

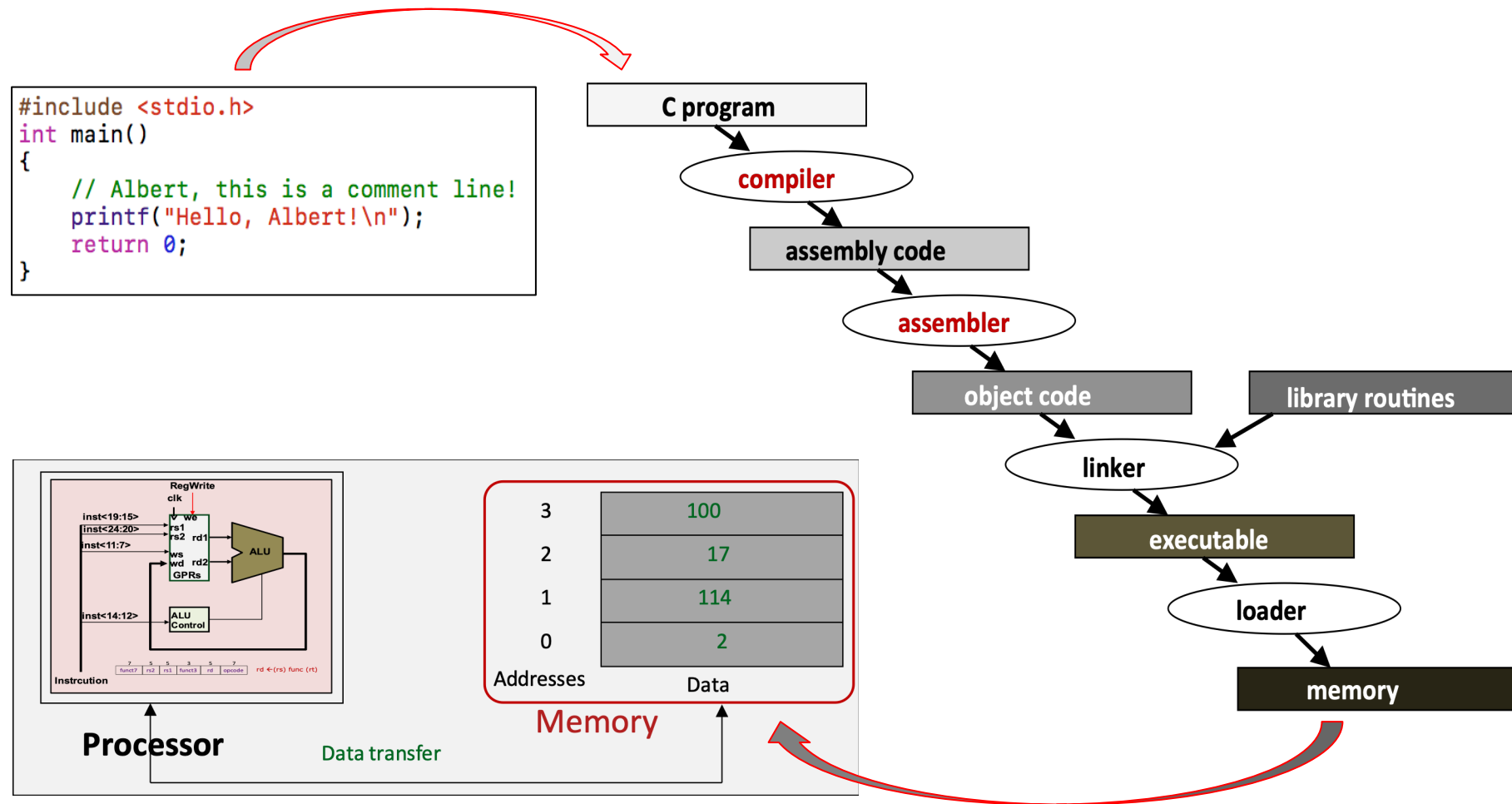
CSE 420

Fundamentals of Computer Architecture

Assembly Language Programming

Prof. Michel A. Kinsy,
Instructors Mishel Paul & Nges Njungle

The Big Picture

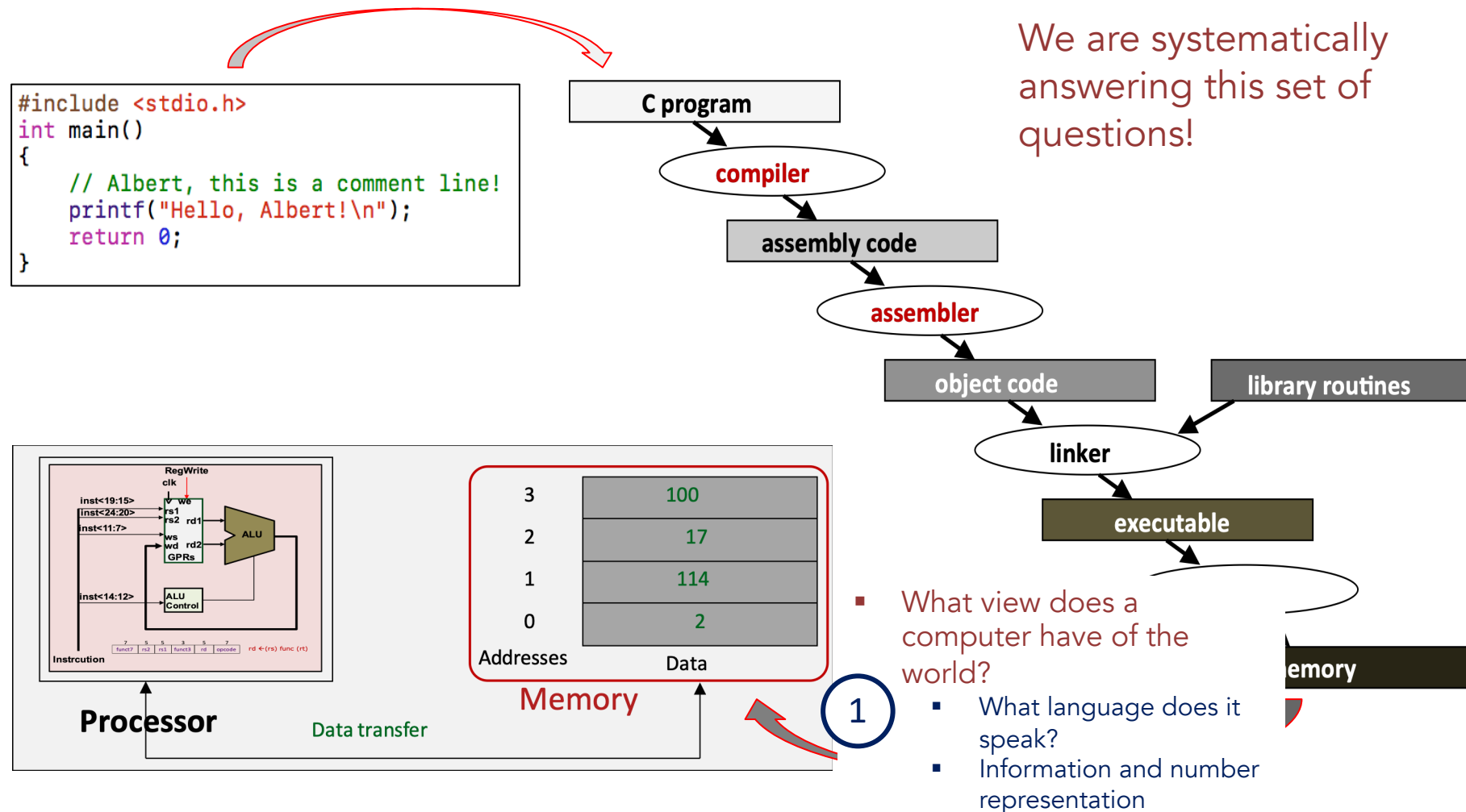


We are systematically answering this set of questions!

