CS 33

Machine Programming (4)

Not a Quiz!

What C code would you compile to get the following assembler code?

```
        movq    $0, %rax
.L2:
        movq    %rax, a(,%rax,8)
        addq    $1, %rax
        cmpq    $10, %rax
        jne     .L2
        ret
```

```
long a[10];
void func() {
  long i=0;
  while (i<10)
    a[i]= i++;
}
```

a

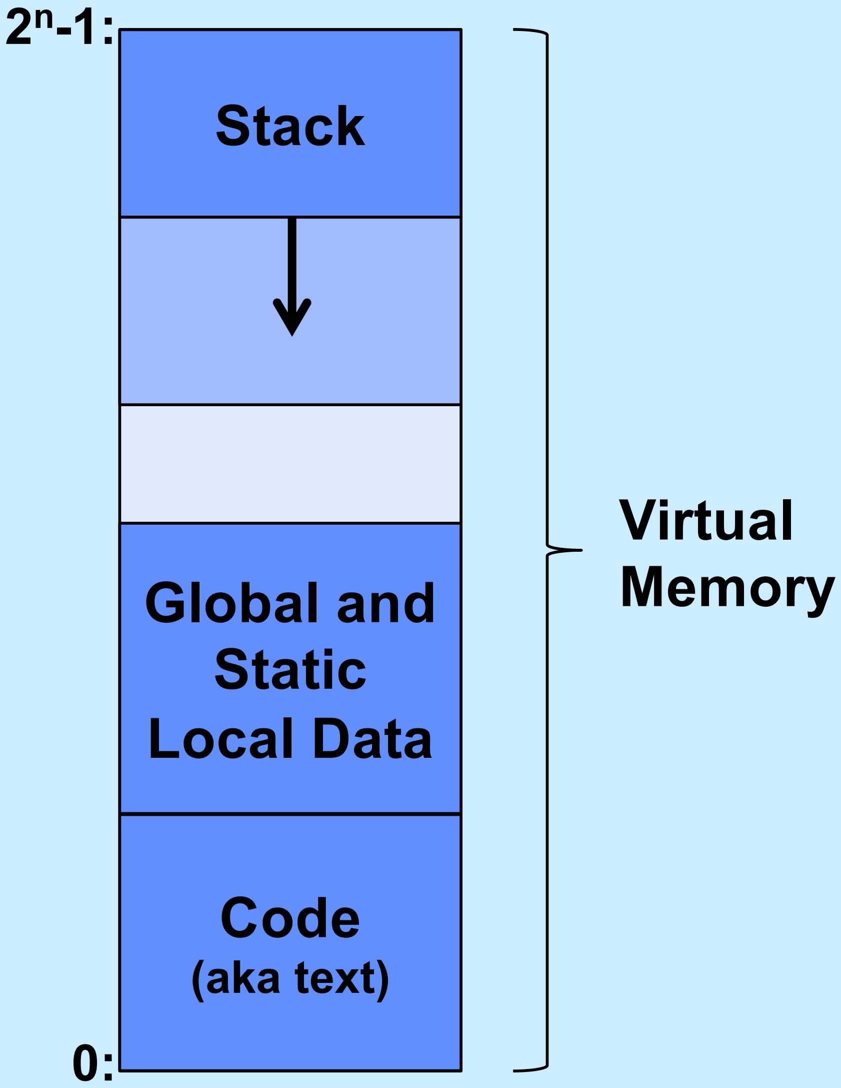
```
long a[10];
void func() {
  long i;
  for (i=0; i<10; i++)
    a[i]= 1;
}
```

b

```
long a[10];
void func() {
  long i=0;
  switch (i) {
case 0:
    a[i] = 0;
    break;
default:
    a[i] = 10
  }
}
```

c

Digression (Again): Where Stuff Is (Roughly)



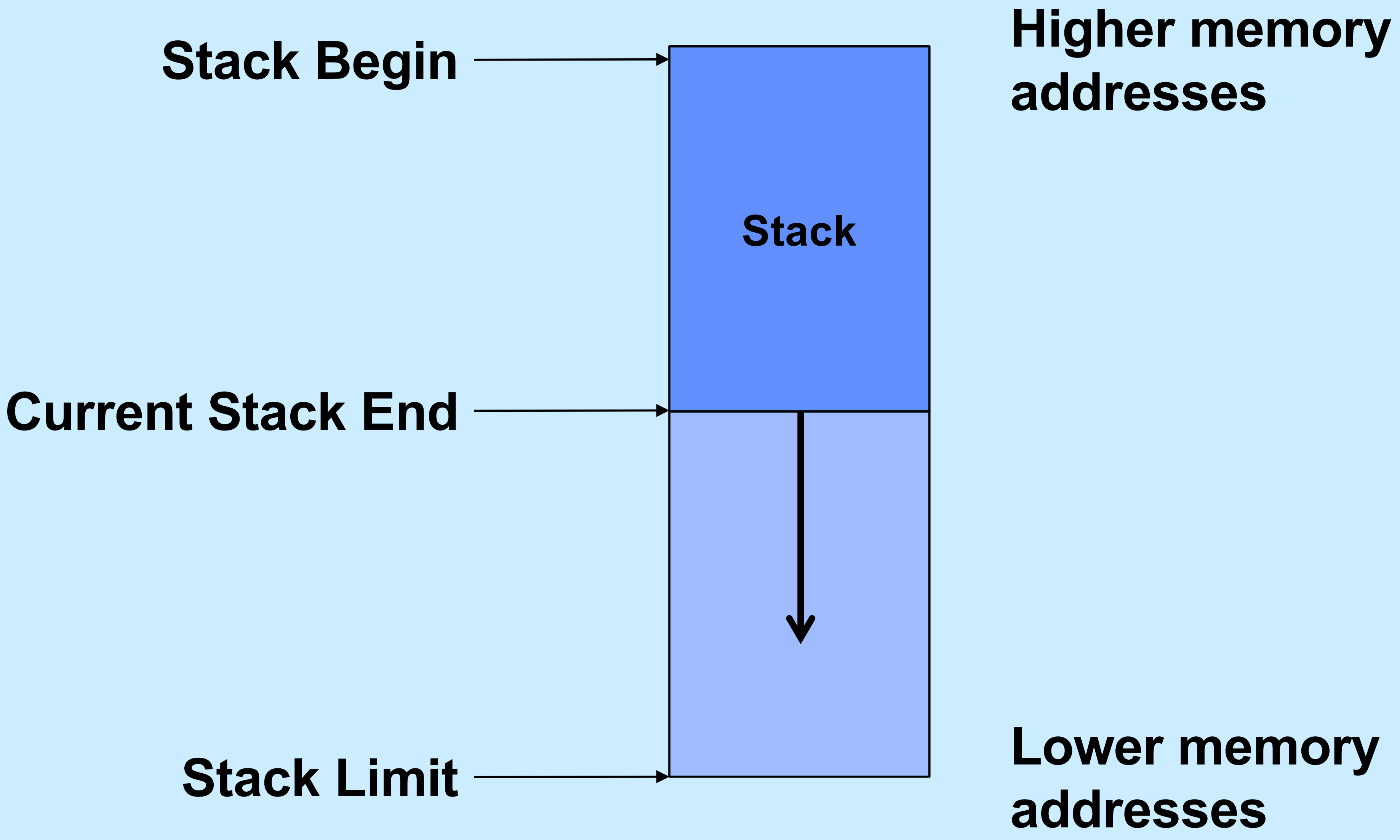
Function Call and Return

- Function A calls function B
- Function B calls function C

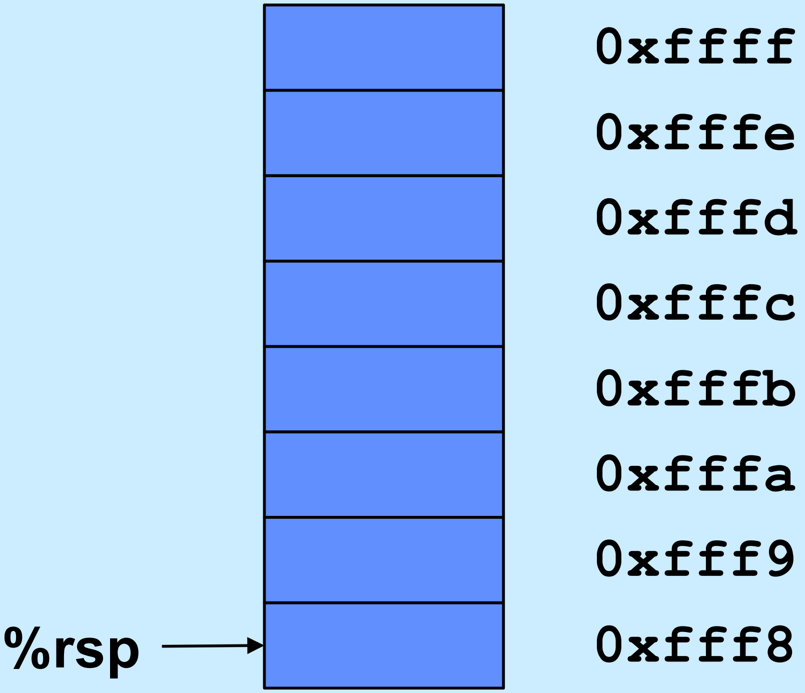
... several million instructions later

- C returns
  - how does it know to return to B?
- B returns
  - how does it know to return to A?

