.
 *
 * As a basic principle, no duplication of functionality
 * should be added, e.g. we don't use lseek when llseek
 * is present. New architectures should use this file
 * and implement the less feature-full calls in user space.
 */

#ifndef __SYSCALL
#define __SYSCALL(x, y)
#endif

```

```
/* fs/read_write.c */
#define __NR3264_lseek 62
__SC_3264(__NR3264_lseek, sys_llseek, sys_lseek)
#define __NR_read 63
__SYSCALL(__NR_read, sys_read)
#define __NR_write 64
__SYSCALL(__NR_write, sys_write)
```

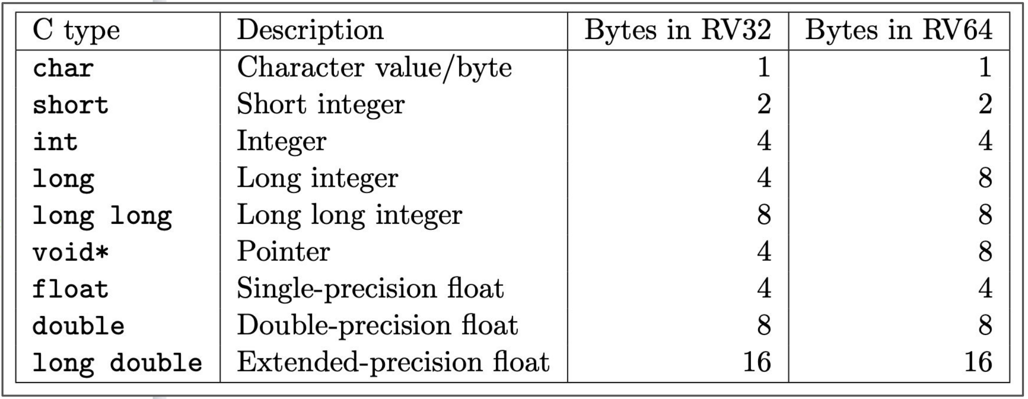
Hello world - no libc - data in memory?



```
exit    = 93
write   = 64

.section .data
hello:  .asciz "Hello, world!\n "
size =  .-hello

.section .text
.globl _start
_start:

        li      a0, 1           # 1 is the default fileno
        la      a1, hello       # address of string
        li      a2, size        # size of string
        li      a7, write       # number of syscall 'write
        ecall                   # enviromental call (inter