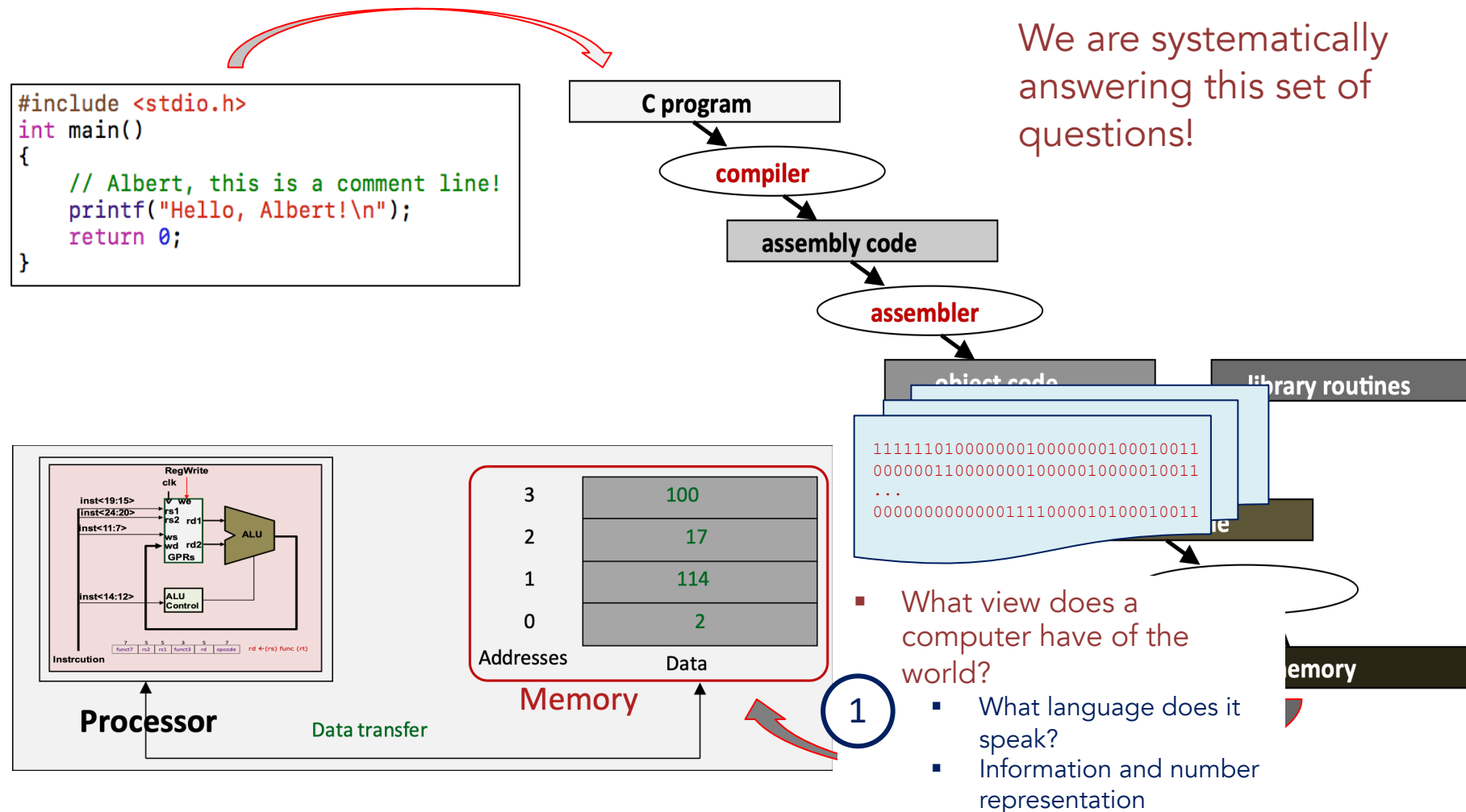
memory



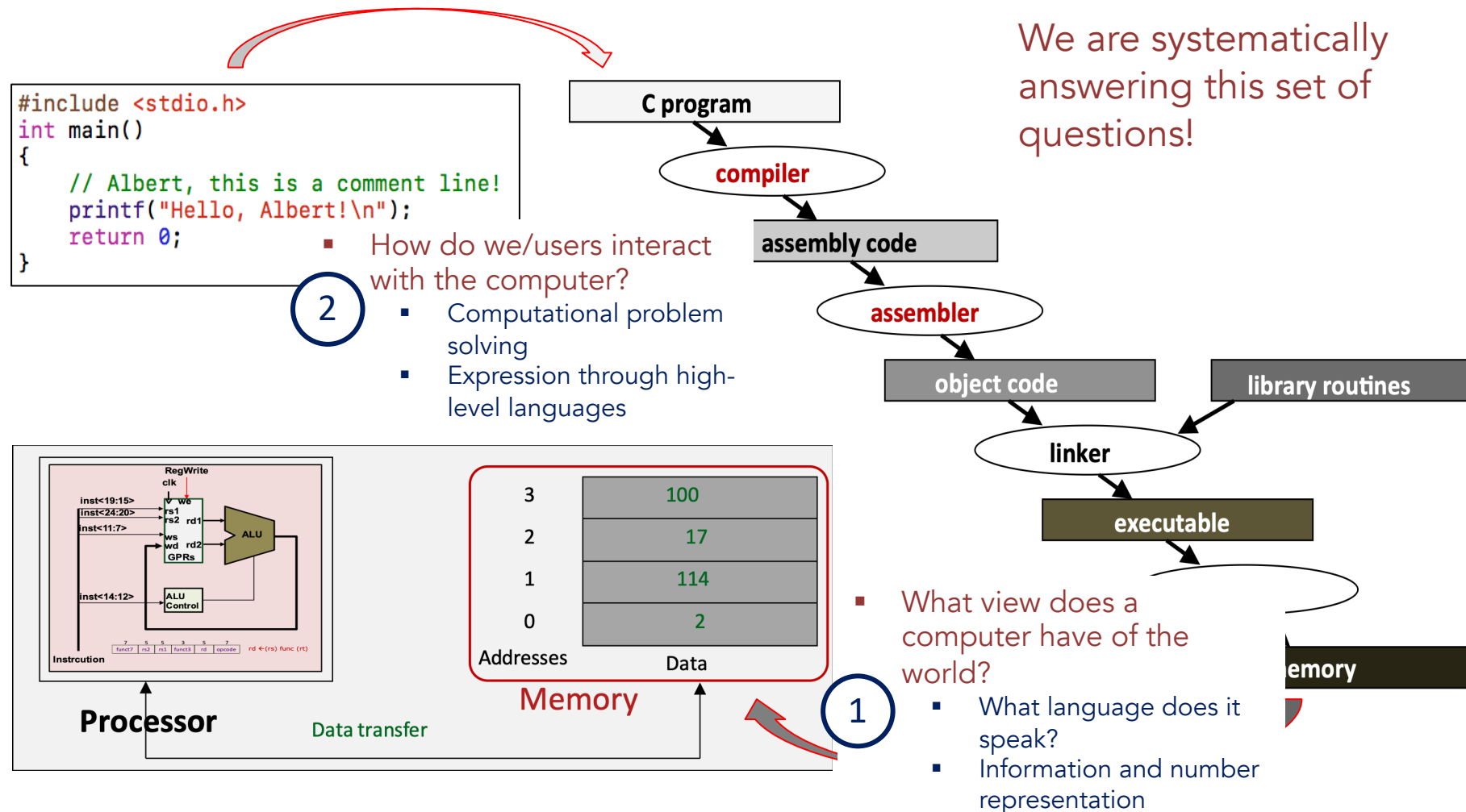
The Big Picture



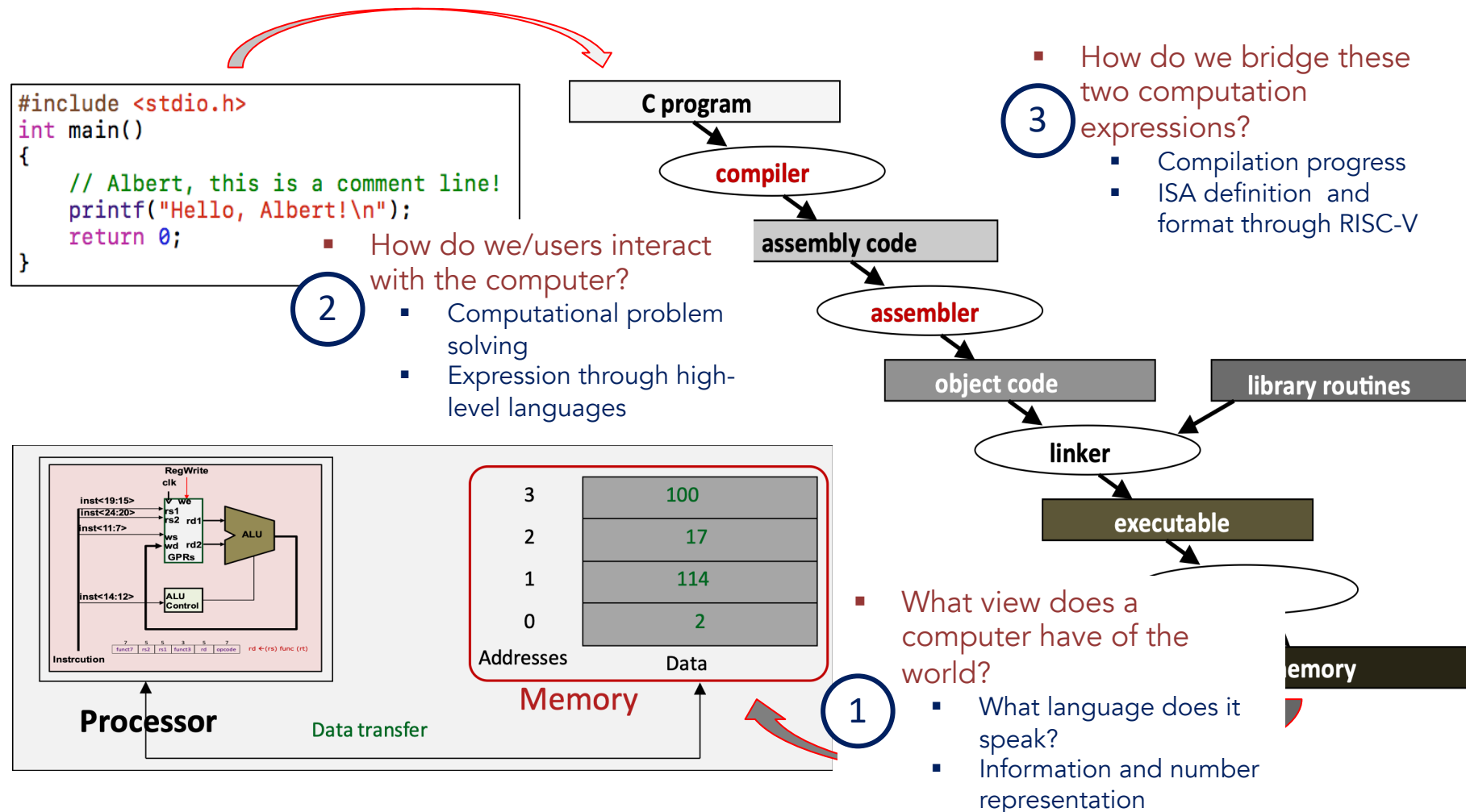
The Big Picture



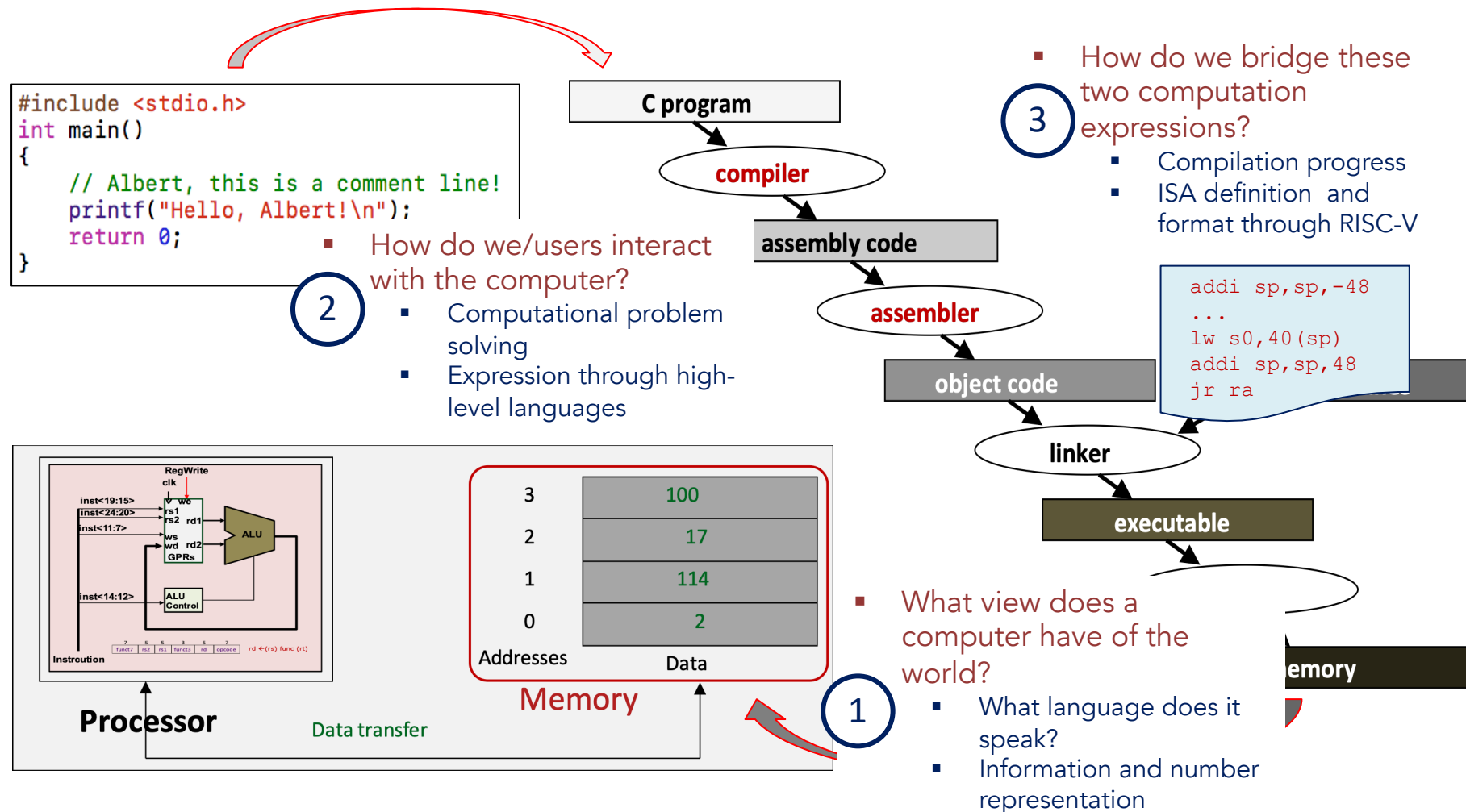
The Big Picture



The Big Picture



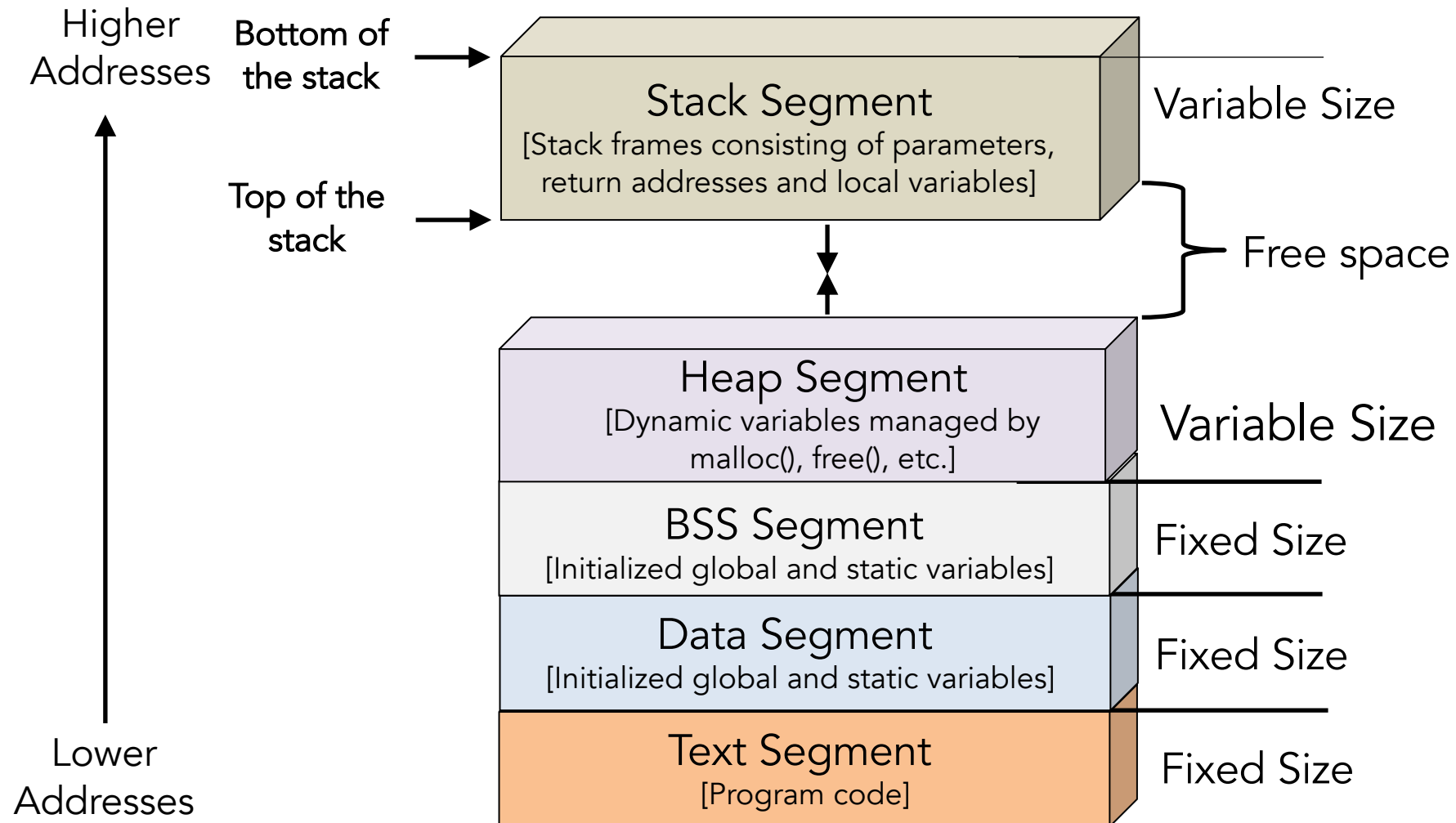
The Big Picture



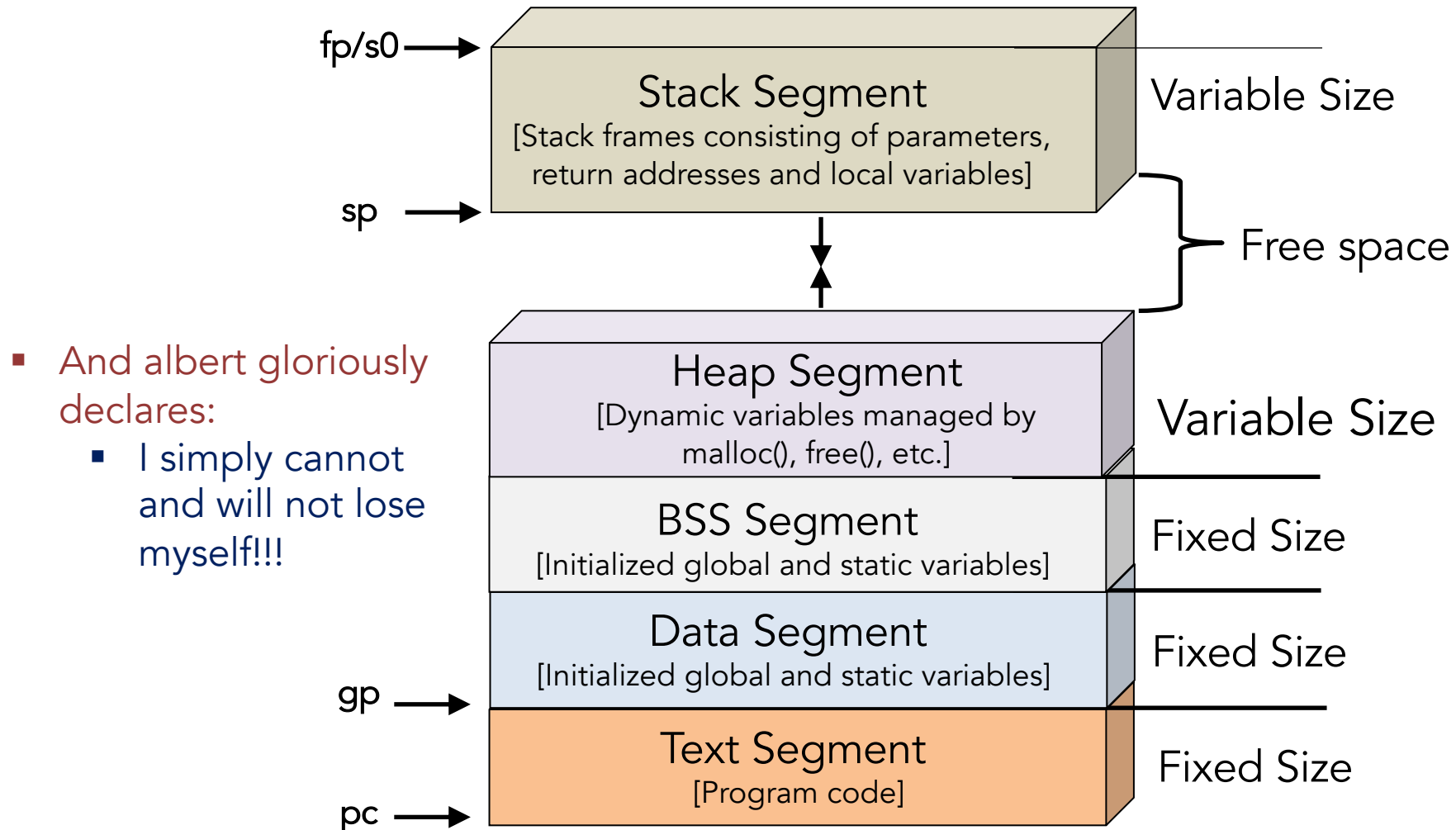
RISC-V ISA Registers

Name	Reg Number	Usage	Preserved across call?
zero	x0	The constant value 0	Yes
ra	x1	Return address	Yes
sp	x2	Stack pointer	Yes
gp	x3	Global pointer	Yes
tp	x4	Thread pointer	Yes
t0 – t2	x5 – x7	Temporary registers	No
s0/fp	x8	Frame pointer	Yes
s1	x9	Saved register	Yes
a0 - a1	x10 – x11	Function arguments/Return values	No
a2 – a7	x12 – x17	Function arguments	No