The Runtime Stack

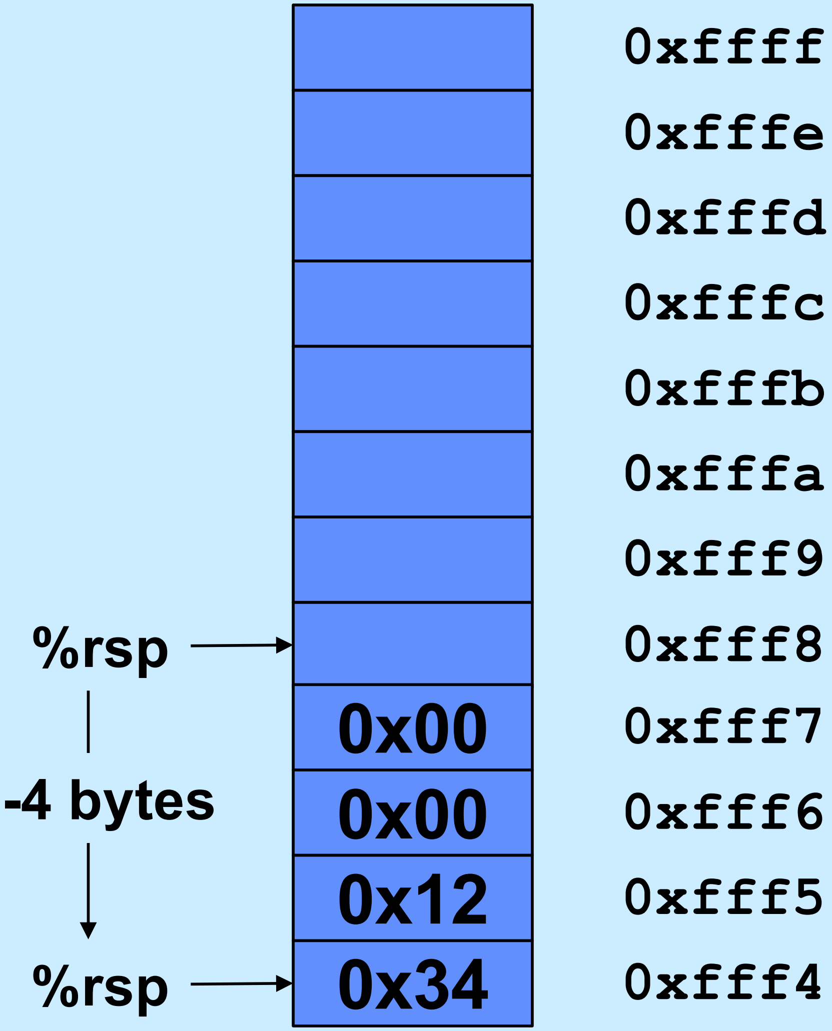


Stack Operations



Push

```
pushl $0x1234
```



0xffff

0xfffe

0xfffd

0xfffc

0xfffb

0xfffa

0xfff9

%rsp → 0xfff8

0x00 0xfff7

-4 bytes

0x00 0xfff6

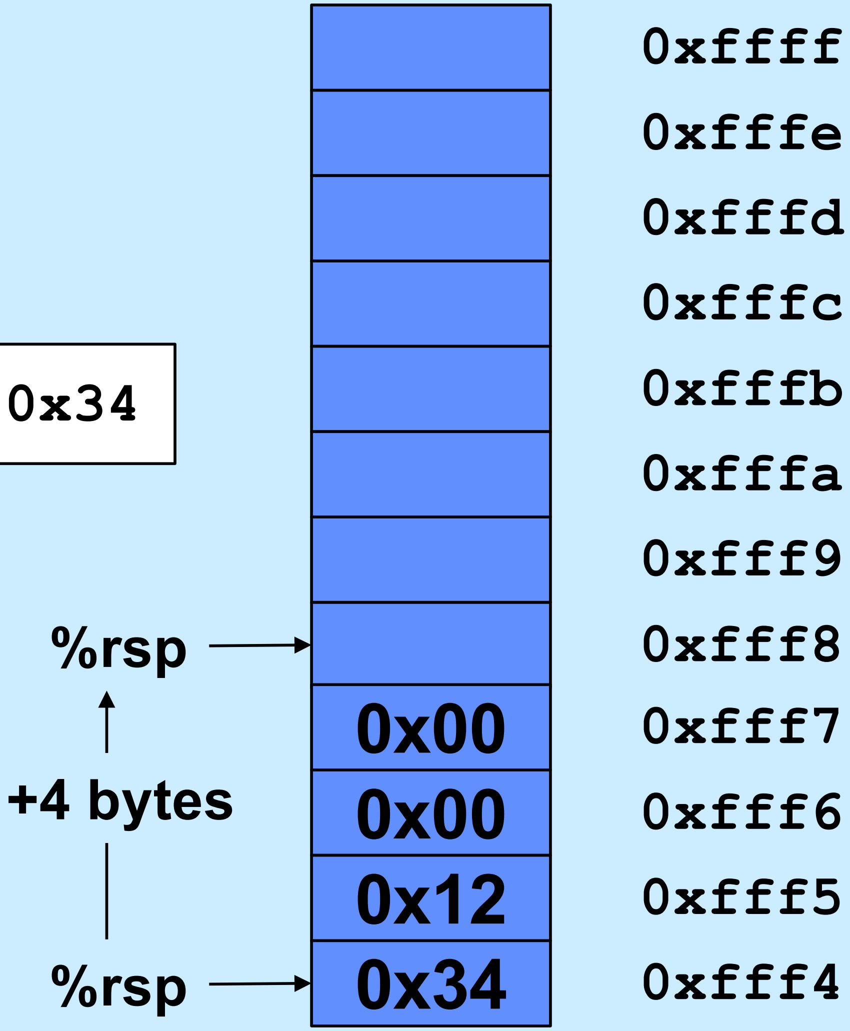
0x12 0xfff5

%rsp → 0x34 0xfff4

Pop

```
popl %r8d
```

%r8d:	0x00	0x00	0x12	0x34



	0xffff
	0xfffe
	0xfffd
	0xfffc
	0xfffb
	0xfffa
	0xfff9
	0xfff8
0x00	0xfff7
0x00	0xfff6
0x12	0xfff5
0x34	0xfff4

Call and Return

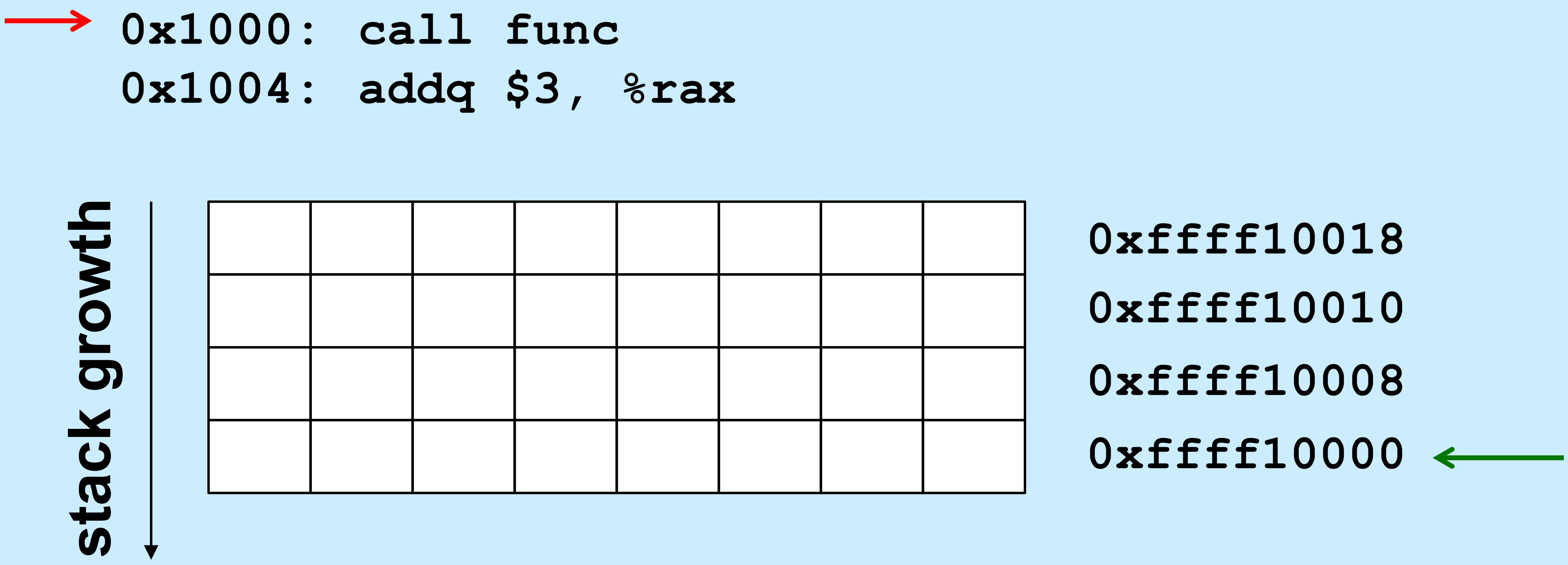
```
0x2000: func:
    ... ...
0x2200: movq $6, %rax
0x2203: ret
```

```
0x1000: call func
0x1004: addq $3, %rax
```

Call and Return

```
0x2000:  func:
    ...  ...
0x2200:  movq $6, %rax
0x2203:  ret
```

```
0x1000:  call func
0x1004:  addq $3, %rax
```



stack growth

0xffff10018

0xffff10010

0xffff10008

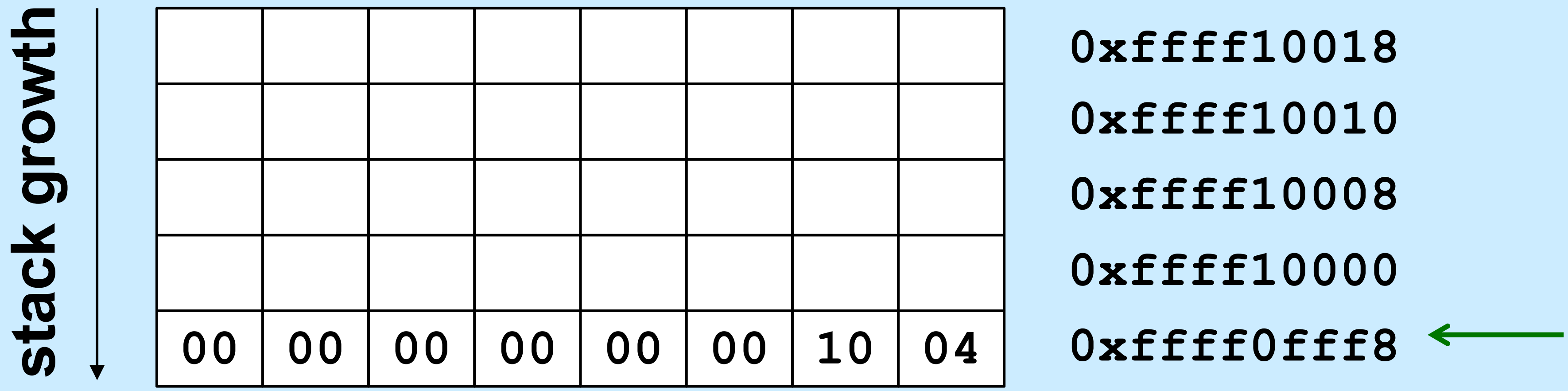
0xffff10000

								%rax
00	00	00	00	00	00	10	00	%rip
00	00	00	0f	ff	f1	00	00	%rsp

Call and Return

```
0x2000:  func:
    ...  ...
0x2200:  movq $6, %rax
0x2203:  ret
```

```
0x1000:  call func
0x1004:  addq $3, %rax
```



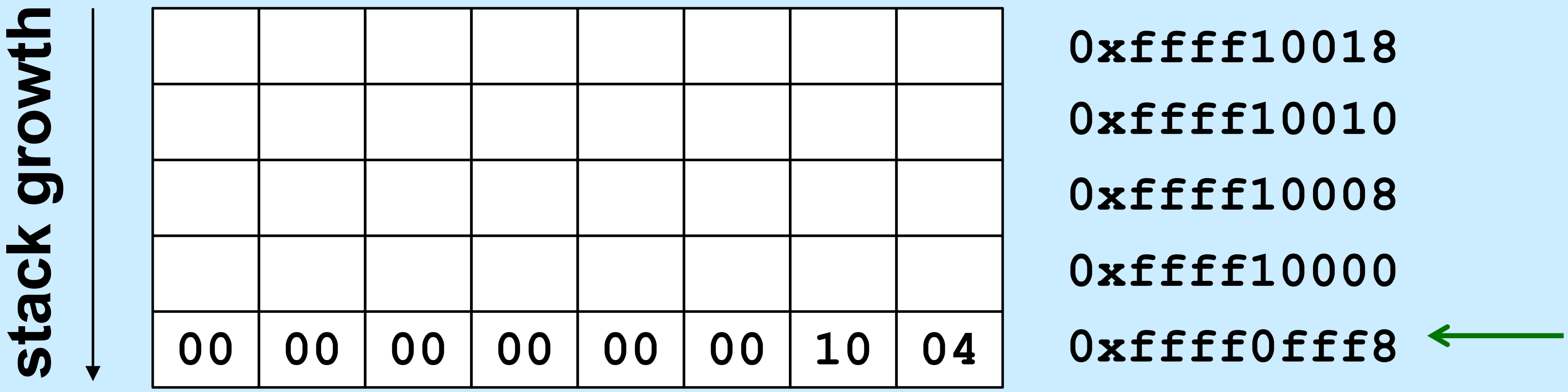
								0xffff10018
								0xffff10010
								0xffff10008
								0xffff10000
00	00	00	00	00	00	10	04	0xffff0fff8

								%rax
00	00	00	00	00	00	20	00	%rip
00	00	00	0f	ff	f0	ff	f8	%rsp

Call and Return

```
  0x2000:  func:
     ...   ...
  0x2200:  movq $6, %rax
→ 0x2203:  ret
```

```
0x1000:  call func
0x1004:  addq $3, %rax
```



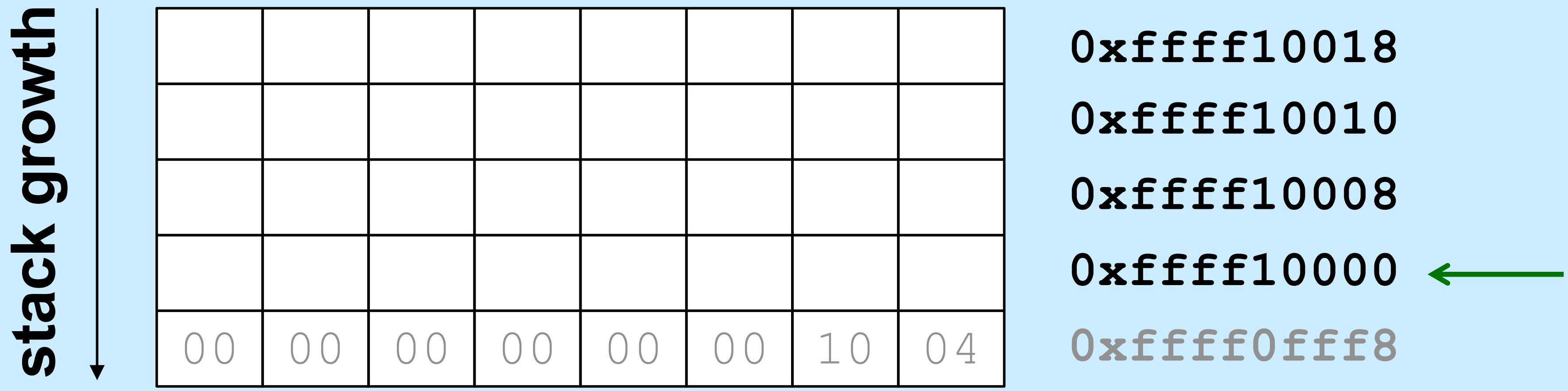
								0xffff10018	
								0xffff10010	
								0xffff10008	
								0xffff10000	
00	00	00	00	00	00	10	04	0xffff0fff8	←