        li      a0, 0           # exit code
        li      a7, exit        # number of syscall 'exit'
        ecall
```

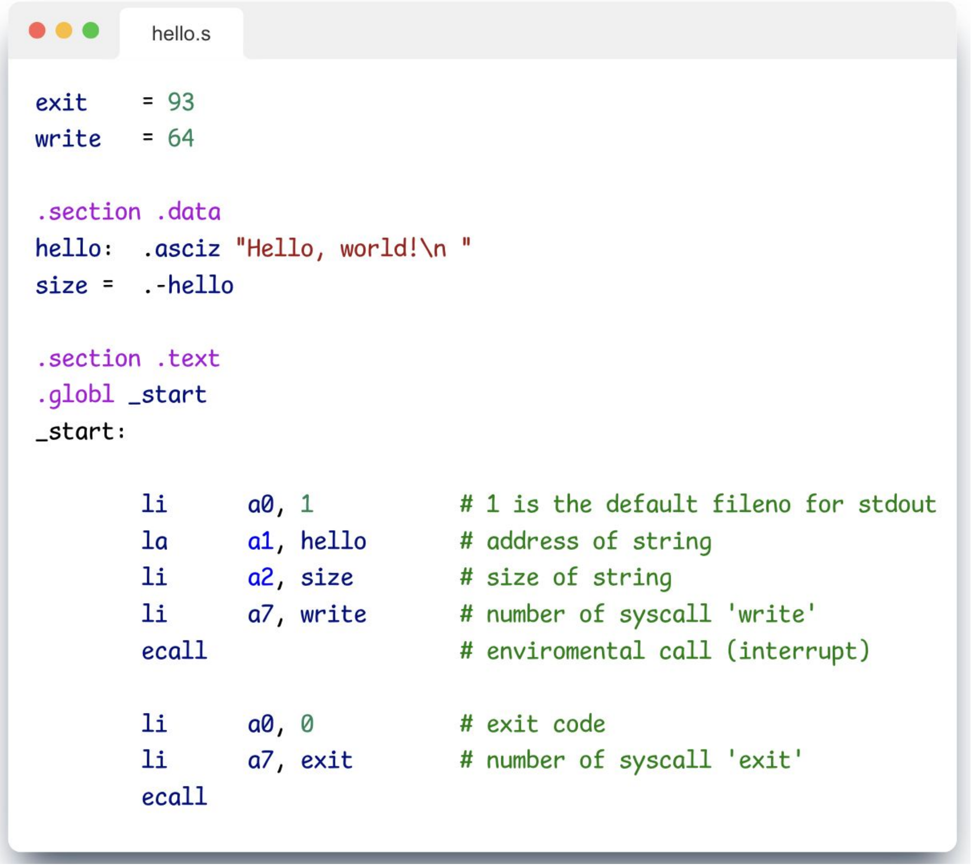


H	e	l	l	o	,		w	o	r	l	d	!	\n



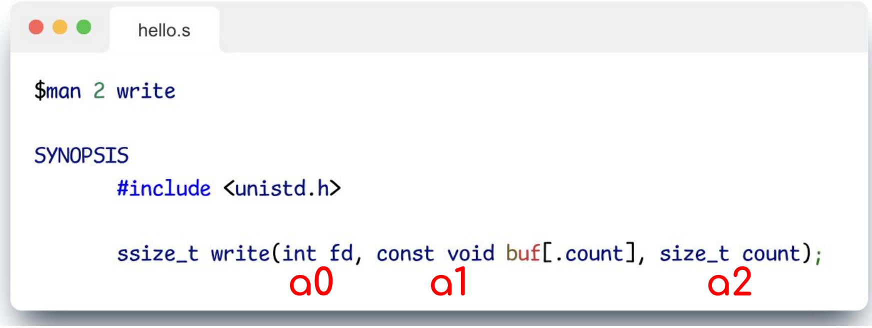
C type	Description	Bytes in RV32	Bytes in RV64
char	Character value/byte	1	1
short	Short integer	2	2
int	Integer	4	4
long	Long integer	4	8
long long	Long long integer	8	8
void*	Pointer	4	8
float	Single-precision float	4	4
double	Double-precision float	8	8
long double	Extended-precision float	16	16

Hello world - no libc - calling convention



```
exit    = 93
write   = 64

.section .data
hello:  .asciz "Hello, world!\n "
size =  .-hello

.section .text
.globl _start
_start:

        li      a0, 1           # 1 is the default fileno for stdout
        la      a1, hello       # address of string
        li      a2, size        # size of string
        li      a7, write       # number of syscall 'write'
        ecall                   # enviromental call (interrupt)

        li      a0, 0           # exit code
        li      a7, exit        # number of syscall 'exit'
        ecall
```

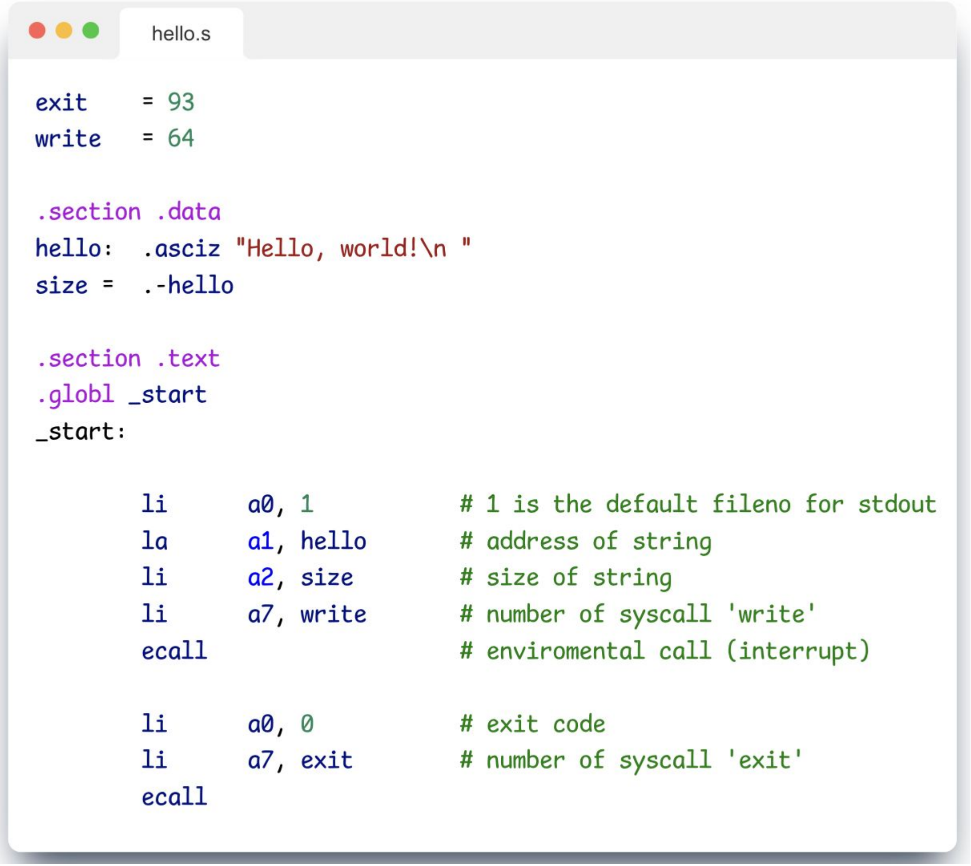
The RISC-V calling convention passes arguments in registers **when possible**. Up to eight integer registers, a0–a7 are used for this purpose.



