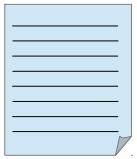
M2 – Instruction Set Architecture

Module Outline

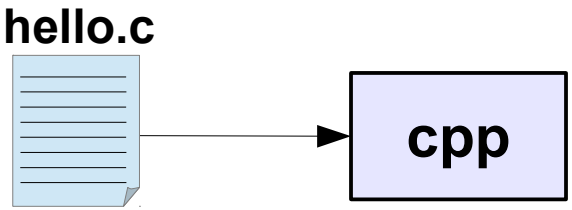
- Addressing modes. Instruction classes.
- MIPS-I ISA.
- Translating and starting a program.
- High level languages, Assembly languages and object code.
- Subroutine and subroutine call. Use of stack for handling subroutine call and return.

Steps in gcc

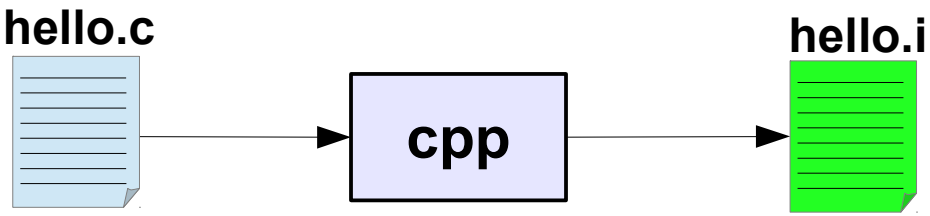
hello.c



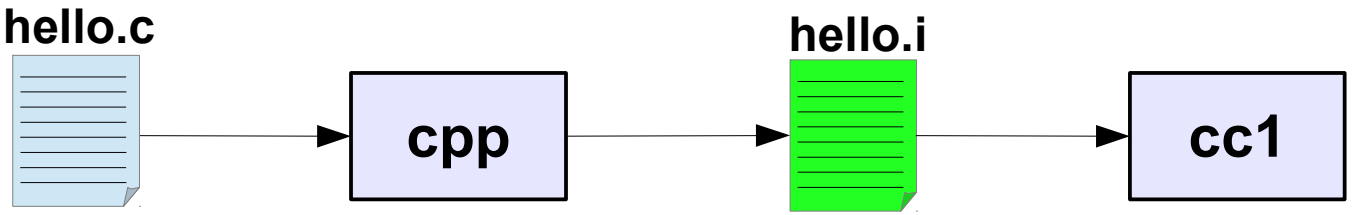
Steps in gcc