00	00	00	00	00	00	00	06	%rax
00	00	00	00	00	00	22	03	%rip
00	00	00	0f	ff	f0	ff	f8	%rsp

Call and Return

```
0x2000:  func:
    ...  ...
0x2200:  movq $6, %rax
0x2203:  ret
```

```
   0x1000:  call func
→  0x1004:  addq $3, %rax
```



								0xffff10018
								0xffff10010
								0xffff10008
								0xffff10000 ←
00	00	00	00	00	00	10	04	0xffff0fff8

00	00	00	00	00	00	00	06	%rax
00	00	00	00	00	00	10	04	%rip
00	00	00	0f	ff	f1	00	00	%rsp

Arguments and Local Variables

```
int mainfunc() {
   long array[3] =
      {2,117,-6};
   long sum =
      ASum(array, 3);
   ...
   return sum;
}
```

```
long ASum(long *a,
      unsigned long size) {
   long i, sum = 0;
   for (i=0; i<size; i++)
      sum += a[i];
   return sum;
}
```

- **Local variables usually allocated on stack**
- **Arguments to functions pushed onto stack**

- **Local variables may be put in registers (and thus not on stack)**

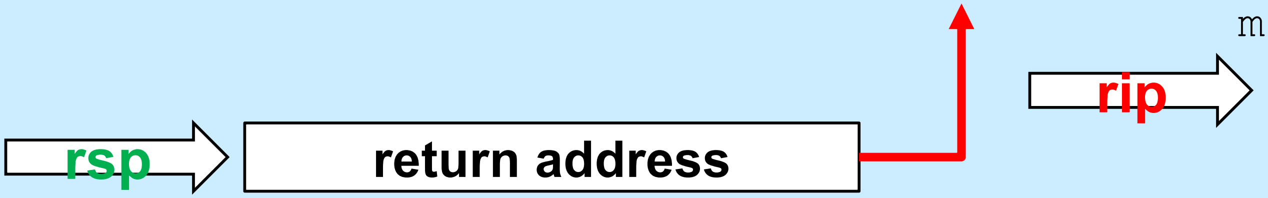
Arguments and Local Variables

```
mainfunc:
   pushq %rbp                 # save old %rbp
   movq %rsp, %rbp            # set %rbp to point to stack frame
   subq $32, %rsp             # alloc. space for locals (array and sum)
   movq $2, -32(%rbp)         # initialize array[0]
   movq $117, -24(%rbp)       # initialize array[1]
   movq $-6, -16(%rbp)        # initialize array[2]
   pushq $3                   # push arg 2
   leaq -32(%rbp), %rax       # array address is put in %rax
   pushq %rax                 # push arg 1
   call ASum
   addq $16, %rsp             # pop args
   movq %rax, -8(%rbp)        # copy return value to sum
   ...
   addq $32, %rsp             # pop locals
   popq %rbp                  # pop and restore old %rbp
   ret
```

Arguments and Local Variables

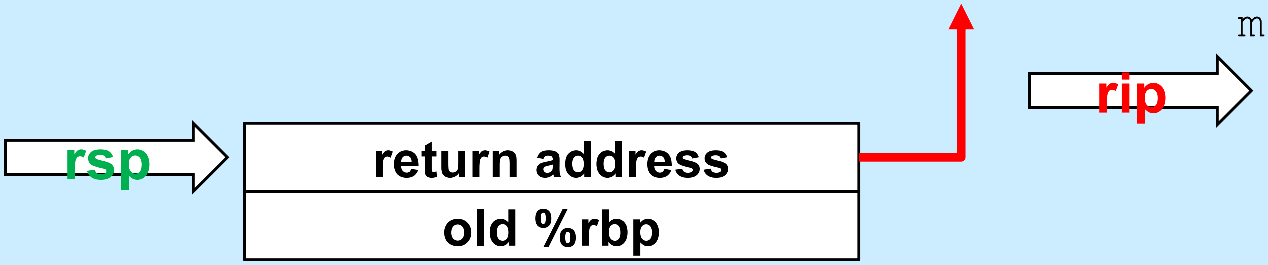
```
ASum:
   pushq %rbp                 # save old %rbp
   movq %rsp, %rbp            # set %rbp to point to stack frame
   movq $0, %rcx              # i in %rcx
   movq $0, %rax              # sum in %rax
   movq 16(%rbp), %rdx        # copy arg 1 (array) into %rdx
loop:
   cmpq 24(%rbp), %rcx        # i < size?
   jge done
   addq (%rdx,%rcx,8), %rax   # sum += a[i]
   incq %rcx                  # i++
   ja loop
done:
   popq %rbp                  # pop and restore %rbp
   ret
```

Enter mainfunc



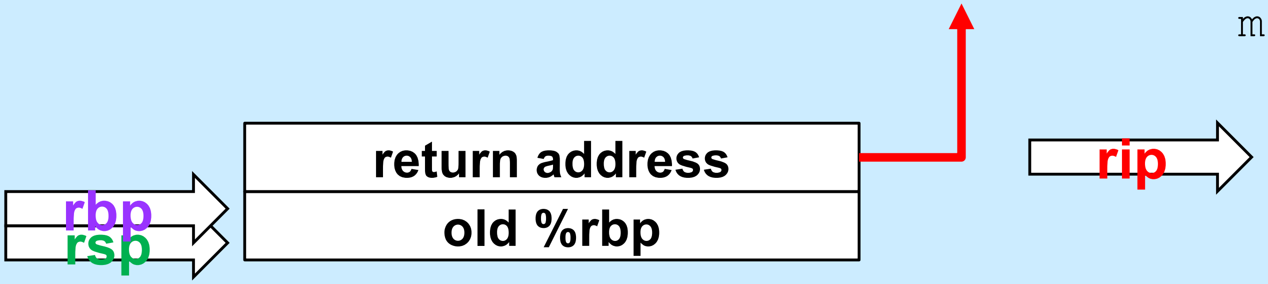
```
mainfunc:
   pushq %rbp
   movq %rsp, %rbp
   subq $32, %rsp
   movq $2, -32(%rbp)
   movq $117, -24(%rbp)
   movq $-6, -16(%rbp)
   pushq $3
   leaq -32(%rbp), %rax
   pushq %rax
   call ASum
   addq $16, %rsp
   movq %rax, -8(%rbp)
   addq $32, %rsp
   popq %rbp
   ret
```

Enter mainfunc



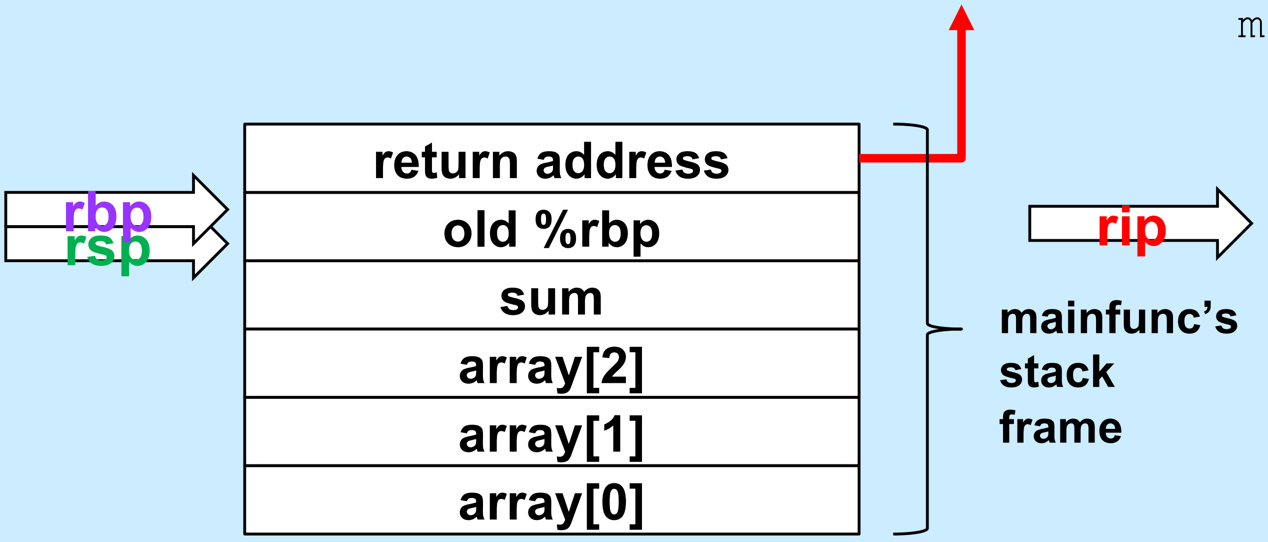
```
mainfunc:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq $2, -32(%rbp)
    movq $117, -24(%rbp)
    movq $-6, -16(%rbp)
    pushq $3
    leaq -32(%rbp), %rax
    pushq %rax
    call ASum
    addq $16, %rsp
    movq %rax, -8(%rbp)
    addq $32, %rsp
    popq %rbp
    ret
```

Setup Frame



```
mainfunc:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq $2, -32(%rbp)
    movq $117, -24(%rbp)
    movq $-6, -16(%rbp)
    pushq $3
    leaq -32(%rbp), %rax
    pushq %rax
    call ASum
    addq $16, %rsp
    movq %rax, -8(%rbp)
    addq $32, %rsp
    popq %rbp
    ret
```

Allocate Local Variables



rbp

rsp

return address

old %rbp

sum

array[2]

array[1]

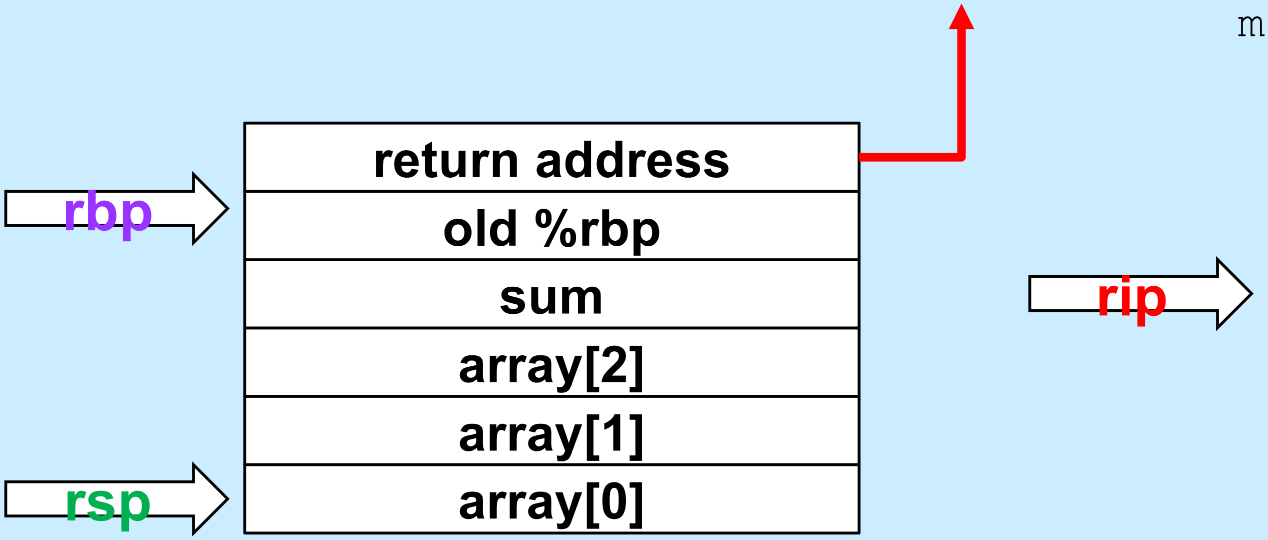
array[0]

mainfunc's stack frame

rip

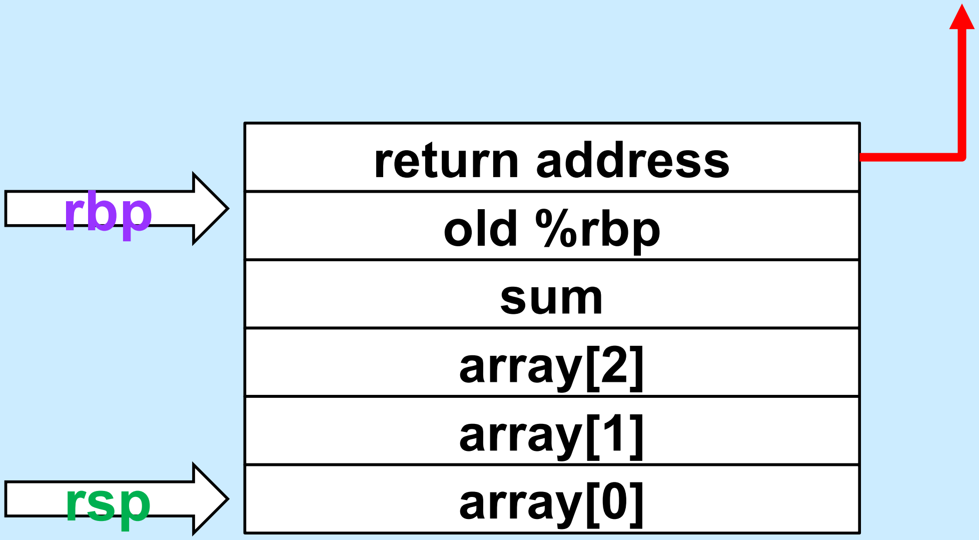
```
mainfunc:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq $2, -32(%rbp)
    movq $117, -24(%rbp)
    movq $-6, -16(%rbp)
    pushq $3
    leaq -32(%rbp), %rax
    pushq %rax
    call ASum
    addq $16, %rsp
    movq %rax, -8(%rbp)
    addq $32, %rsp
    popq %rbp
    ret
```