Program memory management



Program memory management



RISC-V Assembly

```
int    val1  = 5;
int    val2;
int    val3  = 2;
int    val4[3] = {7,8,9};

char   string1[] = "abc";
char   string2[3] = {'a','b','c'};

char   array1[5];
int    array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

```
val1:
    .word 5
val2:
    .zero 4
val3:
    .word 2
val4:
    .word 7
    .word 8
    .word 9
string1:
    .string "abc"
string2:
    .byte 97
    .byte 98
    .byte 99
array1:
    .zero 5
array2:
    .zero 8
array3:
    .zero 6
```

RISC-V Assembly

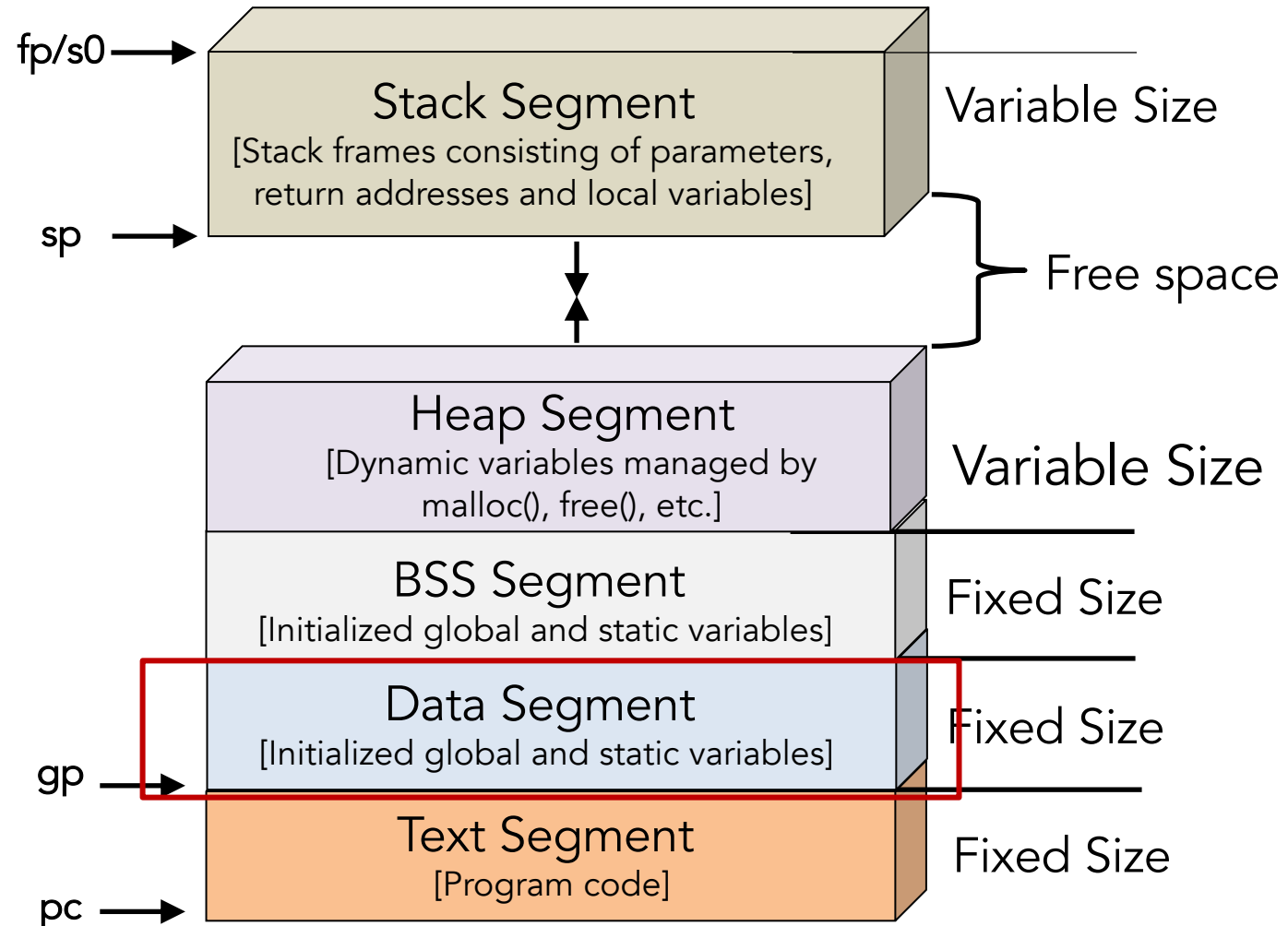
```

val1:
    .word 5
val2:
    .zero 4
val3:
    .word 2
val4:
    .word 7
    .word 8
    .word 9
string1:
    .string "abc"
string2:
    .byte 97
    .byte 98
    .byte 99
array1:
    .zero 5
array2:
    .zero 8
array3:
    .zero 6
    
```

Address	Value
0	5
1	0
2	0
3	0
4	?
5	?
6	?
7	?
8	2
9	0
A	0
B	0

Address	Value
C	7
D	0
E	0
F	0
10	8
11	0
12	0
13	0
14	9
15	0
16	0
17	0

Program memory management



RISC-V Assembly

```
val1:
    .word 5
val2:
    .zero 4
val3:
    .word 2
val4:
    .word 7
    .word 8
    .word 9
string1:
    .string "abc"
string2:
    .byte 97
    .byte 98
    .byte 99
array1:
    .zero 5
array2:
    .zero 8
array3:
    .zero 6
```

Address	Value
0	5
1	0
2	0
3	0
4	?
5	?
6	?
7	?
8	2
9	0
A	0
B	0

Address	Value
C	7
D	0
E	0
F	0
10	8
11	0
12	0
13	0
14	9
15	0
16	0
17	0

RISC-V Assembly

```
val1:
    .word 5
val2:
    .zero 4
val3:
    .word 2
val4:
    .word 7
    .word 8
    .word 9
string1:
    .string "abc"
string2:
    .byte 97
    .byte 98
    .byte 99
array1:
    .zero 5
array2:
    .zero 8
```

Program code addresses vs. hardware addresses

Address	Value
8000	5
8001	0
8002	0
8003	0
8004	?
8005	?
8006	?
8007	?
8008	2
8009	0
800A	0
800B	0

Address	Value
800C	7
800D	0
800E	0
800F	0
8010	8
8011	0
8012	0
8013	0
8014	9
8015	0
8016	0
8017	0

RISC-V Assembly

```
val1:
    .word 5
val2:
    .zero 4
val3:
    .word 2
val4:
    .word 7
    .word 8
    .word 9
string1:
    .string "abc"
string2:
    .byte 97
    .byte 98
    .byte 99
array1:
    .zero 5
array2:
    .zero 8
```

Program code addresses vs. hardware addresses

Address	Value
8018	0x61
8019	0x62
801A	0x63
801B	0
801C	0x61
801D	0x62
801E	0x63
801F	?
8020	?
8021	?
8022	?
8023	?

RISC-V Assembly – Let's Go All In

```
int    val1    = 5;
int    val2;
int    val3    = 2;
int    val4[3] = {7,8,9};

char   string1[] = "abc";
char   string2[3] = {'a','b','c'};

char   array1[5];
int    array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

RISC-V Assembly – Let's Go All In

```
int    val1    = 5;
int    val2;
int    val3    = 2;
int    val4[3] = {7,8,9};

char   string1[] = "abc";
char   string2[3] = {'a','b','c'};

char   array1[5];
int    array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

Note that: num1, num2, num3 and num4 are a0, a1, a2, and a3, respectively.

```
foo:
add t0,a0,zero    # move num1 into t0
add t1,a1,zero    # move num2 into t1
add t2,a2,zero    # move num3 into t2

add t2,t2,a3      # t2 = num1 + num2

add t0,t0,t1      # t0 = num3 + num4

sub t1,t2,t0
# result = (num1 + num2) - (num3 + num4);

add a0,t1,zero
# result returned in a0
jr ra
```

RISC-V Assembly – Let's Go All In

```
int    val1    = 5;
int    val2;
int    val3    = 2;
int    val4[3] = {7,8,9};

char   string1[] = "abc";
char   string2[3] = {'a','b','c'};

char   array1[5];
int    array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

Silly Albert! Who are you kidding!

Note that: num1, num2, num3 and num4 are a0, a1, a2, and a3, respectively.

```
foo:
add t0,a0,zero    # move num1 into t0
add t1,a1,zero    # move num2 into t1
add t2,a2,zero    # move num3 into t2

add t2,t2,a3      # t2 = num1 + num2

add t0,t0,t1      # t0 = num3 + num4

sub t1,t2,t0
# result = (num1 + num2) - (num3 + num4);

add a0,t1,zero
# result returned in a0
jr ra
```

RISC-V Assembly – Let's Go All In

```
int  val1  = 5;
int  val2;
int  val3  = 2;
int  val4[3] = {7,8,9};

char string1[] = "abc";
char string2[3] = {'a','b','c'};

char array1[5];
int  array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

```
        .globl  val1
        .section .sdata,"aw",@progbits
        .align  2
        .type   val1, @object
        .size   val1, 4

val1:
        .word   5

        .globl  val2
        .section .sbss,"aw",@nobits
        .align  2
        .type   val2, @object
        .size   val2, 4

val2:
        .zero   4

        .globl  val3
        .section .sdata
        .align  2
        .type   val3, @object
        .size   val3, 4

val3:
        .word   2
```

RISC-V Assembly – Let's Go All In

```
int  val1  = 5;
int  val2;
int  val3  = 2;
int  val4[3] = {7,8,9};

char string1[] = "abc";
char string2[3] = {'a','b','c'};

char array1[5];
int  array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

```
val1:
    .globl  val1
    .section .sdata,"aw",@progbits
    .align  2
    .type   val1, @object
    .size   val1, 4

    .word   5

val2:
    .globl  val2
    .section .sbss,"aw",@nobits
    .align  2
    .type   val2, @object
    .size   val2, 4

    .zero   4

val3:
    .globl  val3
    .section .sdata
    .align  2
    .type   val3, @object
    .size   val3, 4

    .word   2
```