```
$man 2 write

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void buf[.count], size_t count);
```

a7 used for syscall number

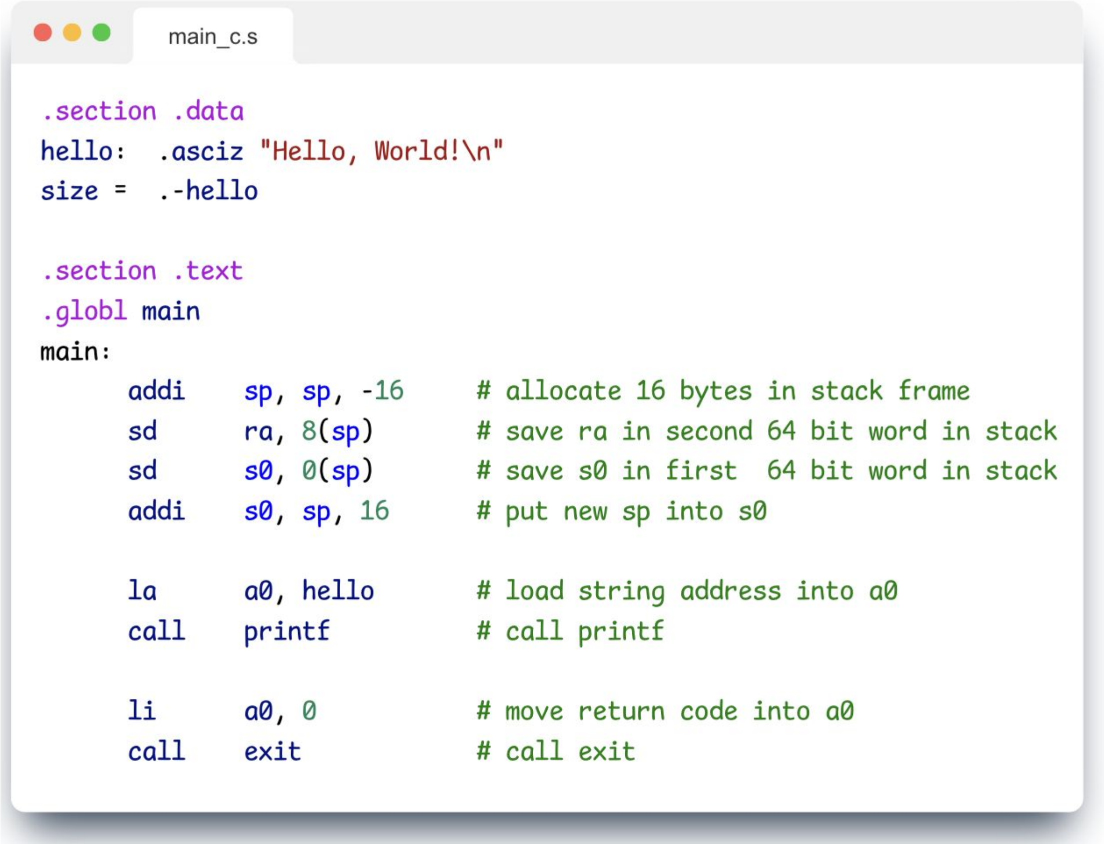
Hello world - no libc - registers