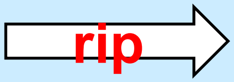
Initialize Local Array



```
mainfunc:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq $2, -32(%rbp)
    movq $117, -24(%rbp)
    movq $-6, -16(%rbp)
    pushq $3
    leaq -32(%rbp), %rax
    pushq %rax
    call ASum
    addq $16, %rsp
    movq %rax, -8(%rbp)
    addq $32, %rsp
    popq %rbp
    ret
```

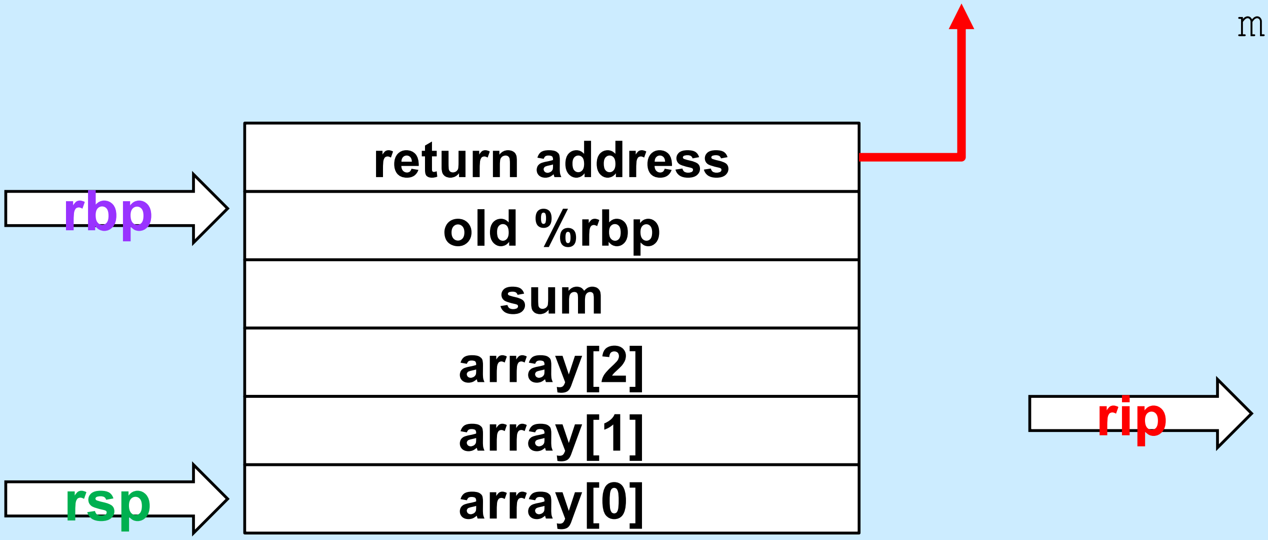
Initialize Local Array





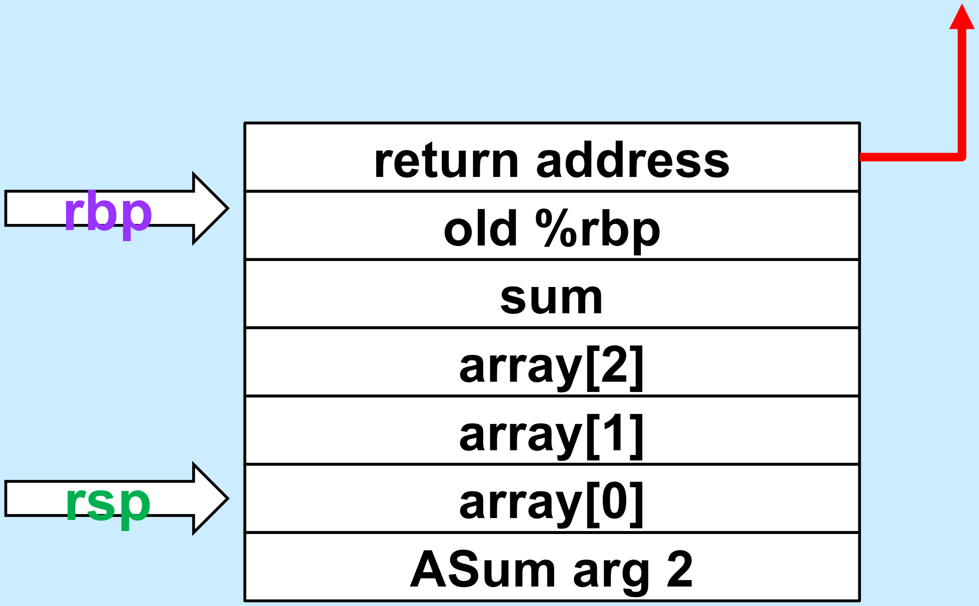
```
mainfunc:
      pushq %rbp
      movq %rsp, %rbp
      subq $32, %rsp
      movq $2, -32(%rbp)
      movq $117, -24(%rbp)
      movq $-6, -16(%rbp)
      pushq $3
      leaq -32(%rbp), %rax
      pushq %rax
      call ASum
      addq $16, %rsp
      movq %rax, -8(%rbp)
      addq $32, %rsp
      popq %rbp
      ret
```

Initialize Local Array

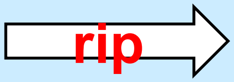


```
mainfunc:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq $2, -32(%rbp)
    movq $117, -24(%rbp)
    movq $-6, -16(%rbp)
    pushq $3
    leaq -32(%rbp), %rax
    pushq %rax
    call ASum
    addq $16, %rsp
    movq %rax, -8(%rbp)
    addq $32, %rsp
    popq %rbp
    ret
```

Push Second Argument

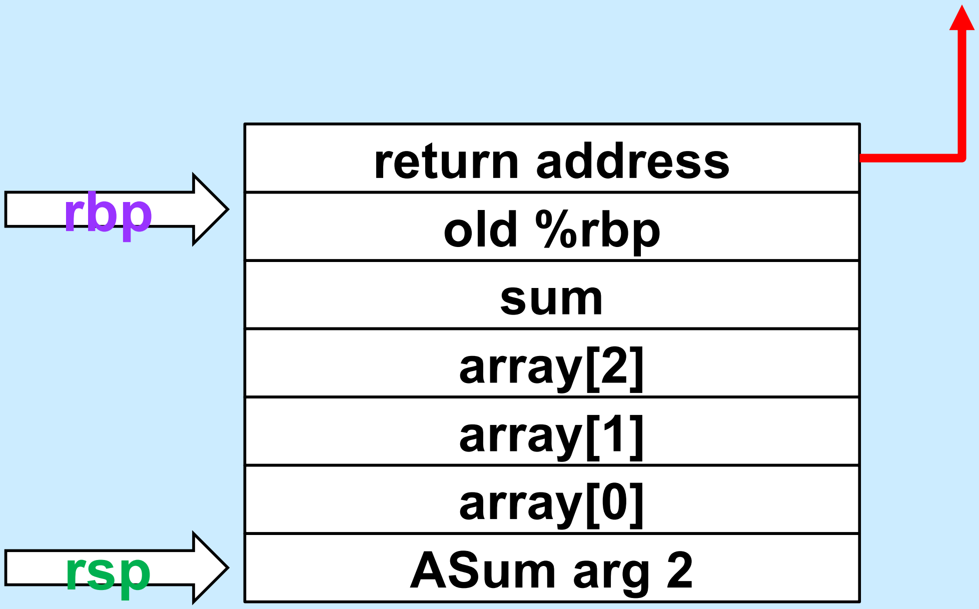


return address
old %rbp
sum
array[2]
array[1]
array[0]
ASum arg 2



```
mainfunc:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq $2, -32(%rbp)
    movq $117, -24(%rbp)
    movq $-6, -16(%rbp)
    pushq $3
    leaq -32(%rbp), %rax
    pushq %rax
    call ASum
    addq $16, %rsp
    movq %rax, -8(%rbp)
    addq $32, %rsp
    popq %rbp
    ret
```

Get Array Address



rbp

rsp

return address

old %rbp

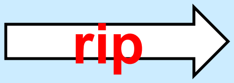
sum

array[2]

array[1]

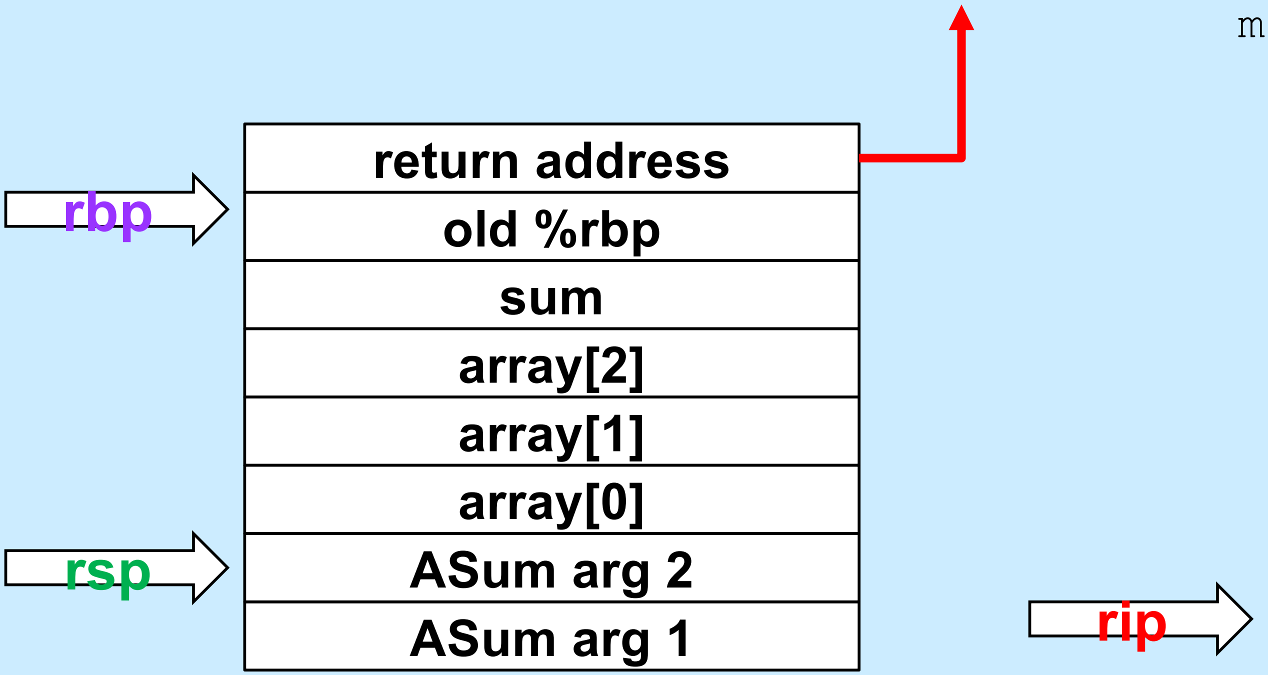
array[0]