RISC-V Assembly – Let's Go All In

```
int    val1    = 5;
int    val2;
int    val3    = 2;
int    val4[3] = {7,8,9};

char   string1[] = "abc";
char   string2[3] = {'a','b','c'};

char   array1[5];
int    array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

```
.globl    val4
.data
.align    2
.type     val4, @object
.size     val4, 12

val4:
.word     7
.word     8
.word     9

.globl    string1
.section   .sdata
.align    2
.type     string1, @object
.size     string1, 4

string1:
.string   "abc"

.globl    string2
.align    2
.type     string2, @object
.size     string2, 3

string2:
.byte     97
.byte     98
.byte     99
```

RISC-V Assembly – Let's Go All In

```
int    val1    = 5;
int    val2;
int    val3    = 2;
int    val4[3] = {7,8,9};

char   string1[] = "abc";
char   string2[3] = {'a','b','c'};

char   array1[5];
int    array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

```
.globl  val4
.data
.align  2
.type   val4, @object
.size   val4, 12

val4:
.word   7
.word   8
.word   9

.globl  string1
.section .sdata
.align  2
.type   string1, @object
.size   string1, 4
string1:
.string "abc"

.globl  string2
.align  2
.type   string2, @object
.size   string2, 3
string2:
.byte   97
.byte   98
.byte   99
```

RISC-V Assembly – Let's Go All In

```
int    val1    = 5;
int    val2;
int    val3    = 2;
int    val4[3] = {7,8,9};

char   string1[] = "abc";
char   string2[3] = {'a','b','c'};

char   array1[5];
int    array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

```
.globl  array1
.section      .sbss
.align      2
.type       array1, @object
.size       array1, 5
array1:
.zero       5

.globl  array2
.section      .sbss
.align      2
.type       array2, @object
.size       array2, 8
array2:
.zero       8

.globl  array3
.section      .sbss
.align      2
.type       array3, @object
.size       array3, 6
array3:
.zero       6
```

RISC-V Assembly – Let's Go All In

```
int  val1  = 5;
int  val2;
int  val3  = 2;
int  val4[3] = {7,8,9};

char  string1[]  = "abc";
char  string2[3] = {'a','b','c'};

char  array1[5];
int   array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

```
        .text
        .align 1
        .globl foo
        .type  foo, @function

foo:
        addi    sp,sp,-48
        sw      s0,44(sp)
        addi    s0,sp,48
        sw      a0,-36(s0)
        sw      a1,-40(s0)
        sw      a2,-44(s0)
        sw      a3,-48(s0)
        lw      a4,-36(s0)
        lw      a5,-40(s0)
        add     a4,a4,a5
        lw      a3,-44(s0)
        lw      a5,-48(s0)
        add     a5,a3,a5
        sub     a5,a4,a5
        sw      a5,-20(s0)
        lw      a5,-20(s0)
        mv      a0,a5
        lw      s0,44(sp)
        addi    sp,sp,48
        jr      ra
```

RISC-V Assembly – Let's Go All In

```
int  val1  = 5;
int  val2;
int  val3  = 2;
int  val4[3] = {7,8,9};

char string1[] = "abc";
char string2[3] = {'a','b'}

char array1[5];
int  array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

▪ Stack pointer adjustment

```
.text
.align 1
.globl foo
.type foo, @function

foo:
    addi sp, sp, -48
    sw    s0, 44(sp)
    addi  s0, sp, 48
    sw    a0, -36(s0)
    sw    a1, -40(s0)
    sw    a2, -44(s0)
    sw    a3, -48(s0)
    lw    a4, -36(s0)
    lw    a5, -40(s0)
    add   a4, a4, a5
    lw    a3, -44(s0)
    lw    a5, -48(s0)
    add   a5, a3, a5
    sub   a5, a4, a5
    sw    a5, -20(s0)
    lw    a5, -20(s0)
    mv    a0, a5
    lw    s0, 44(sp)
    addi  sp, sp, 48
    jr    ra
```

1

RISC-V Assembly – Let's Go All In

```
int  val1  = 5;
int  val2;
int  val3  = 2;
int  val4[3] = {7,8,9};

char string1[] = "abc";
char string2[3] = {'a','b'};

char array1[5];
int  array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

- Save old frame pointer
 - Caller's frame pointer

```
.text
.align 1
.globl foo
.type foo, @function

foo:
    addi    sp,sp,-48
    sw      s0,44(sp)
    addi    s0,sp,48
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    sw      a2,-44(s0)
    sw      a3,-48(s0)
    lw      a4,-36(s0)
    lw      a5,-40(s0)
    add     a4,a4,a5
    lw      a3,-44(s0)
    lw      a5,-48(s0)
    add     a5,a3,a5
    sub     a5,a4,a5
    sw      a5,-20(s0)
    lw      a5,-20(s0)
    mv      a0,a5
    lw      s0,44(sp)
    addi    sp,sp,48
    jr      ra
```

2

RISC-V Assembly – Let's Go All In

```
int val1 = 5;
int val2;
int val3 = 2;
int val4[3] = {7,8,9};
```

```
char string1[10];
char string2[10];
```

```
char array1[5];
int array2[2];
short int array3[3];
```

```
int foo (int num1, int num2, int num3, int num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4);
    return result;
}
```

```
void main(void) {
    array2[0] = foo(val1, val2, val3, val4[1]);
}
```

- The frame pointer (fp/s0) points to the start of the stack frame
 - It points to the base of the stack frame
 - Parameters passed in to the subroutine remain at constant places relative to the frame pointer
- It does not move for the duration of the subroutine call

```
.text
.align 1
.globl foo
.type foo, @function

foo:
    addi sp, sp, -48
    sw s0, 44(sp)
    addi s0, sp, 48
    sw a0, -36(s0)
    sw a1, -40(s0)
    sw a2, -44(s0)
    sw a3, -48(s0)
    lw a4, -36(s0)
    lw a5, -40(s0)
    add a4, a4, a5
    lw a3, -44(s0)
    lw a5, -48(s0)
    add a5, a3, a5
    sub a5, a4, a5
    sw a5, -20(s0)
    lw a5, -20(s0)
    mv a0, a5
    lw s0, 44(sp)
    addi sp, sp, 48
    jr ra
```

3

RISC-V Assembly – Let's Go All In

```
int  val1  = 5;
int  val2;
int  val3  = 2;
int  val4[3] = {7,8,9};

char string1[] = "abc";
char string2[3] = {'a','b','c'};

char array1[5];
int  array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

- Save the function arguments
 - In case, something happens in the middle of the function execution
 - Like divide by zero

```
.text
.align 1
.globl foo
.type foo, @function

foo:
    addi    sp,sp,-48
    sw      s0,44(sp)
    addi    s0,sp,48
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    sw      a2,-44(s0)
    sw      a3,-48(s0)
    lw      a4,-36(s0)
    lw      a5,-40(s0)
    add     a4,a4,a5
    lw      a3,-44(s0)
    lw      a5,-48(s0)
    add     a5,a3,a5
    sub     a5,a4,a5
    sw      a5,-20(s0)
    lw      a5,-20(s0)
    mv      a0,a5
    lw      s0,44(sp)
    addi    sp,sp,48
    jr      ra
```

4

RISC-V Assembly – Let's Go All In

```
int  val1  = 5;
int  val2;
int  val3  = 2;
int  val4[3] = {7,8,9};

char string1[] = "abc";
char string2[3] = {'a','b','c'};

char array1[5];
int  array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

■ Perform actual operations

```
.text
.align 1
.globl foo
.type foo, @function

foo:
    addi    sp,sp,-48
    sw      s0,44(sp)
    addi    s0,sp,48
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    sw      a2,-44(s0)
    sw      a3,-48(s0)
    lw      a4,-36(s0)
    lw      a5,-40(s0)
    add     a4,a4,a5
    lw      a3,-44(s0)
    lw      a5,-48(s0)
    add     a5,a3,a5
    sub     a5,a4,a5
    sw      a5,-20(s0)
    lw      a5,-20(s0)
    mv      a0,a5
    lw      s0,44(sp)
    addi    sp,sp,48
    jr      ra
```

5

RISC-V Assembly – Let's Go All In

```
int  val1  = 5;
int  val2;
int  val3  = 2;
int  val4[3] = {7,8,9};

char string1[] = "abc";
char string2[3] = {'a','b','c'};

char array1[5];
int  array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

▪ Return result

```
.text
.align 1
.globl foo
.type foo, @function

foo:
    addi    sp,sp,-48
    sw      s0,44(sp)
    addi    s0,sp,48
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    sw      a2,-44(s0)
    sw      a3,-48(s0)
    lw      a4,-36(s0)
    lw      a5,-40(s0)
    add     a4,a4,a5
    lw      a3,-44(s0)
    lw      a5,-48(s0)
    add     a5,a3,a5
    sub     a5,a4,a5
    sw      a5,-20(s0)
    lw      a5,-20(s0)
    mv      a0,a5
    lw      s0,44(sp)
    addi    sp,sp,48
    jr      ra
```

6

RISC-V Assembly – Let's Go All In