```
exit    = 93
write   = 64

.section .data
hello:  .asciz "Hello, world!\n "
size =  .-hello

.section .text
.globl _start
_start:

        li      a0, 1           # 1 is the default fileno for stdout
        la      a1, hello       # address of string
        li      a2, size        # size of string
        li      a7, write       # number of syscall 'write'
        ecall                   # enviromental call (interrupt)

        li      a0, 0           # exit code
        li      a7, exit        # number of syscall 'exit'
        ecall
```

- s0-s11 -> Saved registers (must be preserved across function calls)
- t0-t6 -> Temporary registers (can be overwritten by called functions)
- a0-a7 -> Argument/return registers

Hello world - libc



```
.section .data
hello:  .asciz "Hello, World!\n"
size =  .-hello

.section .text
.globl main
main:
      addi    sp, sp, -16     # allocate 16 bytes in stack frame
      sd      ra, 8(sp)       # save ra in second 64 bit word in stack
      sd      s0, 0(sp)       # save s0 in first  64 bit word in stack
      addi    s0, sp, 16      # put new sp into s0

      la      a0, hello       # load string address into a0
      call    printf          # call printf

      li      a0, 0           # move return code into a0
      call    exit            # call exit
```

same

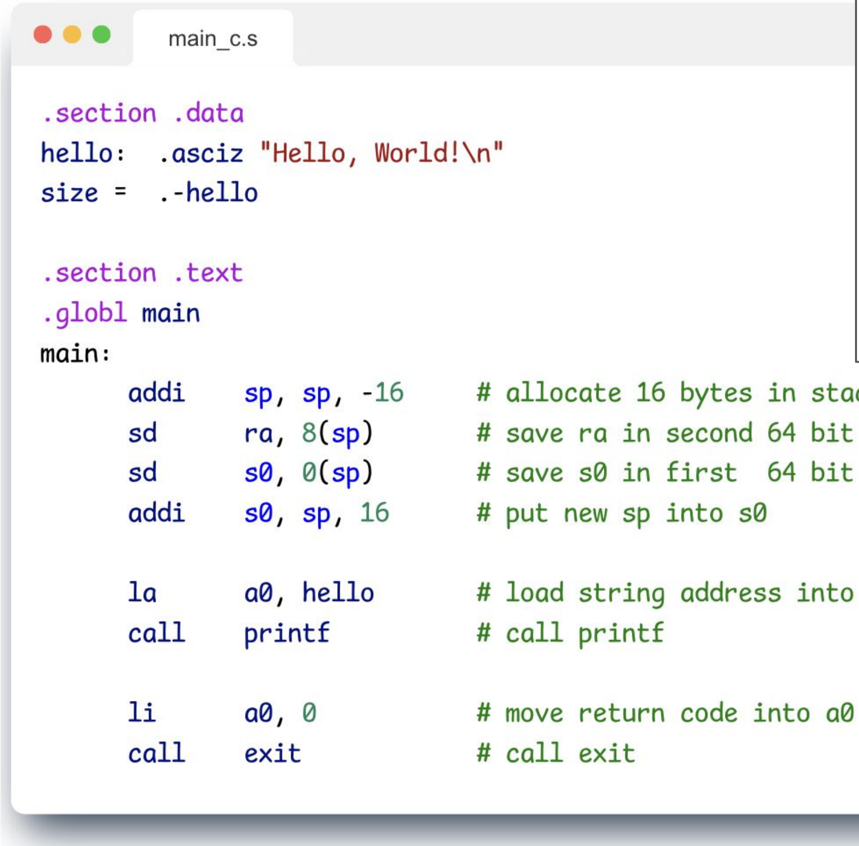
linker will look for main

stack frame preparation
(function prefix)

printf call

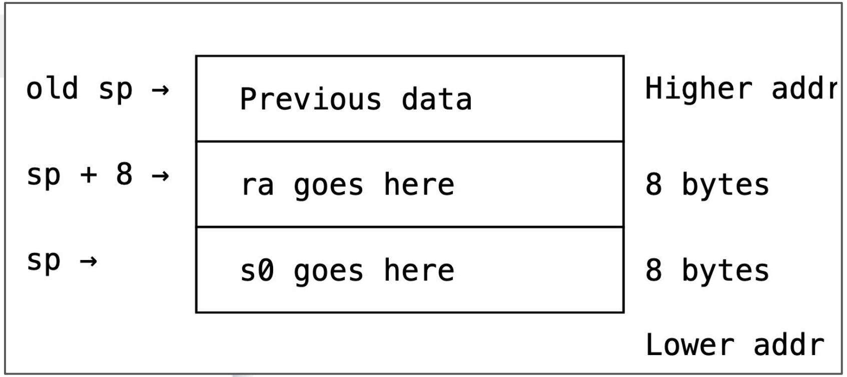
exit call

Hello world - libc - stack frame



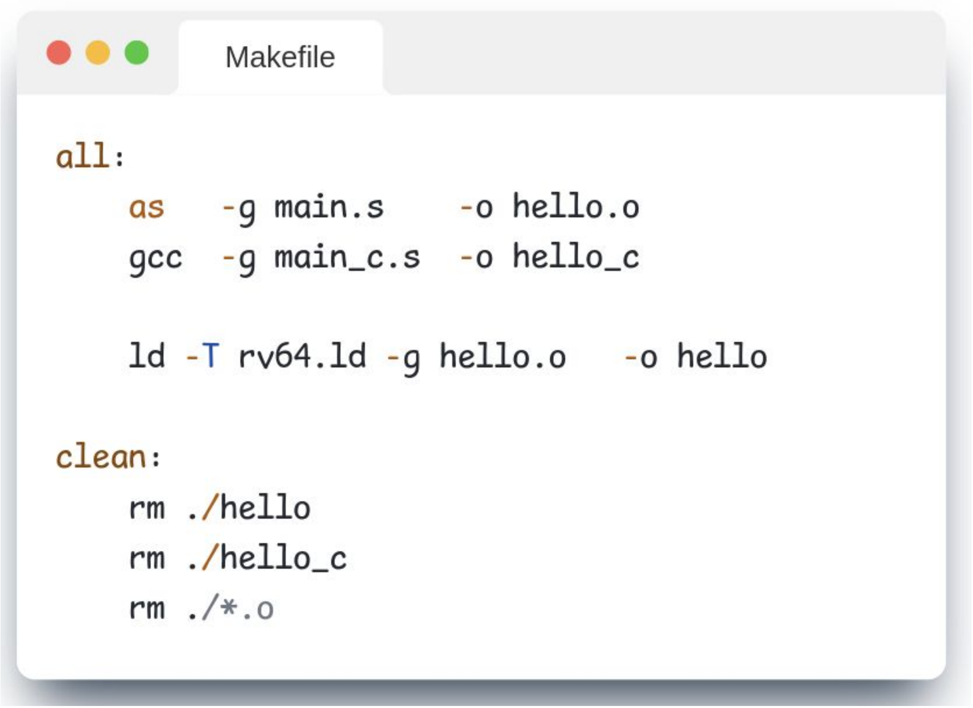
main_c.s

```
.section .data
hello:  .asciz "Hello, World!\n"
size =  .-hello

.section .text
.globl main
main:
        addi    sp, sp, -16     # allocate 16 bytes in stack frame
        sd      ra, 8(sp)       # save ra in second 64 bit word in stack
        sd      s0, 0(sp)       # save s0 in first  64 bit word in stack
        addi    s0, sp, 16      # put new sp into s0

        la      a0, hello       # load string address into a0
        call    printf          # call printf

        li      a0, 0           # move return code into a0
        call    exit            # call exit
```



old sp →	Previous data	Higher addr
sp + 8 →	ra goes here	8 bytes
sp →	s0 goes here	8 bytes
		Lower addr

Makefile