ASum arg 2



```
mainfunc:
   pushq %rbp
   movq %rsp, %rbp
   subq $32, %rsp
   movq $2, -32(%rbp)
   movq $117, -24(%rbp)
   movq $-6, -16(%rbp)
   pushq $3
   leaq -32(%rbp), %rax
   pushq %rax
   call ASum
   addq $16, %rsp
   movq %rax, -8(%rbp)
   addq $32, %rsp
   popq %rbp
   ret
```

Push First Argument



rbp

rsp

return address

old %rbp

sum

array[2]

array[1]

array[0]

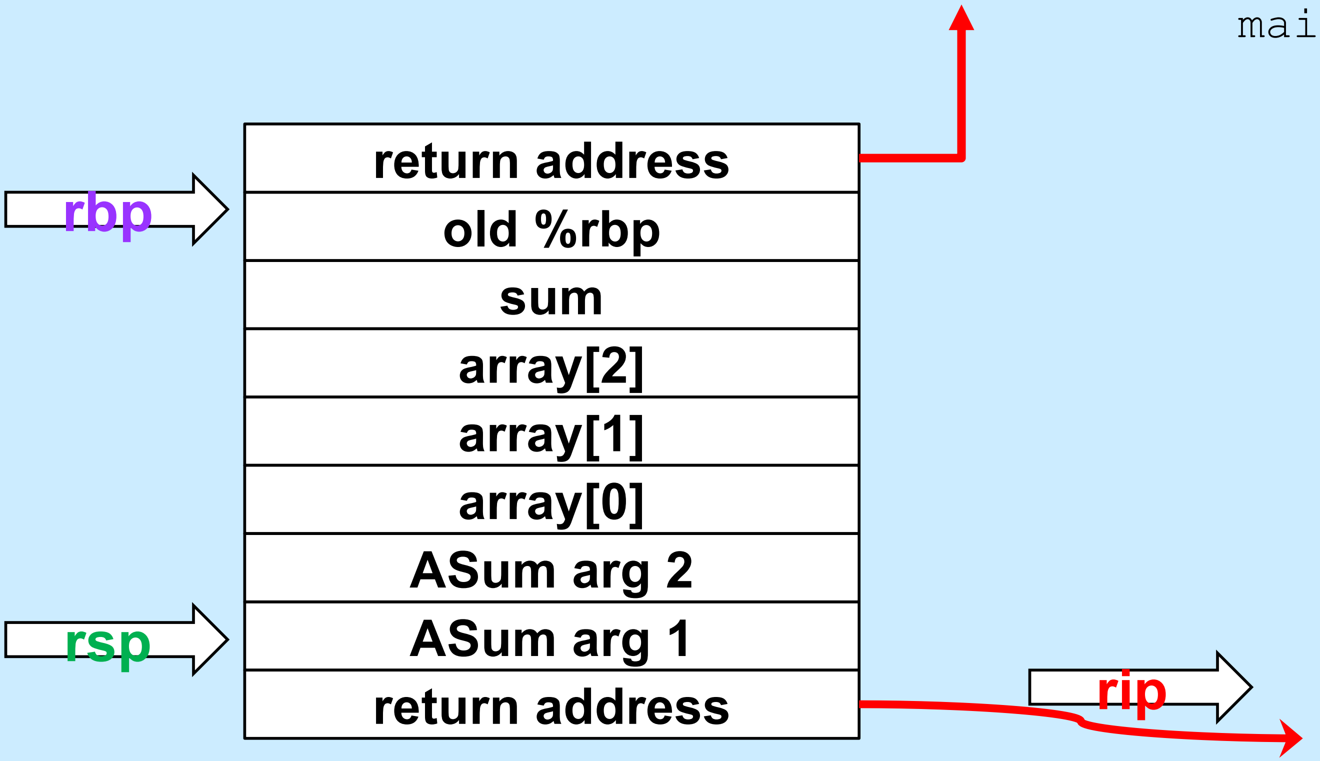
ASum arg 2

ASum arg 1

rip

```
mainfunc:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq $2, -32(%rbp)
    movq $117, -24(%rbp)
    movq $-6, -16(%rbp)
    pushq $3
    leaq -32(%rbp), %rax
    pushq %rax
    call ASum
    addq $16, %rsp
    movq %rax, -8(%rbp)
    addq $32, %rsp
    popq %rbp
    ret
```

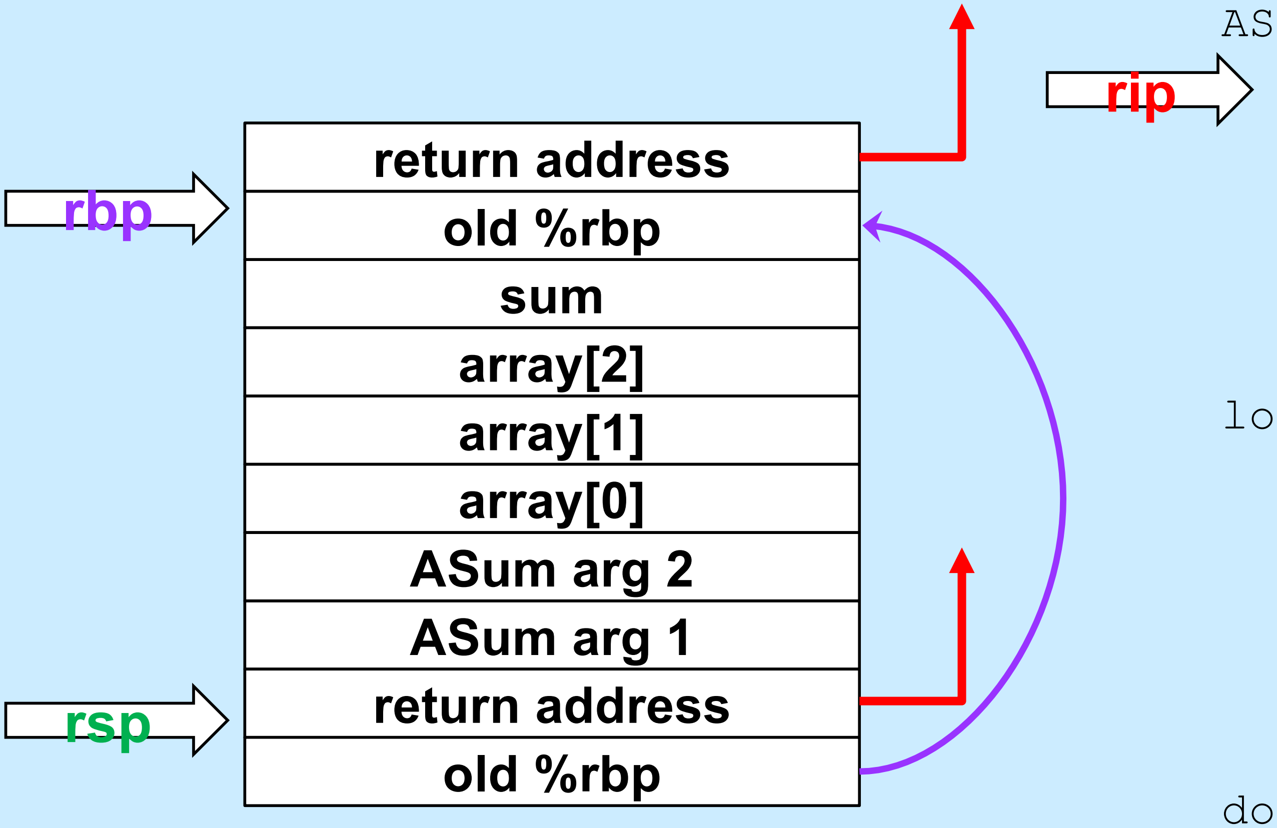
Call ASum



return address
old %rbp
sum
array[2]
array[1]
array[0]
ASum arg 2
ASum arg 1
return address

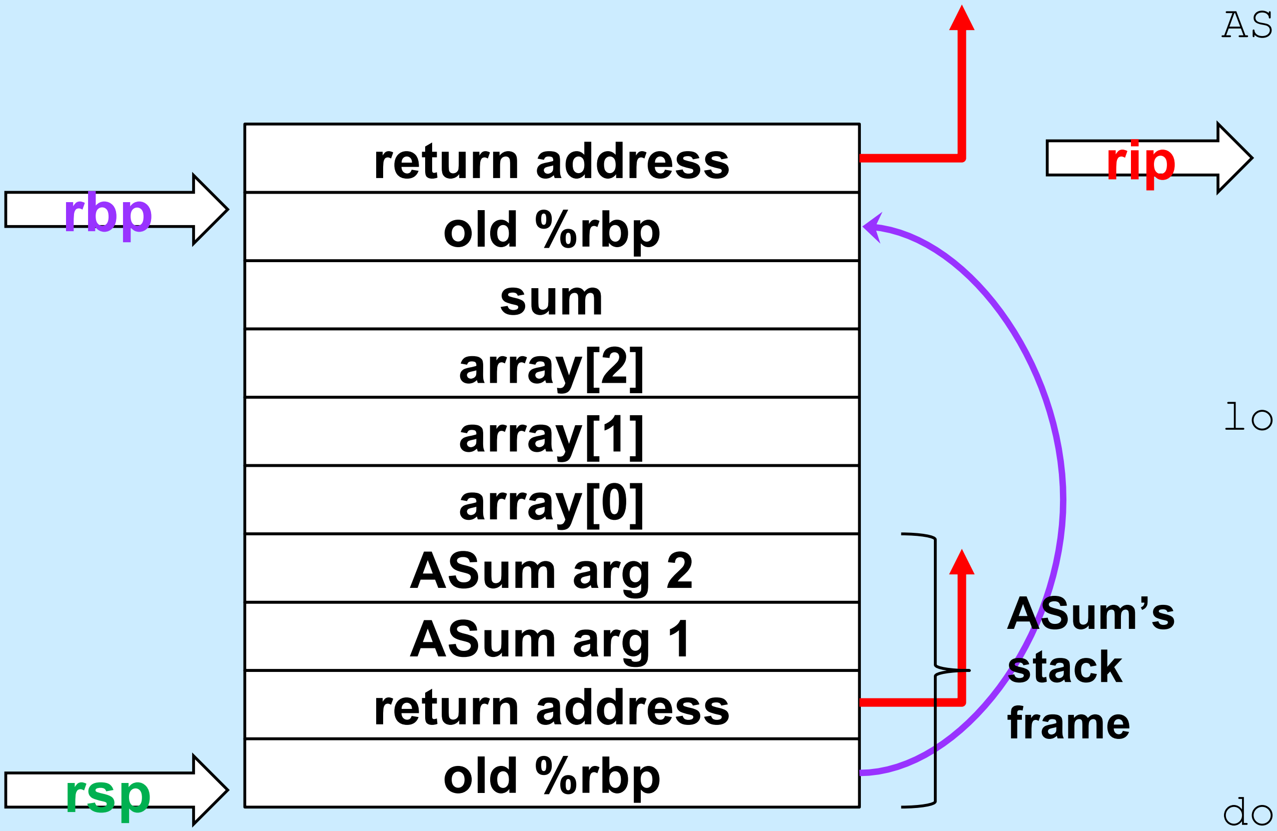
```
mainfunc:
   pushq %rbp
   movq %rsp, %rbp
   subq $32, %rsp
   movq $2, -32(%rbp)
   movq $117, -24(%rbp)
   movq $-6, -16(%rbp)
   pushq $3
   leaq -32(%rbp), %rax
   pushq %rax
   call ASum
   addq $16, %rsp
   movq %rax, -8(%rbp)
   addq $32, %rsp
   popq %rbp
   ret
```