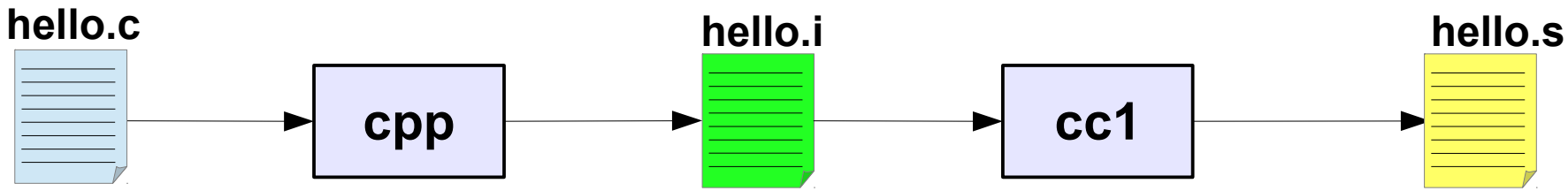
Steps in gcc



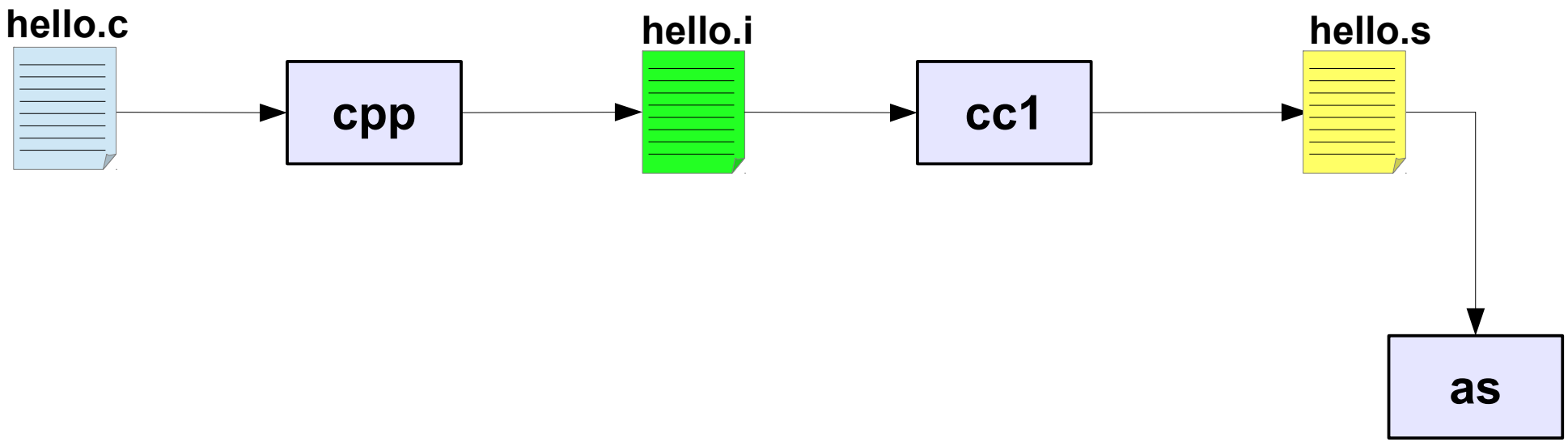
Steps in gcc



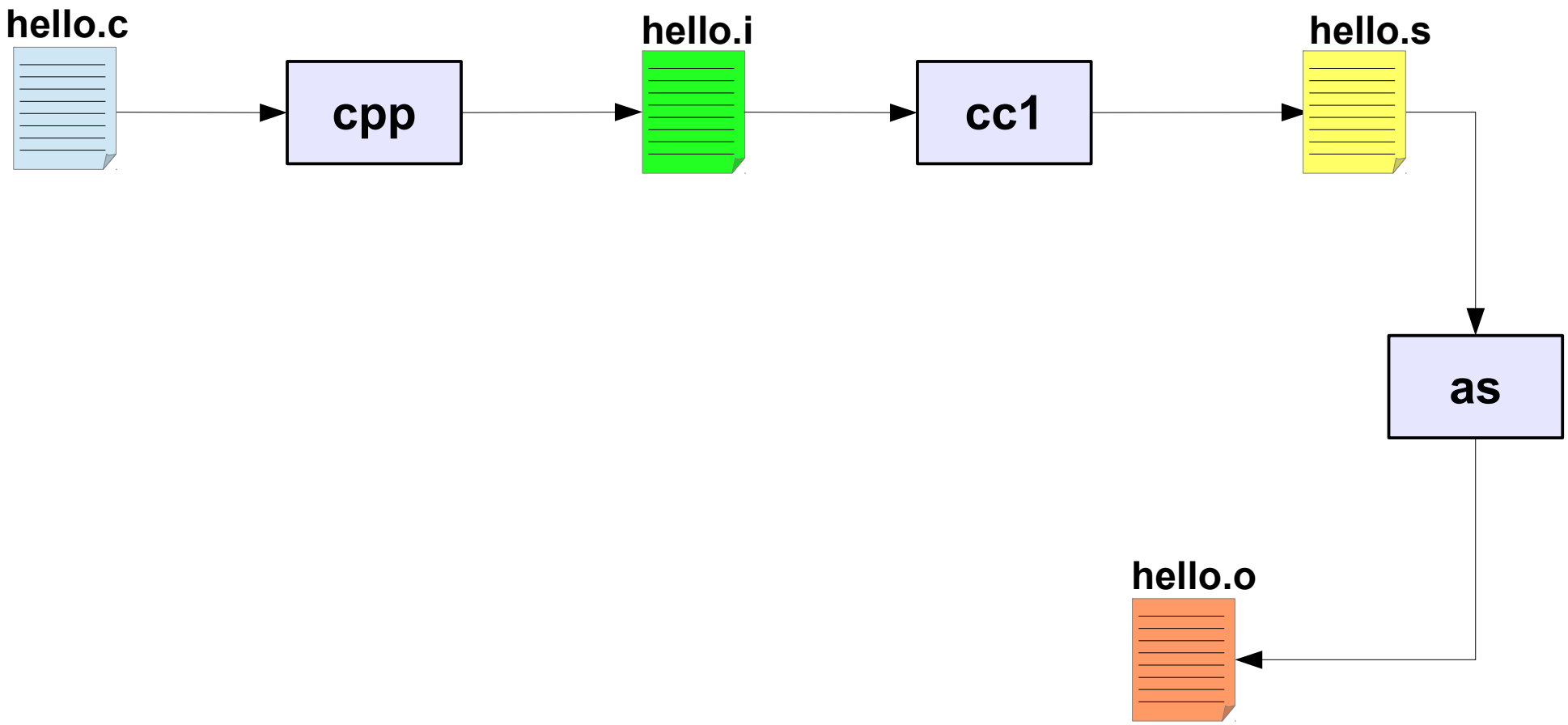
Steps in gcc



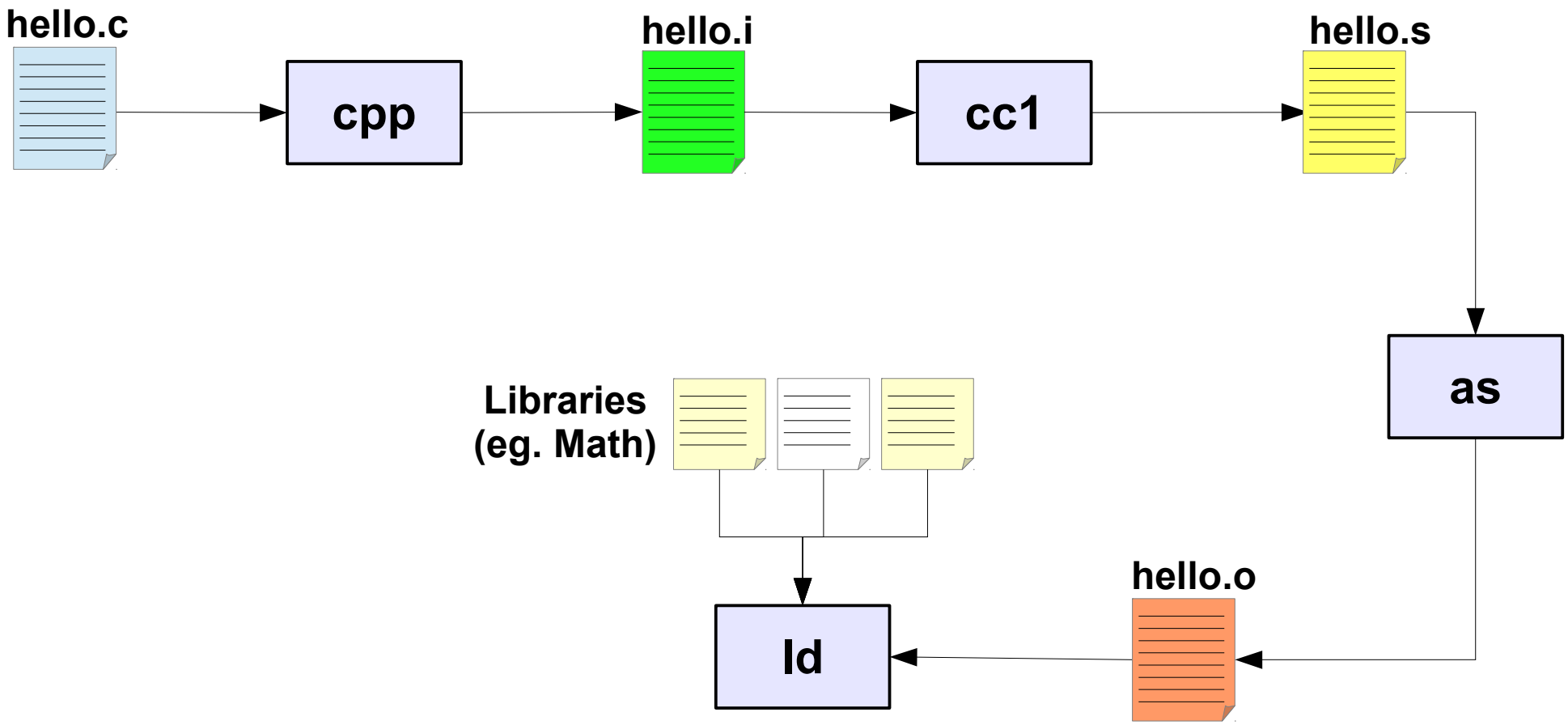
Steps in gcc



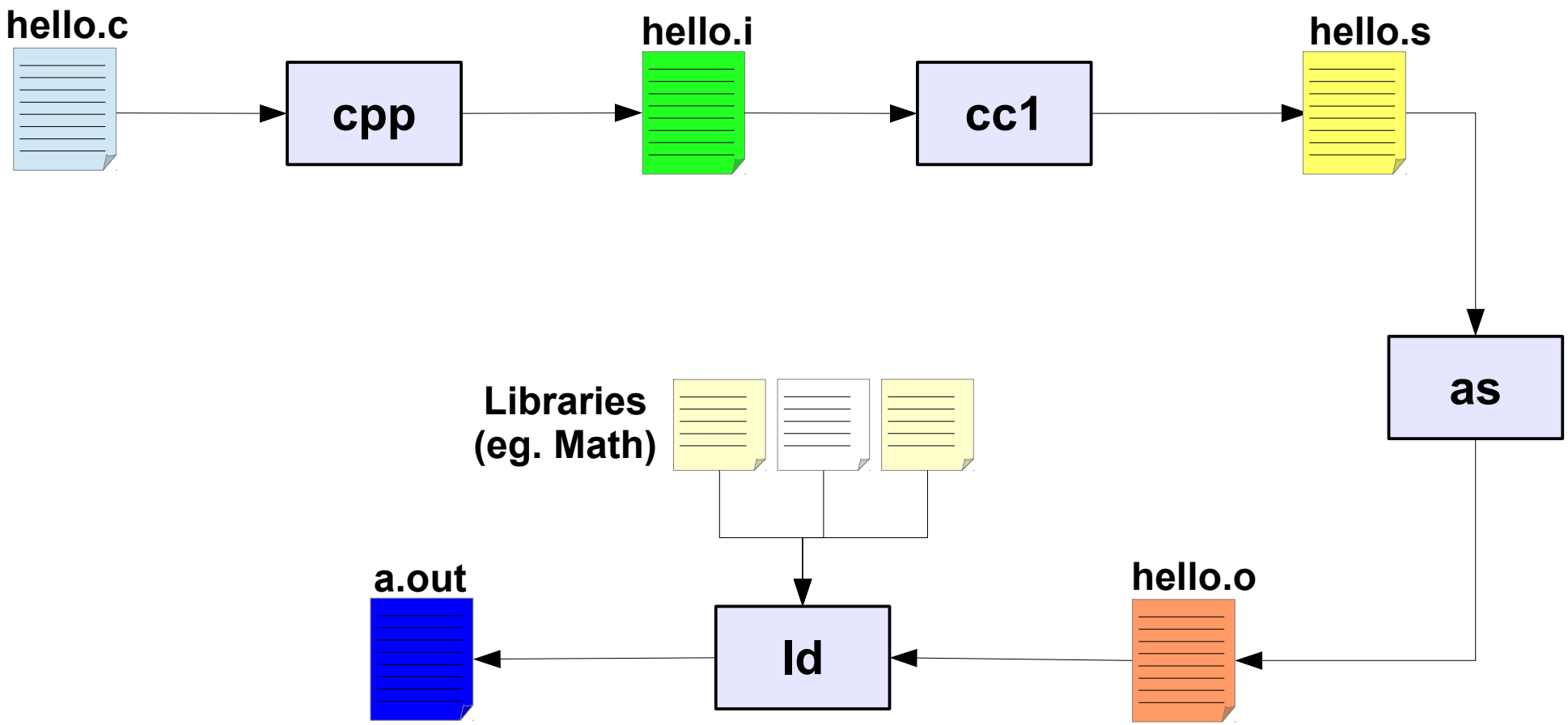