```
all:
    as   -g main.s    -o hello.o
    gcc  -g main_c.s  -o hello_c

    ld -T rv64.ld -g hello.o   -o hello

clean:
    rm ./hello
    rm ./hello_c
    rm ./*.o
```

Running RISC-V 64 VM

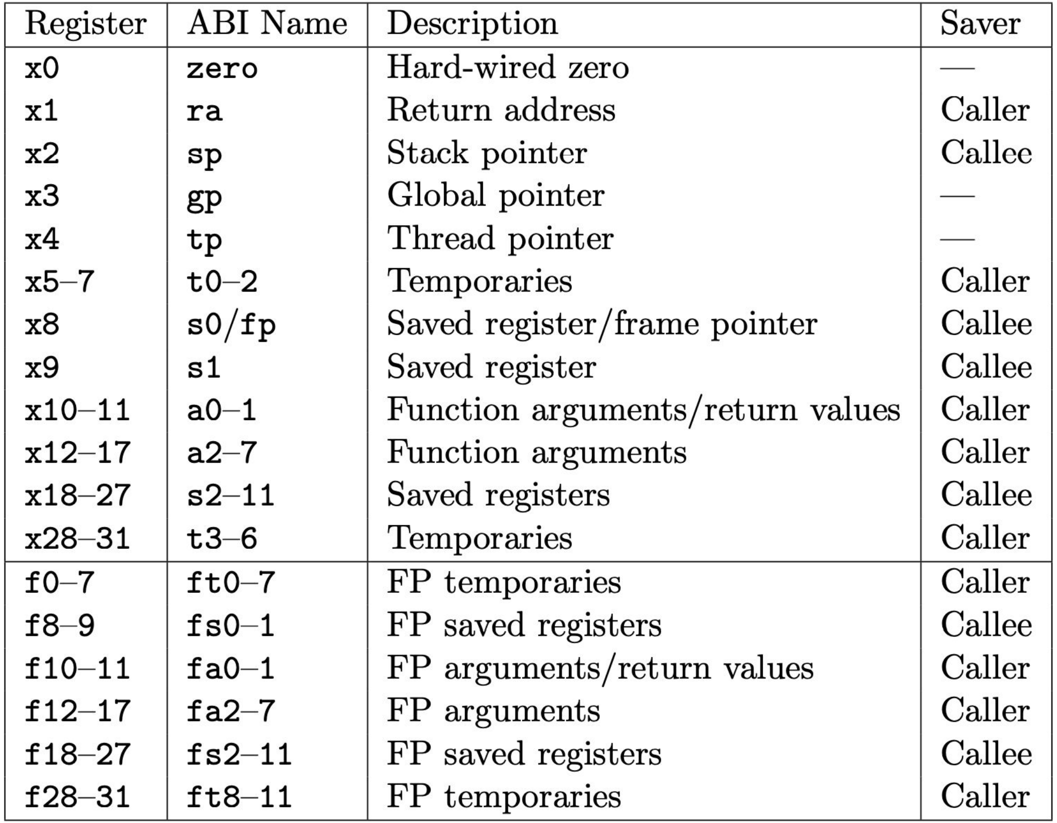
```
$docker run --name rv64 -p 2222:2222 krinkin/rv64vm
   … it takes ~5 min for the first start
$ssh -p 2222 root@localhost
   pwd = rv64
```

Manuals and readings

- GDB tutorial
- RISC-V Instructions Card
- RV32/64 C Calling convention

backup

Register convention



Register	ABI Name	Description	Saver
`x0`	`zero`	Hard-wired zero	—
`x1`	`ra`	Return address	Caller
`x2`	`sp`	Stack pointer	Callee
`x3`	`gp`	Global pointer	—
`x4`	`tp`	Thread pointer	—
`x5–7`	`t0–2`	Temporaries	Caller
`x8`	`s0/fp`	Saved register/frame pointer	Callee
`x9`	`s1`	Saved register	Callee
`x10–11`	`a0–1`	Function arguments/return values	Caller
`x12–17`	`a2–7`	Function arguments	Caller
`x18–27`	`s2–11`	Saved registers	Callee
`x28–31`	`t3–6`	Temporaries	Caller
`f0–7`	`ft0–7`	FP temporaries	Caller
`f8–9`	`fs0–1`	FP saved registers	Callee
`f10–11`	`fa0–1`	FP arguments/return values	Caller
`f12–17`	`fa2–7`	FP arguments	Caller
`f18–27`	`fs2–11`	FP saved registers	Callee
`f28–31`	`ft8–11`	FP temporaries	Caller