Enter ASum



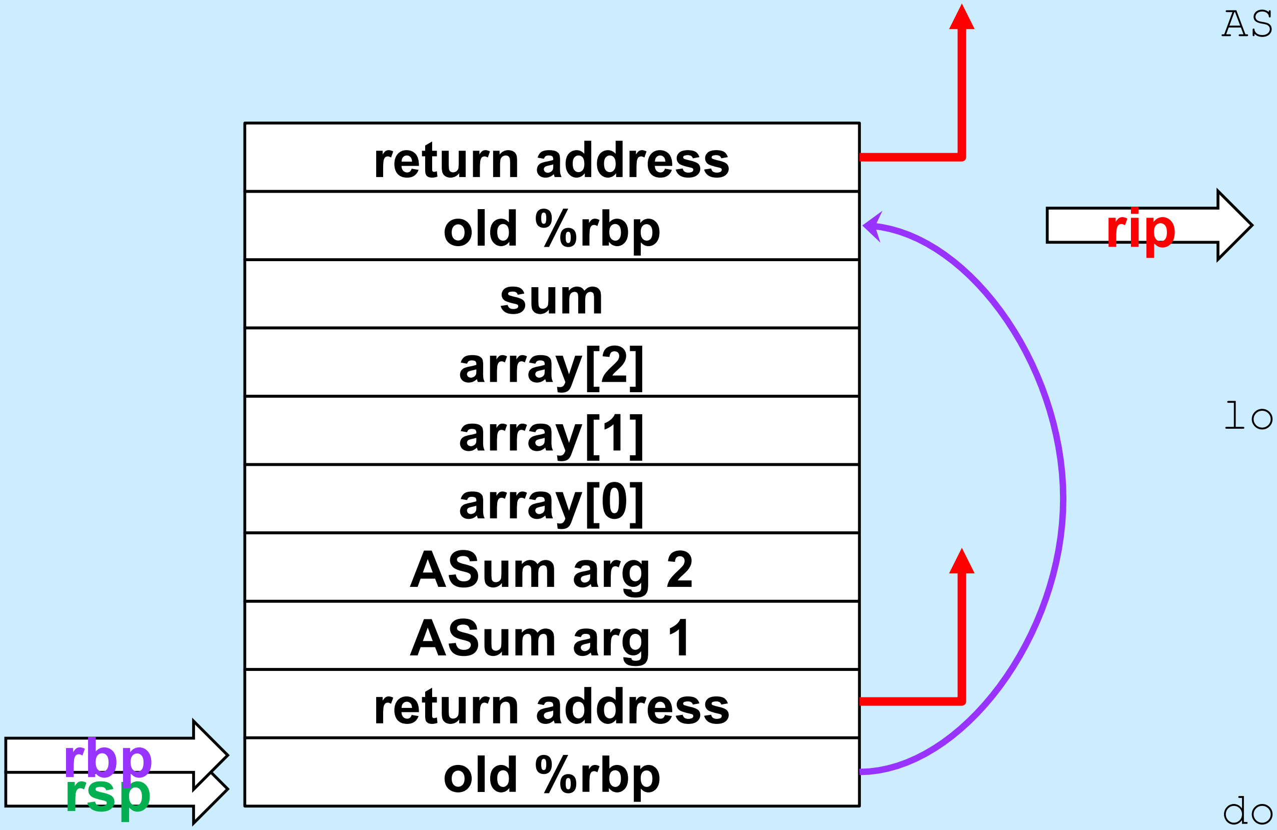
```
ASum:
    pushq %rbp
    movq %rsp, %rbp
    movq $0, %rcx
    movq $0, %rax
    movq 16(%rbp), %rdx
loop:
    cmpq 24(%rbp), %rcx
    jge done
    addq (%rdx,%rcx,8), %rax
    incq  %rcx
    ja loop
done:
    popq %rbp
    ret
```

Setup Frame



```
ASum:
   pushq %rbp
   movq %rsp, %rbp
   movq $0, %rcx
   movq $0, %rax
   movq 16(%rbp), %rdx
loop:
   cmpq 24(%rbp), %rcx
   jge done
   addq (%rdx,%rcx,8), %rax
   incq  %rcx
   ja loop
done:
   popq %rbp
   ret
```

Execute the Function



return address

old %rbp

sum

array[2]

array[1]

array[0]

ASum arg 2

ASum arg 1

return address

old %rbp

```
ASum:
    pushq %rbp
    movq %rsp, %rbp
    movq $0, %rcx
    movq $0, %rax
    movq 16(%rbp), %rdx
loop:
    cmpq 24(%rbp), %rcx
    jge done
    addq (%rdx,%rcx,8), %rax
    incq  %rcx
    ja loop
done:
    popq %rbp
    ret
```

Quiz 1

What's at 16(%rbp)?

a) a local variable

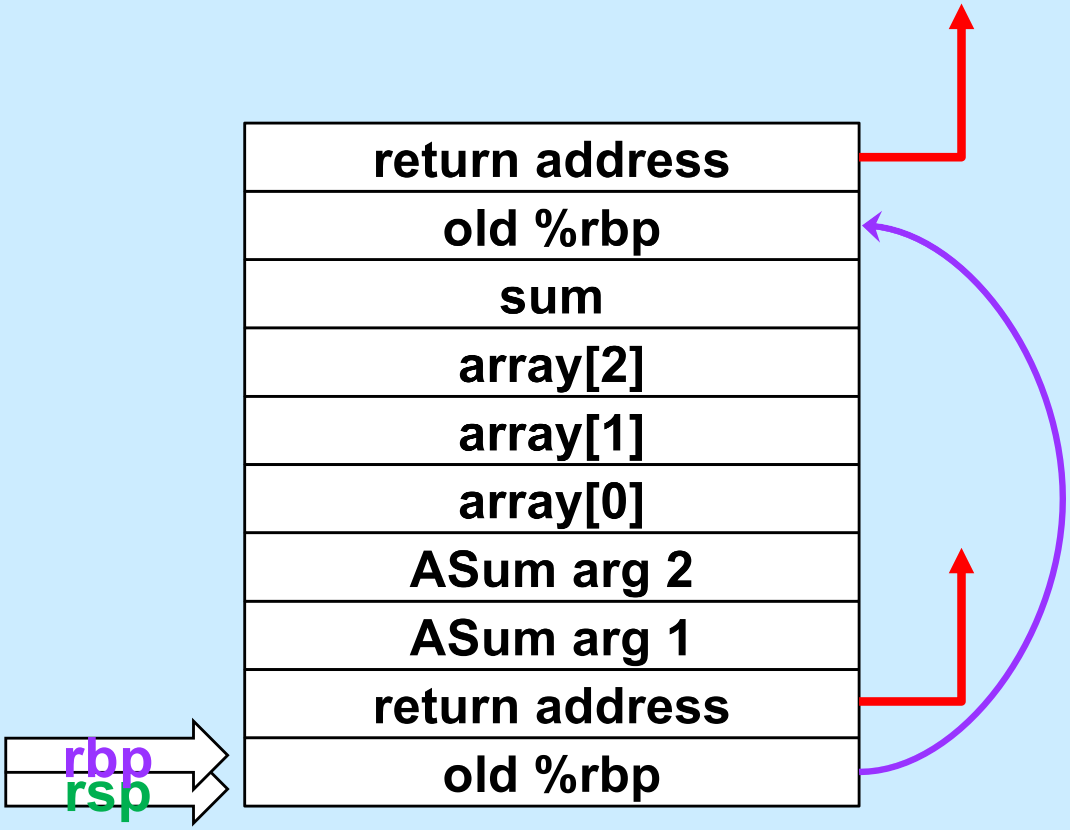
b) the first argument to ASum

c) the second argument to ASum

d) something else

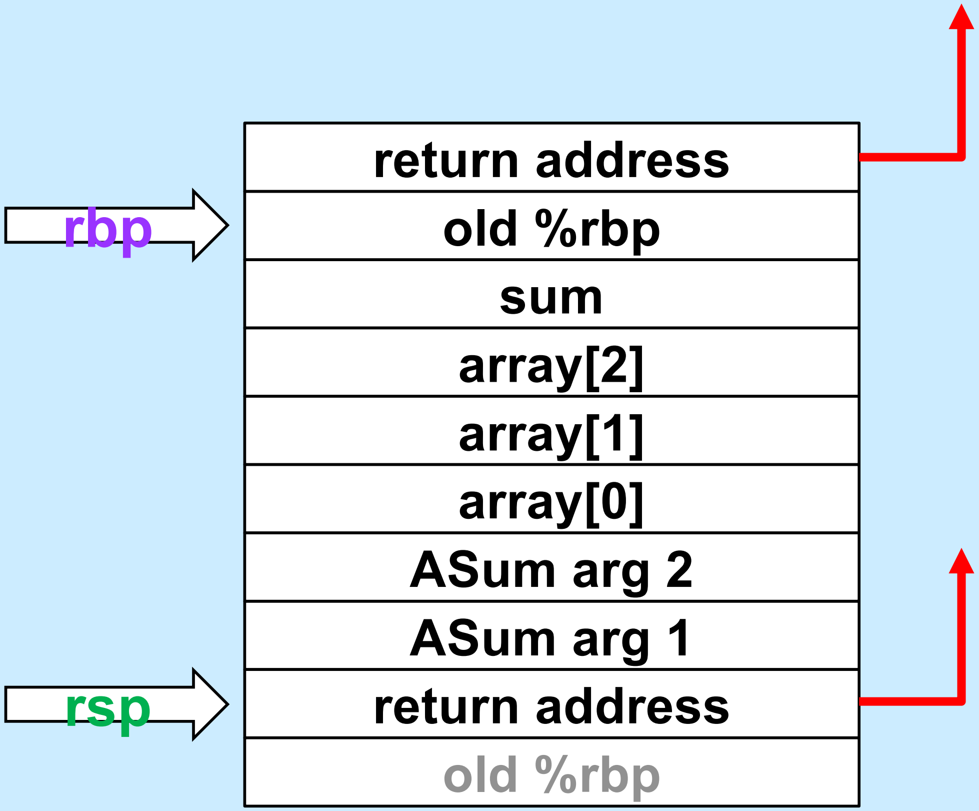
```
ASum:
   pushq %rbp
   movq %rsp, %rbp
   movq $0, %rcx
   movq $0, %rax
   movq 16(%rbp), %rdx
loop:
   cmpq 24(%rbp), %rcx
   jge done
   addq (%rdx,%rcx,8), %rax
   incq  %rcx
   ja loop
done:
   popq %rbp
   ret
```

Prepare to Return



```
ASum:
    pushq %rbp
    movq %rsp, %rbp
    movq $0, %rcx
    movq $0, %rax
    movq 16(%rbp), %rdx
loop:
    cmpq 24(%rbp), %rcx
    jge done
    addq (%rdx,%rcx,8), %rax
    incq  %rcx
    ja loop
done:
    popq %rbp
    ret
```

Return



return address

old %rbp

sum

array[2]

array[1]

array[0]

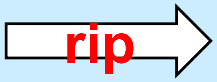
ASum arg 2

ASum arg 1

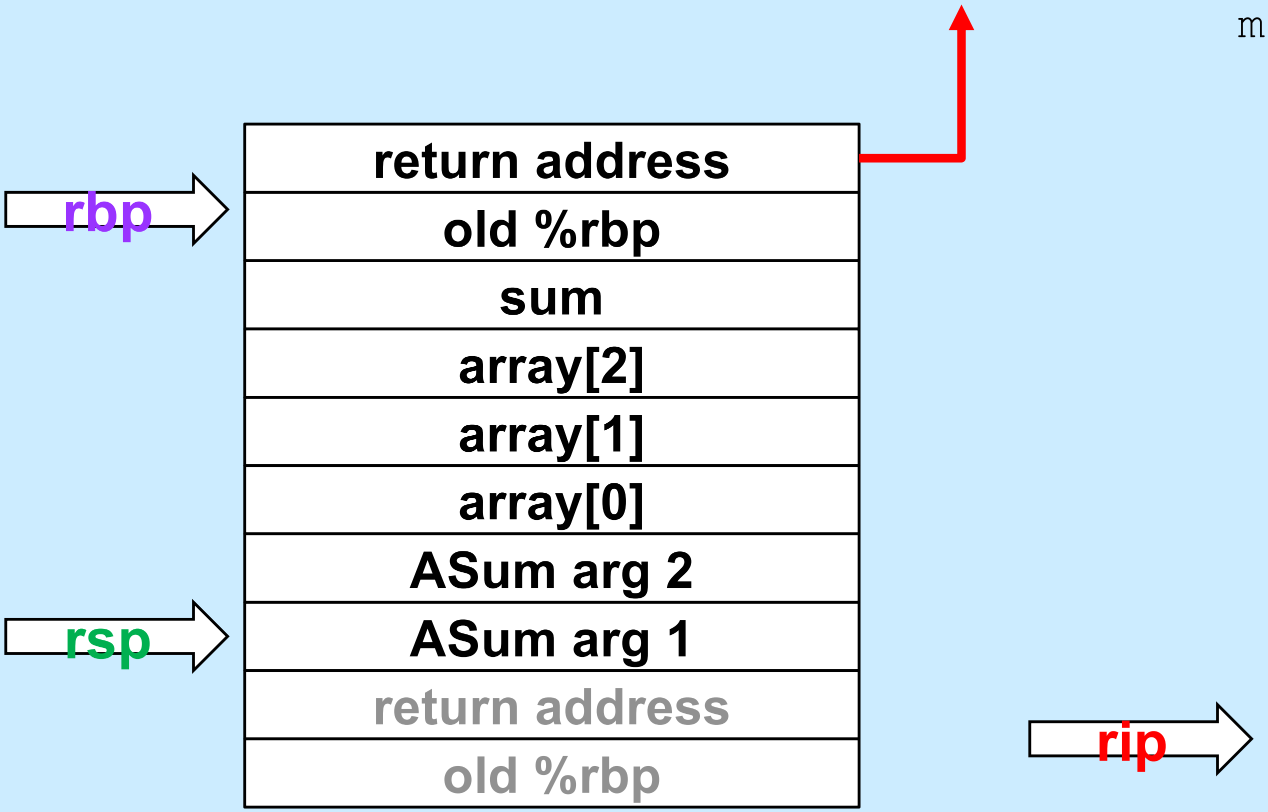
return address

old %rbp

```
ASum:
    pushq %rbp
    movq %rsp, %rbp
    movq $0, %rcx
    movq $0, %rax
    movq 16(%rbp), %rdx
loop:
    cmpq 24(%rbp), %rcx
    jge done
    addq (%rdx,%rcx,8), %rax
    incq  %rcx
    ja loop
done:
    popq %rbp
    ret
```