Steps in gcc



Steps in gcc



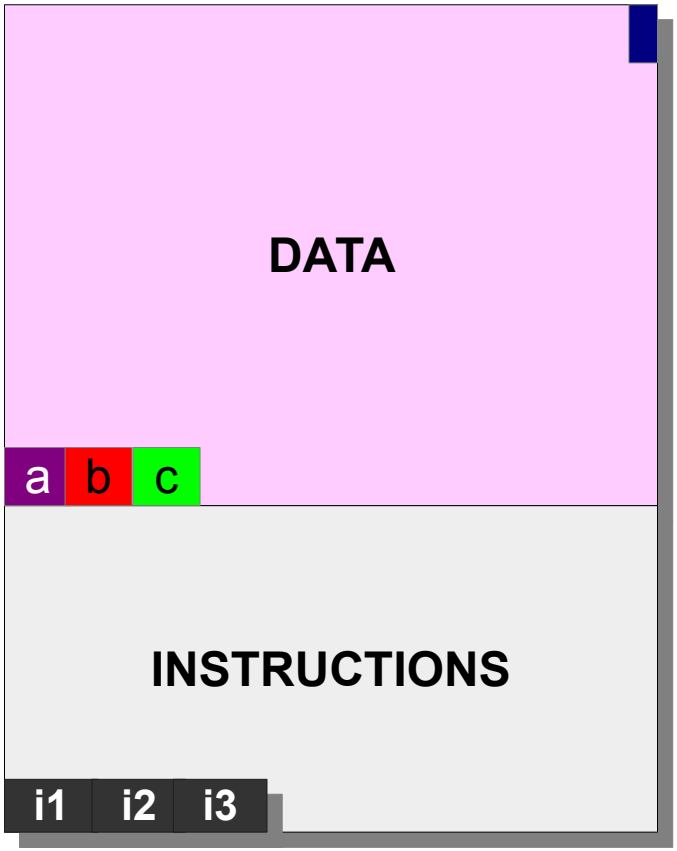
Steps in gcc


hello.c
cpp
hello.i
cc1
hello.s
as
Libraries
(eg. Math)
a.out
ld
hello.o

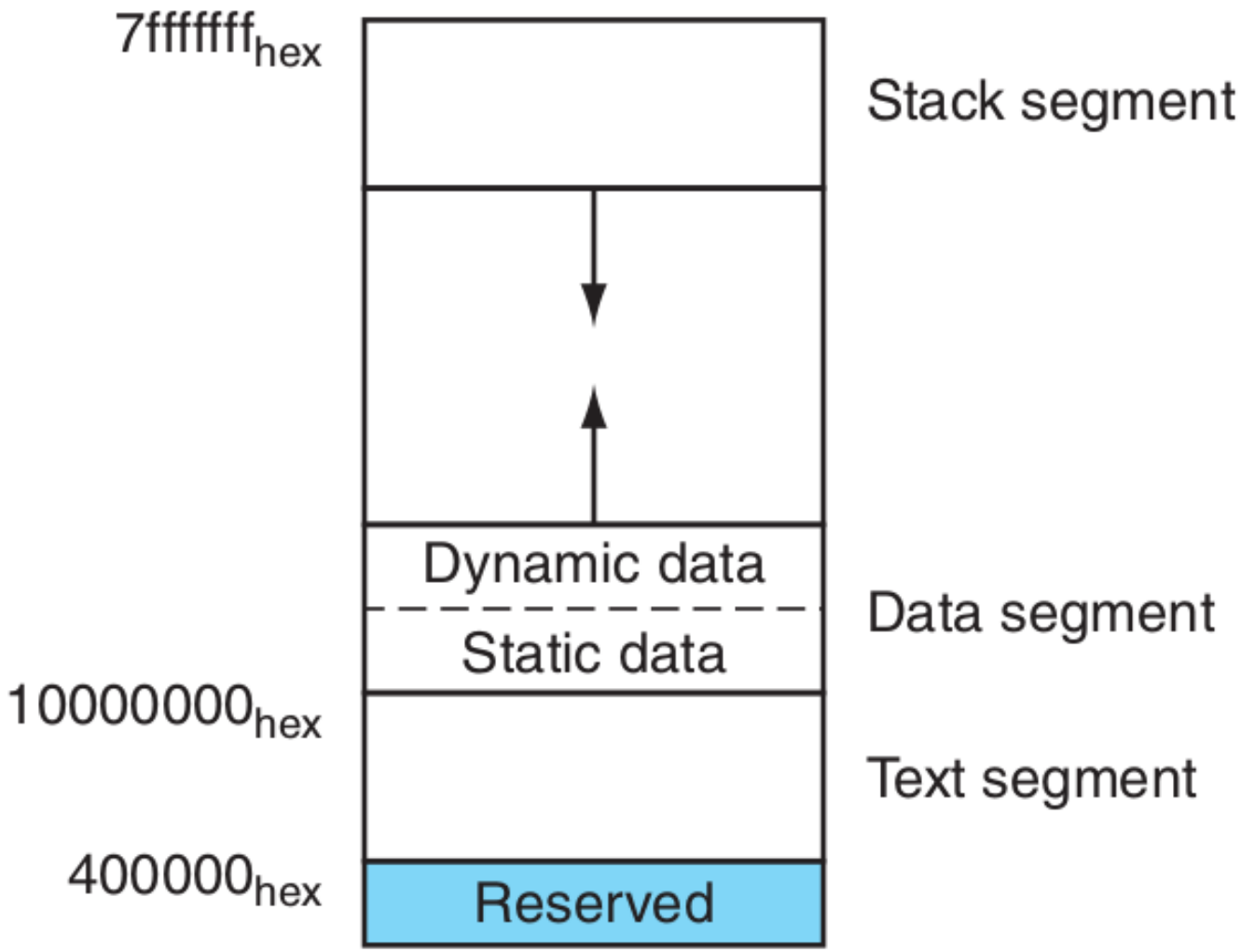
Steps in gcc

Step	Output	FileType	Remarks
C Preprocessor	hello.i	C source text	#include, #define, ...
C Compiler	hello.s	Assembler source text	Individual modules. Labels. Undefined global references.
Assembler	hello.o	Object code	
Linkage Editor	a.out	Executable	Global references resolved