```
int  val1  = 5;
int  val2;
int  val3  = 2;
int  val4[3] = {7,8,9};

char string1[] = "abc";
char string2[3] = {'a','b','c'};

char array1[5];
int  array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

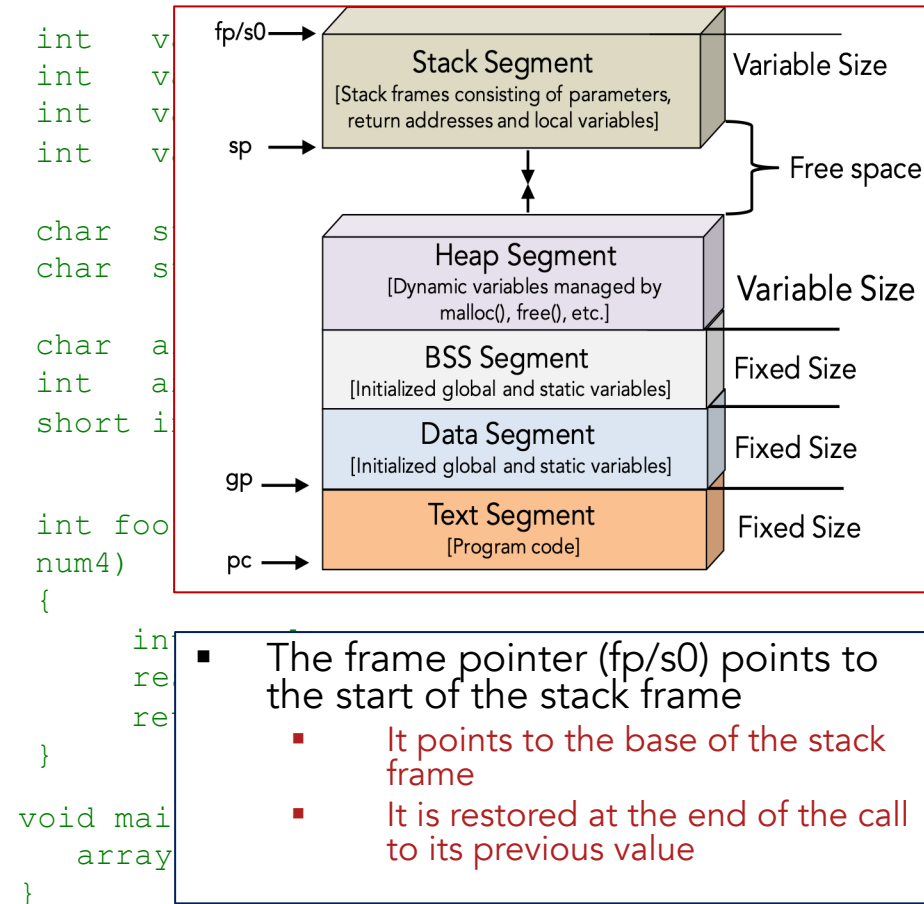
void main(void) {
    array2 [0] = foo(val1,
```

- Rollback things (frame pointer and stack pointer)

```
.text
.align 1
.globl foo
.type foo, @function

foo:
addi sp, sp, -48
sw s0, 44(sp)
addi s0, sp, 48
sw a0, -36(s0)
sw a1, -40(s0)
sw a2, -44(s0)
sw a3, -48(s0)
lw a4, -36(s0)
lw a5, -40(s0)
add a4, a4, a5
lw a3, -44(s0)
lw a5, -48(s0)
add a5, a3, a5
sub a5, a4, a5
sw a5, -20(s0)
lw a5, -20(s0)
mv a0, a5
lw s0, 44(sp)
addi sp, sp, 48
jr ra
```

RISC-V Assembly – Let's Go All In



```
.text
.align 1
.globl foo
.type foo, @function

foo:
    addi sp, sp, -48
    sw s0, 44(sp)
    addi s0, sp, 48
    sw a0, -36(s0)
    sw a1, -40(s0)
    sw a2, -44(s0)
    sw a3, -48(s0)
    lw a4, -36(s0)
    lw a5, -40(s0)
    add a4, a4, a5
    lw a3, -44(s0)
    lw a5, -48(s0)
    add a5, a3, a5
    sub a5, a4, a5
    sw a5, -20(s0)
    lw a5, -20(s0)
    mv a0, a5
    lw s0, 44(sp)
    addi sp, sp, 48
    jr ra
```

A red box highlights the `.text` label and the `lw s0, 44(sp)` and `addi sp, sp, 48` instructions. A blue box highlights the `addi sp, sp, -48`, `sw s0, 44(sp)`, and `addi s0, sp, 48` instructions. A blue circle with the number 7 is next to the `addi sp, sp, 48` instruction.

RISC-V Assembly – Let's Go All In

```
int    val1    = 5;
int    val2;
int    val3    = 2;
int    val4[3] = {7,8,9};

char   string1[] = "abc";
char   string2[3] = {'a','b','c'};

char   array1[5];
int    array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

```
.text
.align 1
.globl foo
.type   foo, @function

foo:
    addi    sp,sp,-48
    sw      s0,44(sp)
    addi    s0,sp,48
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    sw      a2,-44(s0)
    sw      a3,-48(s0)
    lw      a4,-36(s0)
    lw      a5,-40(s0)
    add     a4,a4,a5
    lw      a3,-44(s0)
    lw      a5,-48(s0)
    add     a5,a3,a5
    sub     a5,a4,a5
    sw      a5,-20(s0)
    lw      a5,-20(s0)
    mv      a0,a5
    lw      s0,44(sp)
    addi    sp,sp,48
    jr      ra
```

▪ Jump out and time to go Albert!

RISC-V Assembly – Let's Go All In

```
int    val1    = 5;
int    val2;
int    val3    = 2;
int    val4[3] = {7,8,9};

char   string1[] = "abc";
char   string2[3] = {'a','b','c'};

char   array1[5];
int    array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

```
.text
.align 1
.globl foo
.type   foo, @function

foo:
    addi    sp,sp,-48
    sw      s0,44(sp)
    addi    s0,sp,48
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    sw      a2,-44(s0)
    sw      a3,-48(s0)
    lw      a4,-36(s0)
    lw      a5,-40(s0)
    add     a4,a4,a5
    lw      a3,-44(s0)
    lw      a5,-48(s0)
    add     a5,a3,a5
    sub     a5,a4,a5
    sw      a5,-20(s0)
    lw      a5,-20(s0)
    mv      a0,a5
    lw      s0,44(sp)
    addi    sp,sp,48
    jr      ra
```

▪ Jump out and time to go Albert!

RISC-V Assembly – Let's Go All In

```
int    val1  = 5;
int    val2;
int    val3  = 2;
int    val4[3] = {7,8,9};

char   string1[] = "abc";
char   string2[3] = {'a','b','c'};

char   array1[5];
int    array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

```
        .zero    6
        .text
        .align   1
        .globl   main
        .type     main, @function

main:
        addi     sp,sp,-16
        sw       ra,12(sp)
        sw       s0,8(sp)
        addi     s0,sp,16
        lui      a5,%hi(val1)
        lw       a4,%lo(val1)(a5)
        lui      a5,%hi(val2)
        lw       a1,%lo(val2)(a5)
        lui      a5,%hi(val3)
        lw       a2,%lo(val3)(a5)
        lui      a5,%hi(val4)
        addi     a5,a5,%lo(val4)
        lw       a5,4(a5)
        mv       a3,a5
        mv       a0,a4
        call     foo
        mv       a4,a0
        lui      a5,%hi(array2)
        sw       a4,%lo(array2)(a5)
        li       a5,0
        mv       a0,a5
        lw       ra,12(sp)
        lw       s0,8(sp)
        addi     sp,sp,16
        jr       ra
```

RISC-V Assembly – Let's Go All In

```
int  val1  = 5;
int  val2;
int  val3  = 2;
int  val4[3] = {7,8,9};

char string1[] = "abc";
char string2[3] = {'a','b','c'};

char array1[5];
int  array2[2];
short int array3[3];

int foo (int num1, int num2, int num3, int
num4)
{
    int result;
    result = (num1 + num2) - (num3 + num4) ;
    return result;
}

void main(void) {
    array2 [0] = foo(val1, val2, val3, val4[1]);
}
```

```
.zero    6
.text
.align   1
.globl   main
.type    main, @function

main:
    addi   sp, sp, -16
    sw     ra, 12(sp)
    sw     s0, 8(sp)
    addi   s0, sp, 16
    lui    a5, %hi(val1)
    lw     a4, %lo(val1)(a5)
    lui    a5, %hi(val2)
    lw     a1, %lo(val2)(a5)
    lui    a5, %hi(val3)
    lw     a2, %lo(val3)(a5)
    lui    a5, %hi(val4)
    addi   a5, a5, %lo(val4)
    lw     a5, 4(a5)
    mv     a3, a5
    mv     a0, a4
    call   foo
    mv     a4, a0
    lui    a5, %hi(array2)
    sw     a4, %lo(array2)(a5)
    li     a5, 0
    mv     a0, a5
    lw     ra, 12(sp)
    lw     s0, 8(sp)
    addi   sp, sp, 16
    jr     ra
```

Pseudo Instructions

Useful Pseudo-instructions	Meaning	Actual Translation
<i>mv rd, rs</i>	Copy register rs to rd	<code>addi rd, rs, 0</code>
<i>li rd, immediate</i>	Load immediate into rd	<code>addi rd, \$0, immed</code>
<i>beqz rs, offset</i> <i>bnez rs, offset</i> <i>bgtz rs, offset</i>	Branch if = zero Branch if != zero Branch if > zero	<code>beq rs, x0, offset</code> <code>bne rs, x0, offset</code> <code>blt x0, rs, offset</code>
<i>j offset</i> <i>jal offset</i> <i>jr rs</i> <i>jalr rs</i>	Jump Jump and link Jump register Jump and link register	<code>jal x0, offset</code> <code>jal x1, offset</code> <code>jalr x0, rs, 0</code> <code>jalr x1, rs, 0</code>
<i>call offset</i>	Call far-away subroutine	<code>auipc x6, offset[31:12]</code> <code>jalr x1, x6, offset[11:0]</code>
<i>ret</i>	Return from subroutine	<code>jalr x0, x1, 0</code>
<i>nop</i>	Do nothing useful	<code>add x0, x0, x0</code>

Function Entry

```
prolog:
    addi    sp, sp, -16    # -16 (frame size) depends on the amount of space needed
    sw      ra, 12(sp)     # offset must be framesize-4
    sw      fp, 8(sp)      # store the caller's fp
    sw      gp, 4(sp)      # store caller's gp
    mv      fp, sp         # fp is now the new function's frame base pointer

# ...
# The actual functional body of the function
# ...

epilog:
    lw      gp, 4(sp)
    lw      fp, 8(sp)
    addi    sp, sp, 16     # reverse the effect of first instruction
    jr      ra
```

Assembly Programs

- If (condition) then do this

```
if ( i == j )  
    i++ ;  
j++;
```

Assuming t1 stores i and t2 stores j:

```
        bne  t1, t2, NEXT    # branch if !( i == j )  
        addi t1, t1, 1       # i++  
NEXT:    addi t2, t2, 1       # j++
```

Assembly Programs

- If (condition) then do this else do that

```
if ( i == j && i == k )  
  
    i++ ;  
else  
    j++;  
  
j = i + k ;
```

Assuming t0 stores i, t1 stores j, and t2 stores k:

```
    bne  t0, t1, ELSE    # cond1: branch if !( i == j )  
    bne  t0, t2, ELSE    # cond2: branch if !( i == k )  
    addi t0, t0, 1        # if-body: i++  
    j NEXT              # jump over else  
ELSE: addi t1, t1, 1      # else-body: j++  
NEXT: add  t1, t0, t2     # j = i + k
```