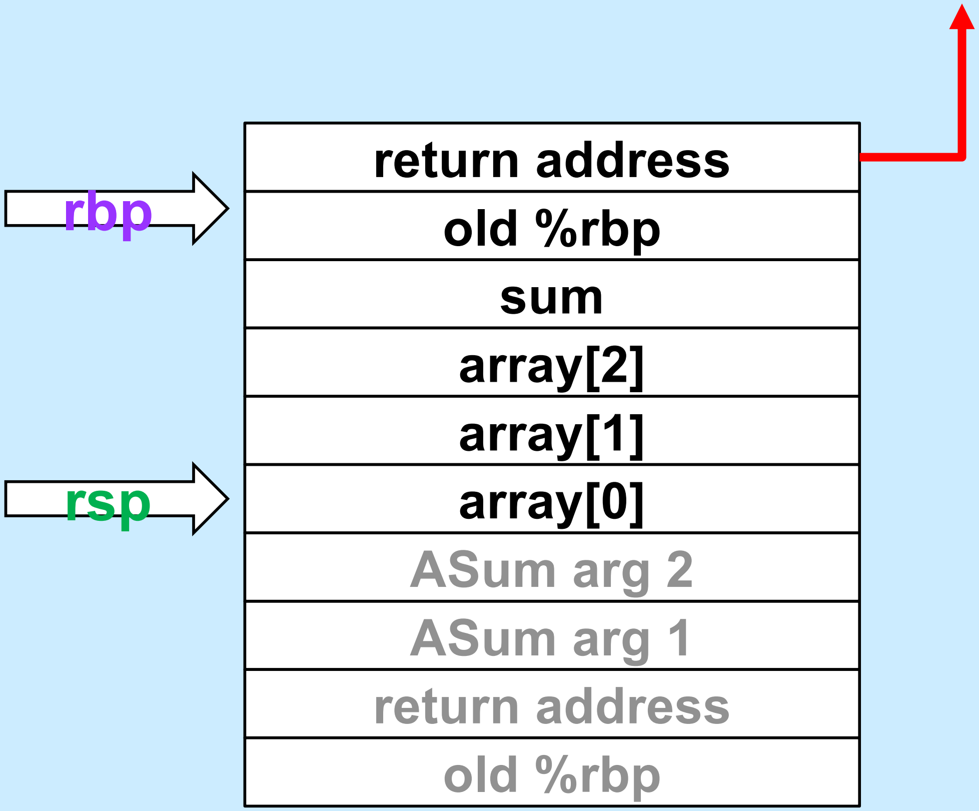


Pop Arguments

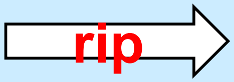


```
mainfunc:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq $2, -32(%rbp)
    movq $117, -24(%rbp)
    movq $-6, -16(%rbp)
    pushq $3
    leaq -32(%rbp), %rax
    pushq %rax
    call ASum
    addq $16, %rsp
    movq %rax, -8(%rbp)
    addq $32, %rsp
    popq %rbp
    ret
```

Save Return Value

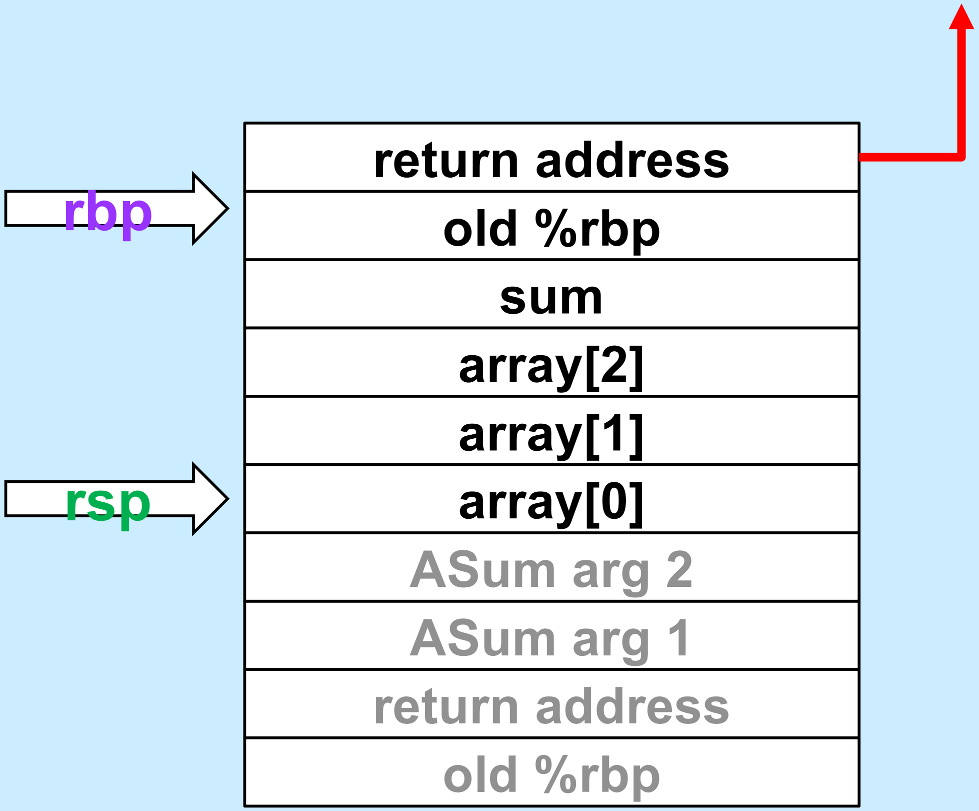


return address
old %rbp
sum
array[2]
array[1]
array[0]
ASum arg 2
ASum arg 1
return address
old %rbp



```
mainfunc:
   pushq %rbp
   movq %rsp, %rbp
   subq $32, %rsp
   movq $2, -32(%rbp)
   movq $117, -24(%rbp)
   movq $-6, -16(%rbp)
   pushq $3
   leaq -32(%rbp), %rax
   pushq %rax
   call ASum
   addq $16, %rsp
   movq %rax, -8(%rbp)
   addq $32, %rsp
   popq %rbp
   ret
```

Pop Local Variables



rbp

rsp

return address

old %rbp

sum

array[2]

array[1]

array[0]

ASum arg 2

ASum arg 1

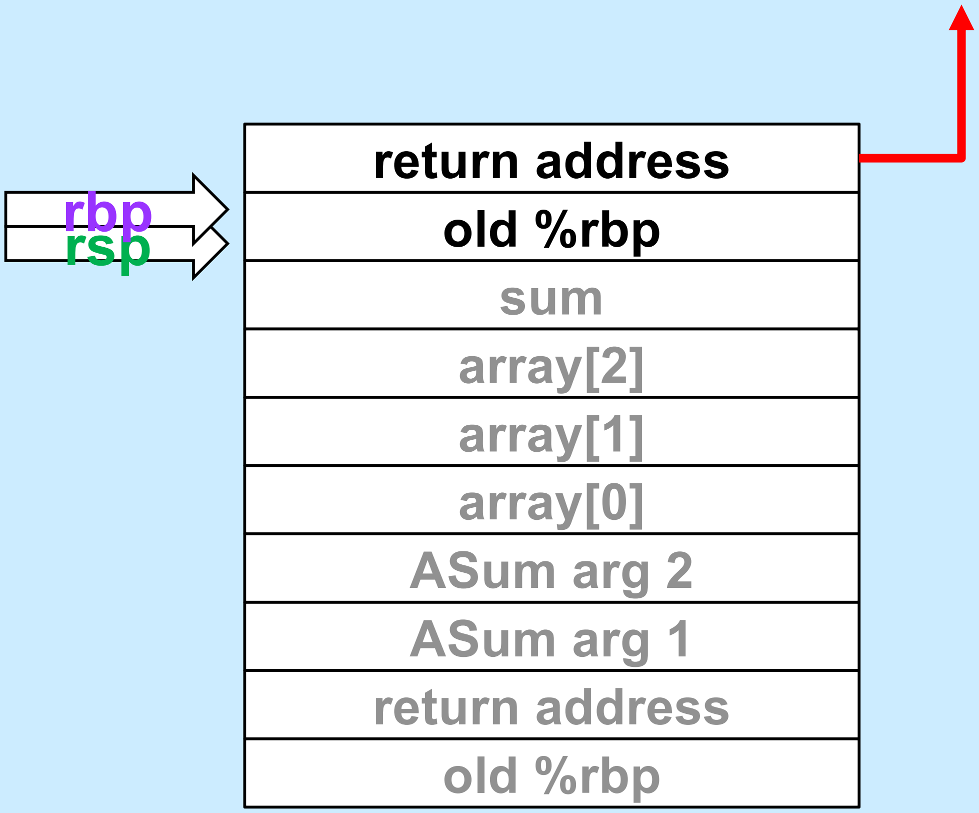
return address

old %rbp

rip

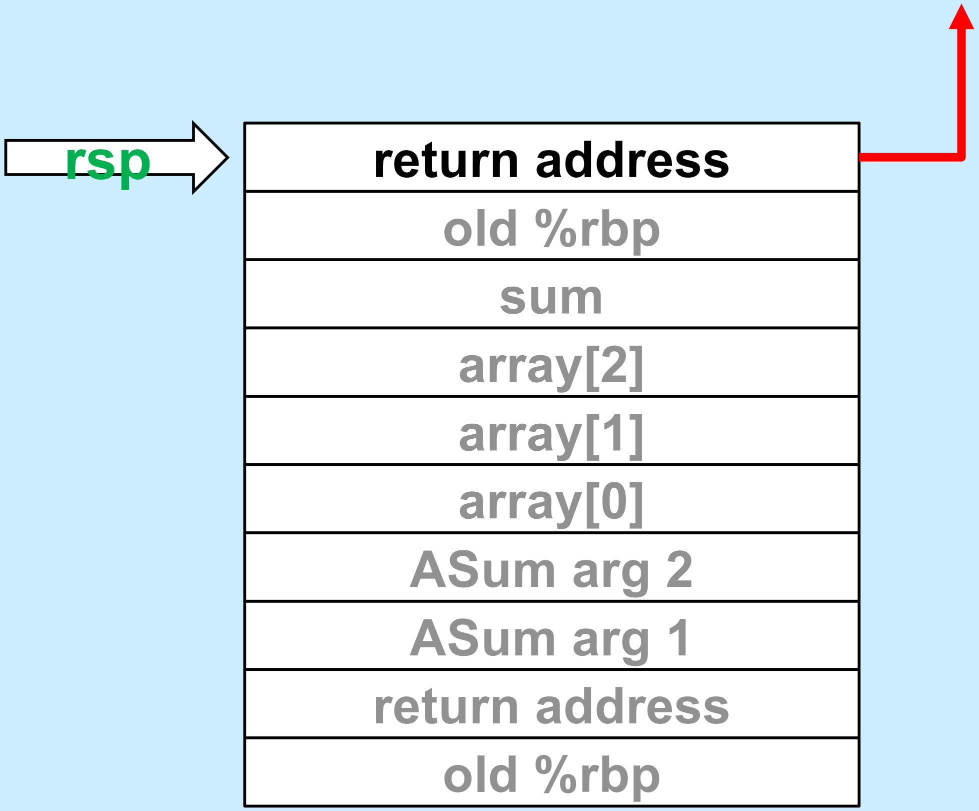
```
mainfunc:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq $2, -32(%rbp)
    movq $117, -24(%rbp)
    movq $-6, -16(%rbp)
    pushq $3
    leaq -32(%rbp), %rax
    pushq %rax
    call ASum
    addq $16, %rsp
    movq %rax, -8(%rbp)
    addq $32, %rsp
    popq %rbp
    ret
```