Assembly Programs

- While (condition) then keep doing this

```
while ( i < j )  
{  
    k = k * 2;  
    i++;  
}
```

Assuming t0 stores i, t1 stores j, and t2 stores k:

```
Loop: bge  t0, t1, DONE # branch if ! ( i < j )  
      add  t2, t2, t2    # k = k * 2  
      addi t0, t0, 1     # i++  
      j    Loop         # jump back to top of loop  
DONE:
```

Assembly Programs

- For (these many times) do this

```
for (i = 0; i < j; i++)  
{  
    k = k * 2;  
}
```

Assuming t0 stores i, t1 stores j, and t2 stores k:

```
    addi t0, $0, 0        # i = 0  
Loop: bge t0, t1, DONE    # branch if ! ( i < j )  
    add t2, t2, t2        # k = k * 2  
    addi t0, t0, 1        # i++  
    j Loop                # jump back to top of loop  
DONE:
```

Assembly Programs

- For (these many times) do this

```
switch( i ) {  
    case 1:  
        i++ ;  
    break;  
    case 2:  
        i += 2 ;  
    break;  
    case 3:  
        i += 3 ;  
    break;  
}
```

Assuming t0 stores i and t1 stores temp:

```
        addi t1, zero, 1      # case 1: set temp to 1  
        bne  t0, t1, CASE2    # false: branch to case 2  
        j    CASE1_BODY      # true: branch to case 1 body  
CASE2:   addi t1, zero, 2      # case 2: set temp to 2  
        bne  t0, t1, CASE3    # false: branch to case 3  
        j    CASE2_BODY      # true: branch to case 2 body  
CASE3:   addi t1, zero, 3      # case 3: set temp to 3  
        bne  t0, t1, EXIT     # false: branch to exit  
        j    CASE3_BODY      # true: branch to case 3 body  
CASE1_BODY: addi t0, t0, 1      # case 1 body: i++  
        j    EXIT            # break  
CASE2_BODY: addi t0, t0, 2      # case 2 body: i += 2  
        j    EXIT            # break  
CASE3_BODY: addi t0, t0, 3      # case 3 body: i += 3  
        j    EXIT            # break  
EXIT:
```

Assembly Programs

- Short program

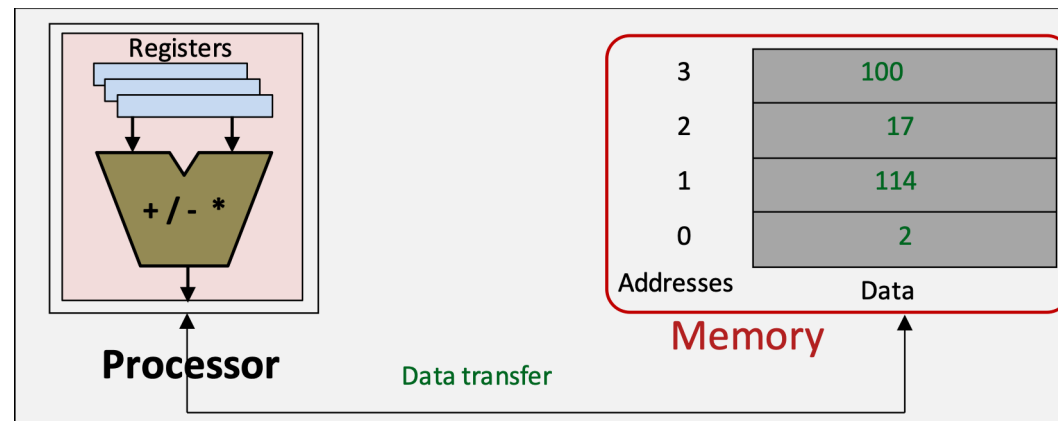
```
clear(int array[], int size)
{
    int i;
    for (i = 0; i < size; i += 1)
        array[i] = 0;
}
```

```
        mv    t0,zero        # i = 0

Loop:   sll   t1,t0,2         # t1 = i * 4
        add  t2,a0,t1        # t2 = &array[i]
        sw   zero, 0(t2)     # array[i] = 0
        addi t0,t0,1         # i = i + 1
        slt  a3,t0,a1        # a3 = (i < size)
        bne  a3,zero,Loop    # if (...)
                                # goto loop
```

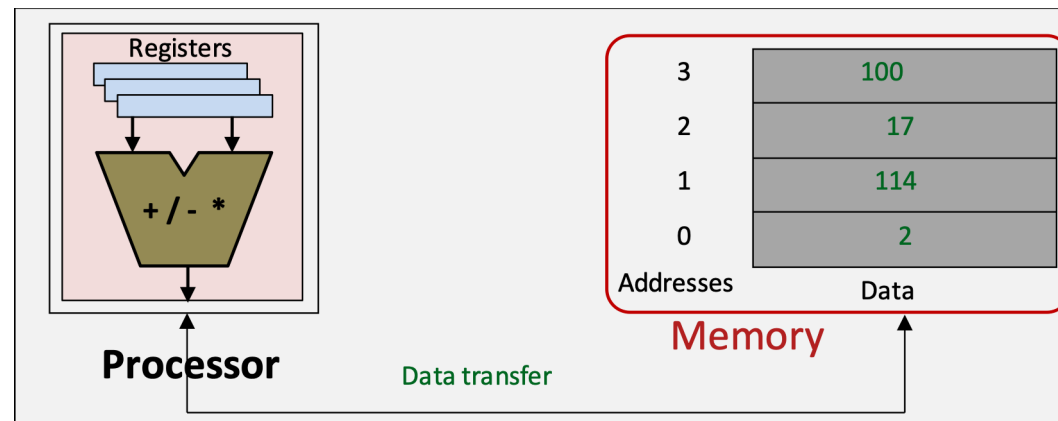
Data Addressing Modes

- RISC-V has four data addressing modes
 1. Register
 - Data is in a register
 2. Immediate
 - Data is specified in the instruction directly
 3. Displacement
 - Data is in memory with address calculated as $\text{Base} + \text{Offset}$
 4. PC-Relative
 - Data is in memory with address calculated as $\text{PC} + \text{Offset}$



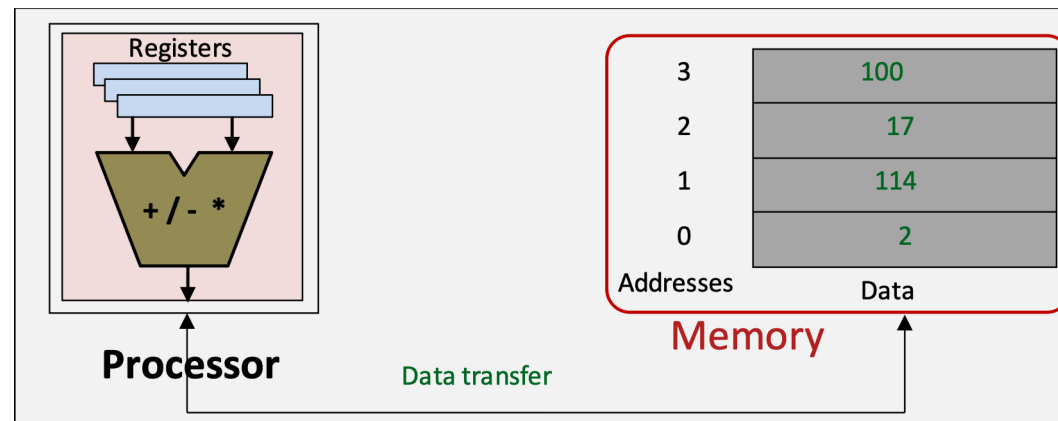
Data Addressing Modes

- RISC-V has four data addressing modes
 - Memory related addressing modes
 - Register indirect
 - `add x2, (x1)`
 - $x1 \leftarrow x2 + \text{Mem}[x1]$
 - Displacement
 - `add x2, 64(x1)`
 - $x1 \leftarrow x2 + \text{Mem}[64 + x1]$

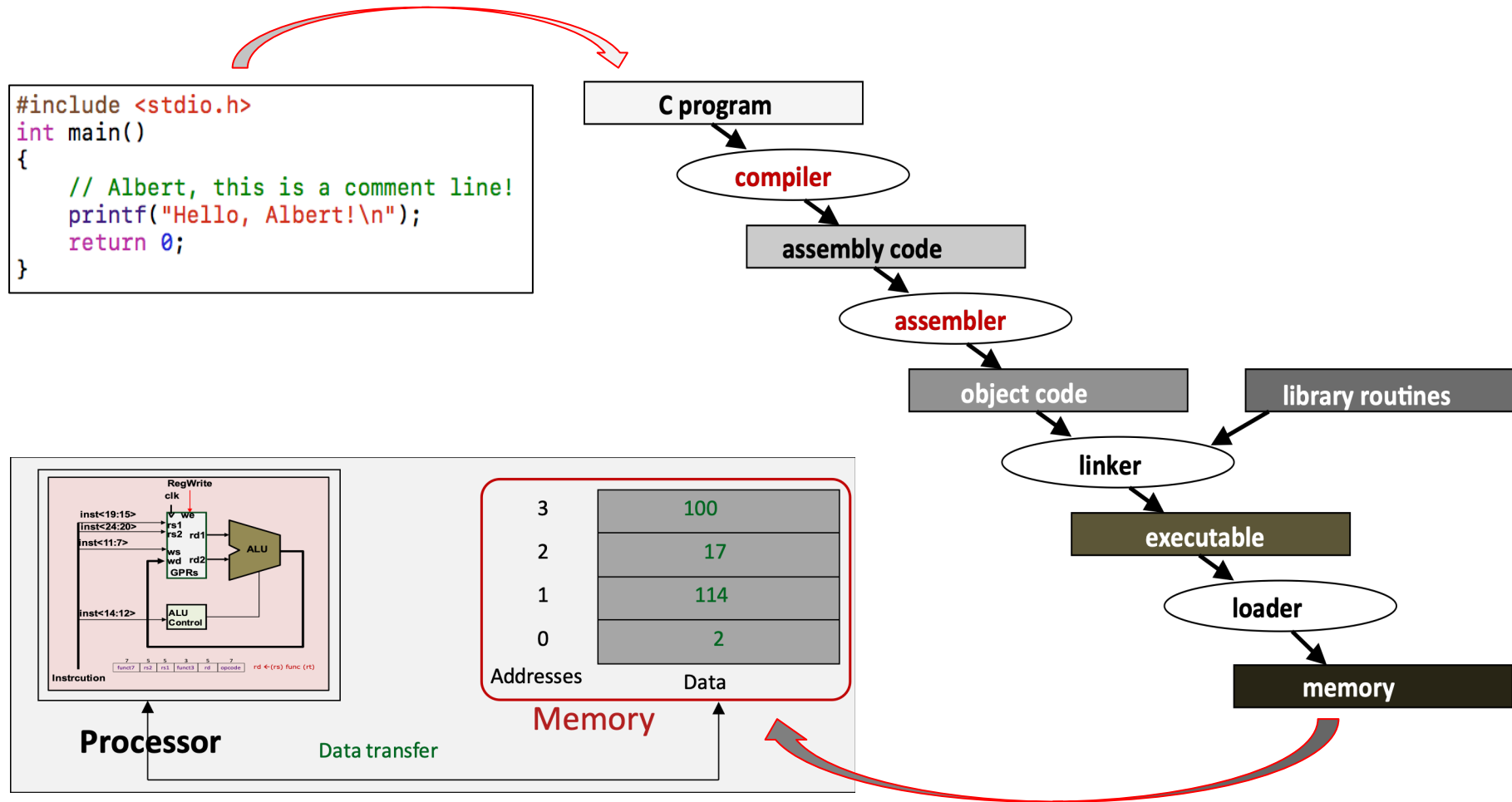


Data Addressing Modes

- Other memory address modes (non RISC-V)
 - Direct or absolute
 - `add x1,(1024)`
 - $x1 \leftarrow x1 + \text{Mem}[1024]$
 - Memory indirect
 - `add x1,@(x3)`
 - $x1 \leftarrow x1 + \text{Mem}[\text{Mem}[x3]]$



The Big Picture



Application Compiling Process

- High-level language program (in C)

```
void swap (int array[], int i) {  
    int temp;  
    temp      = array[i];  
    array[i]   = array[i+1];  
    array[i+1] = temp;  
}
```

one-to-many

- Assembly language program (for RISC-V)

```
swap:  
    addi sp,sp,-48  
    ...  
    mv a5,a1  
    ...  
    ld s0,40(sp)  
    addi sp,sp,48  
    jr ra
```

C compiler

Application Compiling Process

- Assembly language program (for RISC-V)

```
swap:
    addi sp, sp, -48
    ...
    mv a5, a1
    ...
    ld s0, 40(sp)
    addi sp, sp, 48
    jr ra
```

one-to-one

assembler

```
graph TD
    A[Assembly Code] -- "one-to-one" --> B((assembler))
    B --> C[Machine Code]
```

- Machine (object, binary) code (for RISC-V)

```
111111010000    00010    000 00010    0010011
000000110000    00010    000 01000    0010011
...
```

Application Compiling Process

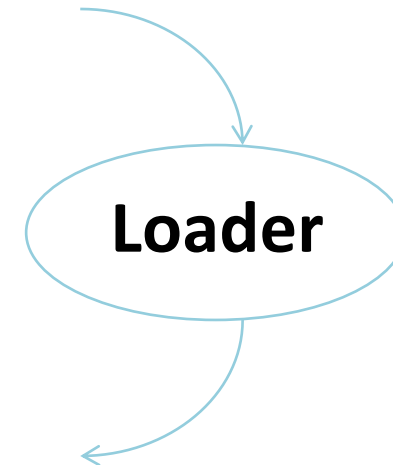
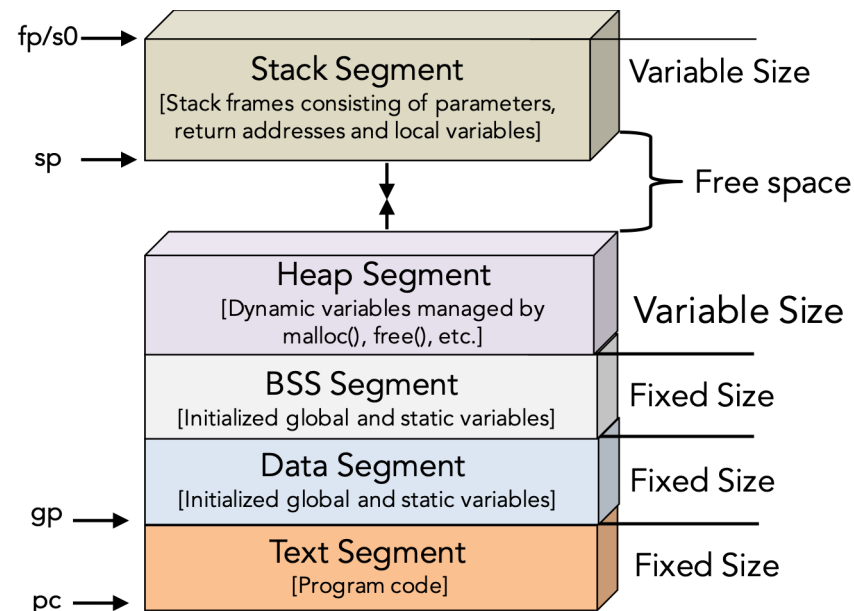
- Assembler (or compiler) translates program into machine instructions
- Provides information for building a complete program from the pieces
 - Header: described contents of object module
 - Text segment: translated instructions
 - Static data segment: data allocated for the life of the program
 - Relocation info: for contents that depend on absolute location of loaded program
 - Symbol table: global definitions and external refs
 - Debug info: for associating with source code

Application Compiling Process

- Machine (object, binary) code (for RISC-V)

```
111111010000 00010 000 00010 0010011
000000110000 00010 000 01000 0010011
```

- Memory organization



Application Compiling Process

- Load from image file on disk into memory
 1. Read header to determine segment sizes
 2. Create virtual address space
 3. Copy text and initialized data into memory
 4. Set up arguments on stack
 5. Initialize registers (including sp, fp, gp)
 6. Jump to startup routine
- Copies arguments to a0, ... and calls main

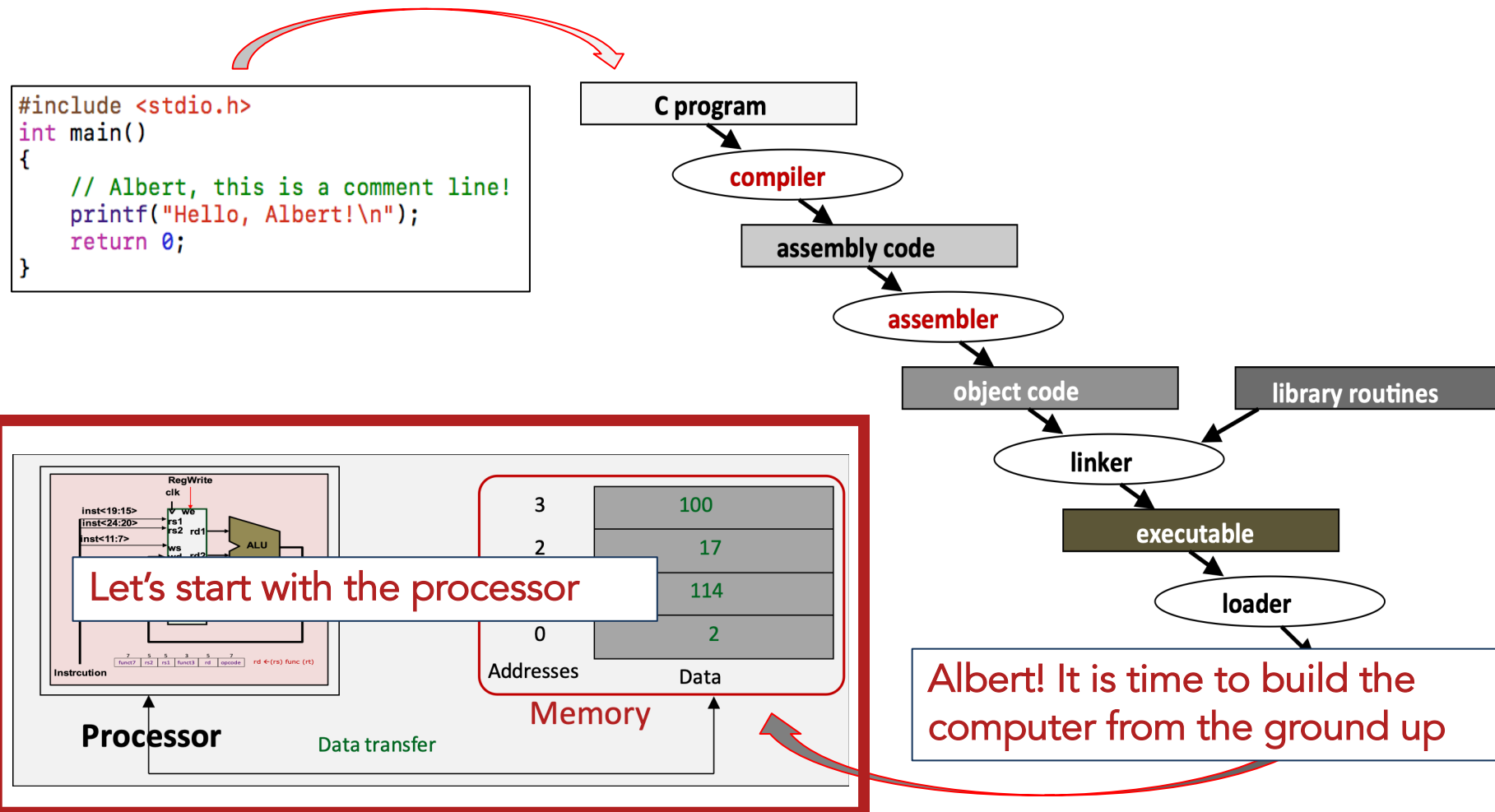
The Big Idea in Today's Computers

- Stored Program Computer

Program = A sequence of instructions

- Instructions represented in binary, just like data
- Instructions and data stored in memory
- Programs can operate on programs
 - e.g., compilers, linkers, ...
- Binary compatibility allows compiled programs to work on different computers
 - Standardized ISAs

The Big Picture



Next Lecture Module

- RISC-V ISA