Prepare to Return

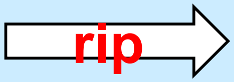


```
mainfunc:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq $2, -32(%rbp)
    movq $117, -24(%rbp)
    movq $-6, -16(%rbp)
    pushq $3
    leaq -32(%rbp), %rax
    pushq %rax
    call ASum
    addq $16, %rsp
    movq %rax, -8(%rbp)
    addq $32, %rsp
    popq %rbp
    ret
```

Return



```
mainfunc:
    pushq %rbp
    movq %rsp, %rbp
    subq $32, %rsp
    movq $2, -32(%rbp)
    movq $117, -24(%rbp)
    movq $-6, -16(%rbp)
    pushq $3
    leaq -32(%rbp), %rax
    pushq %rax
    call ASum
    addq $16, %rsp
    movq %rax, -8(%rbp)
    addq $32, %rsp
    popq %rbp
    ret
```



Using Registers

- **ASum modifies registers:**
  - **%rsp**
  - **%rbp**
  - **%rcx**
  - **%rax**
  - **%rdx**
- **Suppose its caller uses these registers**

```
...
movq $33, %rcx
movq $167, %rdx
pushq $6
pushq array
call ASum
  # assumes unmodified %rcx and %rdx
addq $16, %rsp
addq %rax,%rcx    # %rcx was modified!
addq %rdx, %rcx   # %rdx was modified!
```

```
ASum:
   pushq %rbp
   movq %rsp, %rbp
   movq $0, %rcx
   movq $0, %rax
   movq 16(%rbp), %rdx
loop:
   cmpq 24(%rbp), %rcx
   jge done
   addq (%rdx,%rcx,8), %rax
   incq  %rcx
   ja loop
done:
   popq %rbp
   ret
```

Register Values Across Function Calls

- **ASum modifies registers:**
  - **%rsp**
  - **%rbp**
  - **%rcx**
  - **%rax**
  - **%rdx**
- **May the caller of ASum depend on its registers being the same on return?**
  - **ASum saves and restores %rbp and makes no net changes to %rsp**
    - **their values are unmodified on return to its caller**
  - **%rax, %rcx, and %rdx are not saved and restored**
    - **their values might be different on return**

```
ASum:
   pushq %rbp
   movq %rsp, %rbp
   movq $0, %rcx
   movq $0, %rax
   movq 16(%rbp), %rdx
loop:
   cmpq 24(%rbp), %rcx
   jge done
   addq (%rdx,%rcx,8), %rax
   incq  %rcx
   ja loop
done:
   popq %rbp
   ret
```

Register-Saving Conventions

- Caller-save registers
  - if the caller wants their values to be the same on return from function calls, it must save and restore them

```
    pushq %rcx
    call func
    popq %rcx
```

- Callee-save registers
  - if the callee wants to use these registers, it must first save them, then restore their values before returning

```
func:
    pushq %rbx
    movq $6, %rbx
    ...
    popq %rbx
```

x86-64 General-Purpose Registers: Usage Conventions

`%rax`	Return value	`%r8`	Caller saved
`%rbx`	Callee saved	`%r9`	Caller saved
`%rcx`	Caller saved	`%r10`	Caller saved
`%rdx`	Caller saved	`%r11`	Caller Saved
`%rsi`	Caller saved	`%r12`	Callee saved
`%rdi`	Caller saved	`%r13`	Callee saved
`%rsp`	Stack pointer	`%r14`	Callee saved
`%rbp`	Base pointer	`%r15`	Callee saved

Passing Arguments in Registers

- Observations
  - accessing registers is much faster than accessing primary memory
    - » if arguments were in registers rather than on the stack, speed would increase
  - most functions have just a few arguments

- Actions
  - change calling conventions so that the first six arguments are passed in registers
    - » in caller-save registers
  - any additional arguments are pushed on the stack

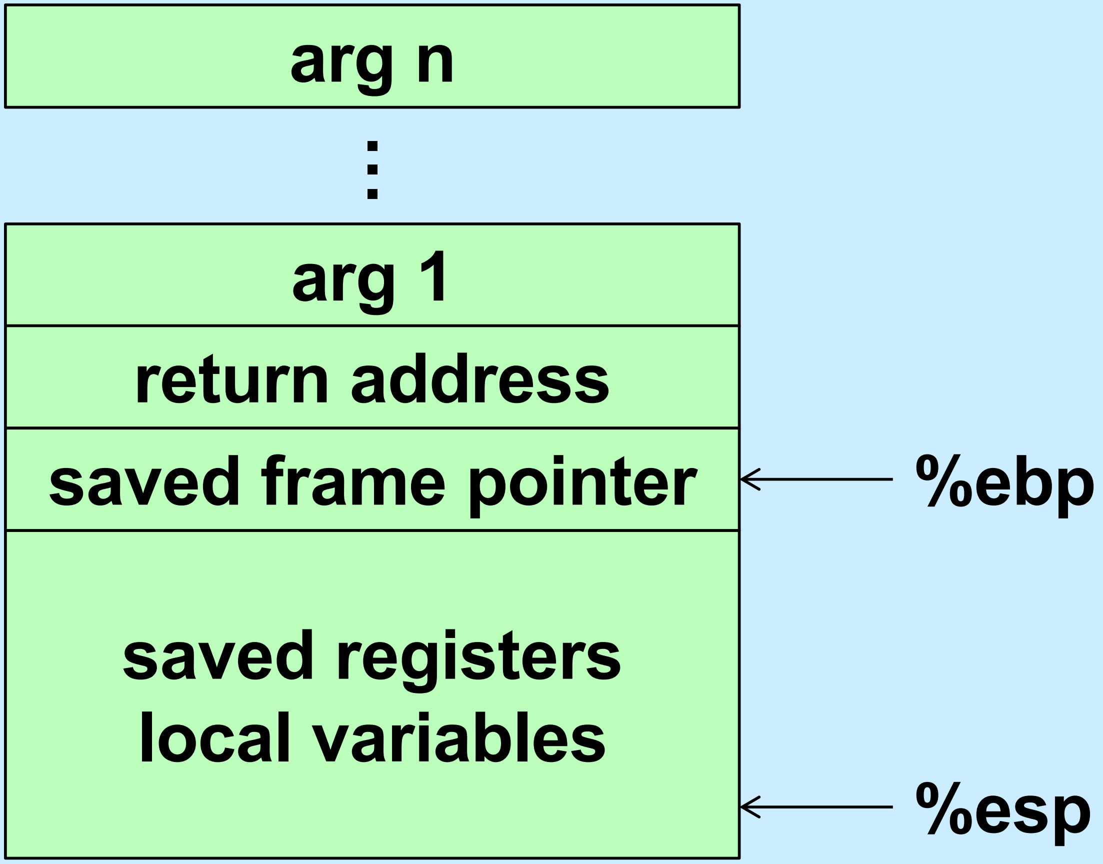
Why Bother with a Base Pointer?

- It (%rbp) points to the beginning of the stack frame
  - making it easy for people to figure out where things are in the frame
  - but people don’t execute the code ...
- The stack pointer always points somewhere within the stack frame
  - it moves about, but the compiler knows where it is pointing
    - » a local variable might be at 8(%rsp) for one instruction, but at 16(%rsp) for a subsequent one
    - » tough for people, but easy for the compiler
- Thus the base pointer is superfluous
  - it can be used as a general-purpose register

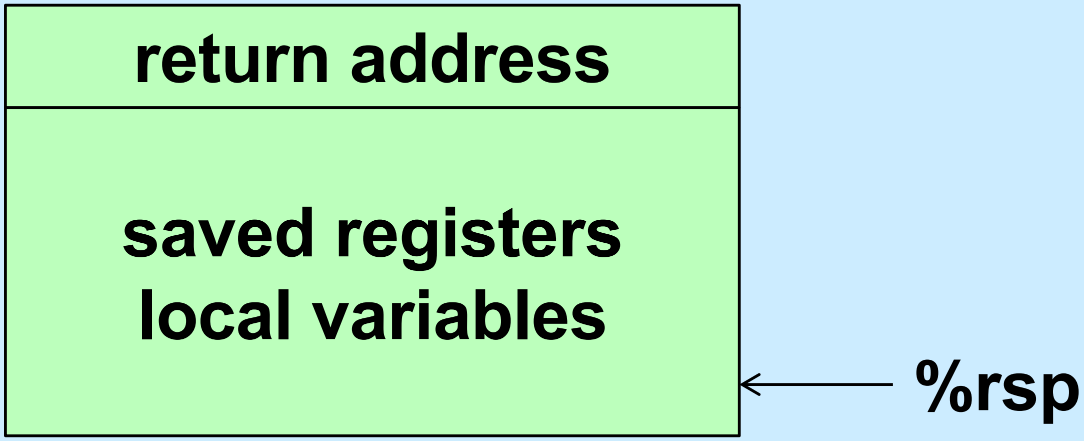
x86-64 General-Purpose Registers: Updated Usage Conventions

`%rax`	Return value	`%r8`	Argument #5
`%rbx`	Callee saved	`%r9`	Argument #6
`%rcx`	Argument #4	`%r10`	Caller saved
`%rdx`	Argument #3	`%r11`	Caller Saved
`%rsi`	Argument #2	`%r12`	Callee saved
`%rdi`	Argument #1	`%r13`	Callee saved
`%rsp`	Stack pointer	`%r14`	Callee saved
`%rbp`	Callee saved	`%r15`	Callee saved

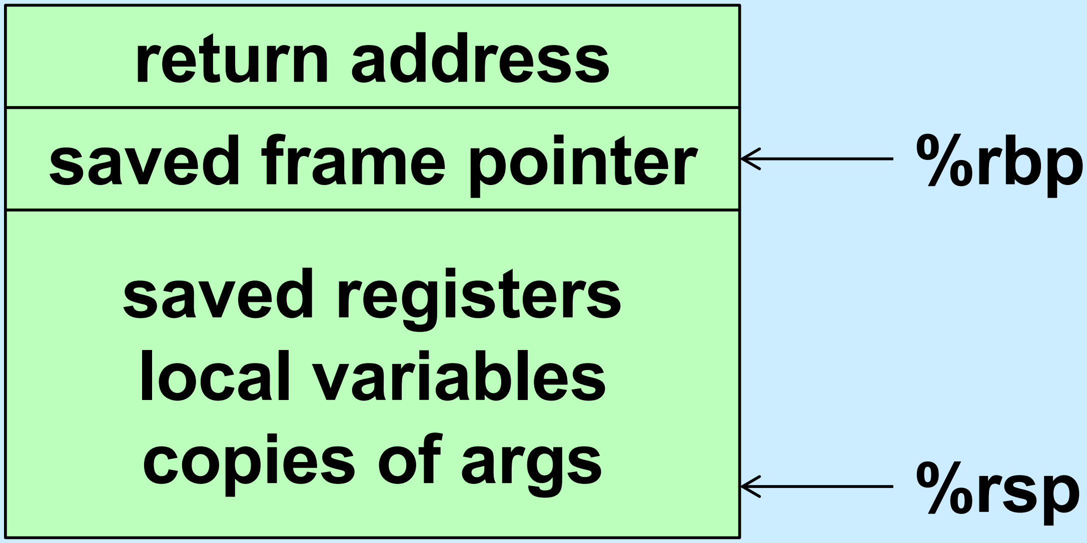
The IA32 Stack Frame



The x86-64 Stack Frame



The -O0 x86-64 Stack Frame (Traps and Buffer)



Summary

- What's pushed on the stack
  - return address
  - saved registers
    - caller-saved by the caller
    - callee-saved by the callee
  - local variables
  - function parameters
    - those too large to be in registers (structs)
    - those beyond the six that we have registers for
  - large return values (structs)
    - caller allocates space on stack
    - callee copies return value to that space

Quiz 2

Suppose function A is compiled using the convention that %rbp is used as the base pointer, pointing to the beginning of the stack frame. Function B is compiled using the convention that there’s no need for a base pointer. Will there be any problems if A calls B or if B calls A?

a) Neither case will work

b) A calling B works, but B calling A doesn’t

c) B calling A works, but A calling B doesn’t

d) Both work