Memory Layout of a Program in Execution



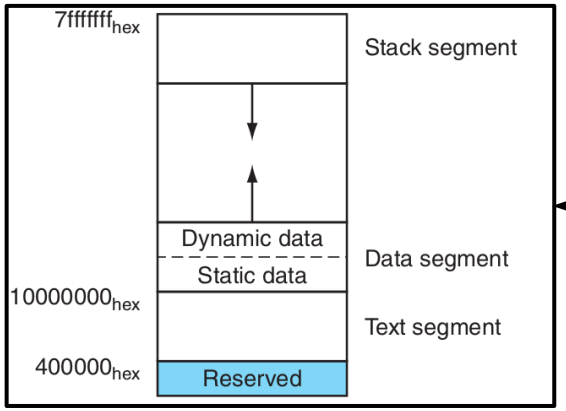
Memory Layout of a Program in Execution

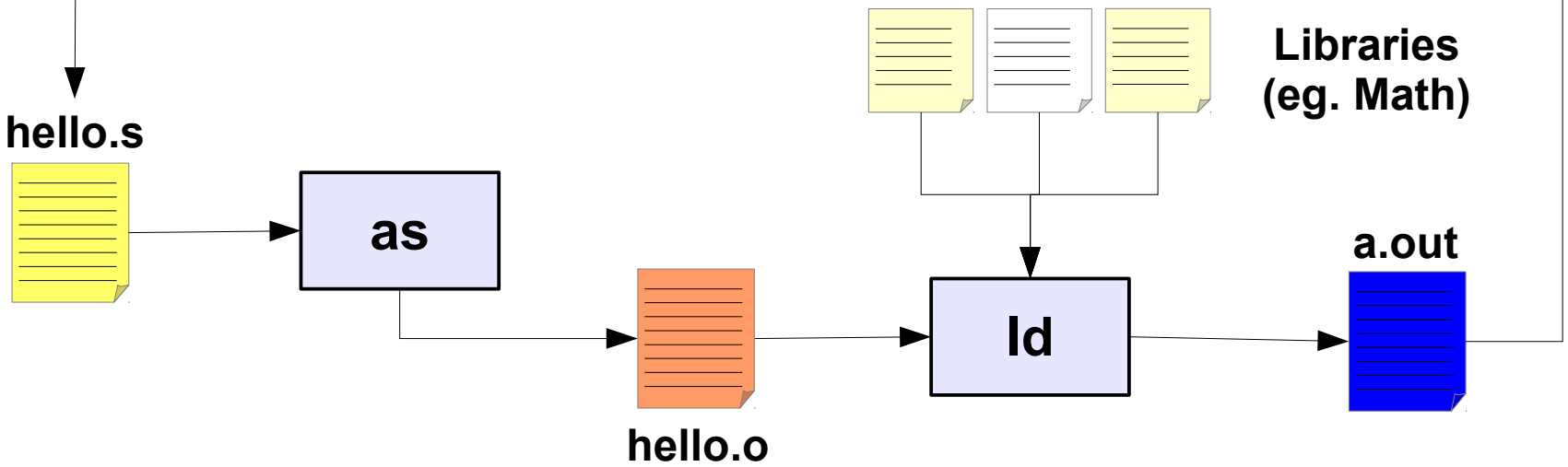


Program to Memory Layout

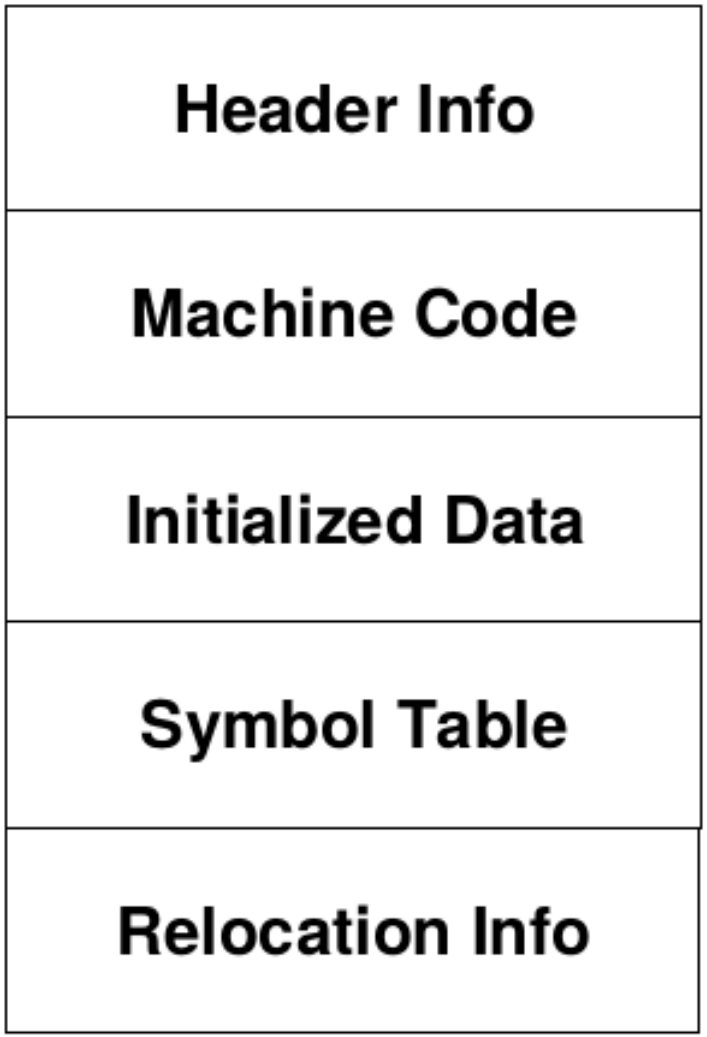
```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```





Object File Format (hello.o)



What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra

        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

Encodes easily!

What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

Assembler Directives

What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

Text segment begins here.
Align at word boundary.
main is a global “label”. Can be referred from other modules.

Assembler Directives

What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

Text segment begins here.
Align at word boundary.
main is a global “label”. Can be referred from other modules.

Assembler Directives

Data segment begins here.
Align at byte boundary.

What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

Labels

What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

Labels

Assembler Actions

```
ble     $t0, 100, loop
la      $a0, str
```

- How to encode?
- What information is required?

Assembler Actions

```
ble     $t0, 100, loop
la      $a0, str
```

- How to encode?
- Information required:
  - Where to branch? Where is “loop”?
  - Where is “str”?

Assembler Actions

- Labels – `main, loop, str` are names for memory locations.

Assembler Actions

- Labels – `main, loop, str` are names for memory locations.
  - main: first instruction in the text segment
  - loop: 6th instruction in the text segment
  - str: starts from the first byte in the data segment
  - printf: undefined

Assembler Actions

- Labels – `main, loop, str` are names for memory locations.
  - main: first instruction in the text segment
  - loop: 6th instruction in the text segment
  - str: starts from the first byte in the data segment
  - printf: undefined

main	0
loop	20
str	800
printf	??

Assembler Actions

- Labels – `main, loop, str` are names for memory locations.
  - main: first instruction in the text segment
  - loop: 6th instruction in the text segment
  - str: starts from the first byte in the data segment
  - printf: undefined
- Symbol Table

main	0
loop	20
str	800
printf	??

What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

What does the Assembler do?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra


        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

Library Function Call

Unresolved references

- Defined in other object files
  - Library functions, external calls to other modules

Unresolved references

- Defined in other object files
  - Library functions, external calls to other modules
- Will exist in the symbol table entries of those obj files

Unresolved references

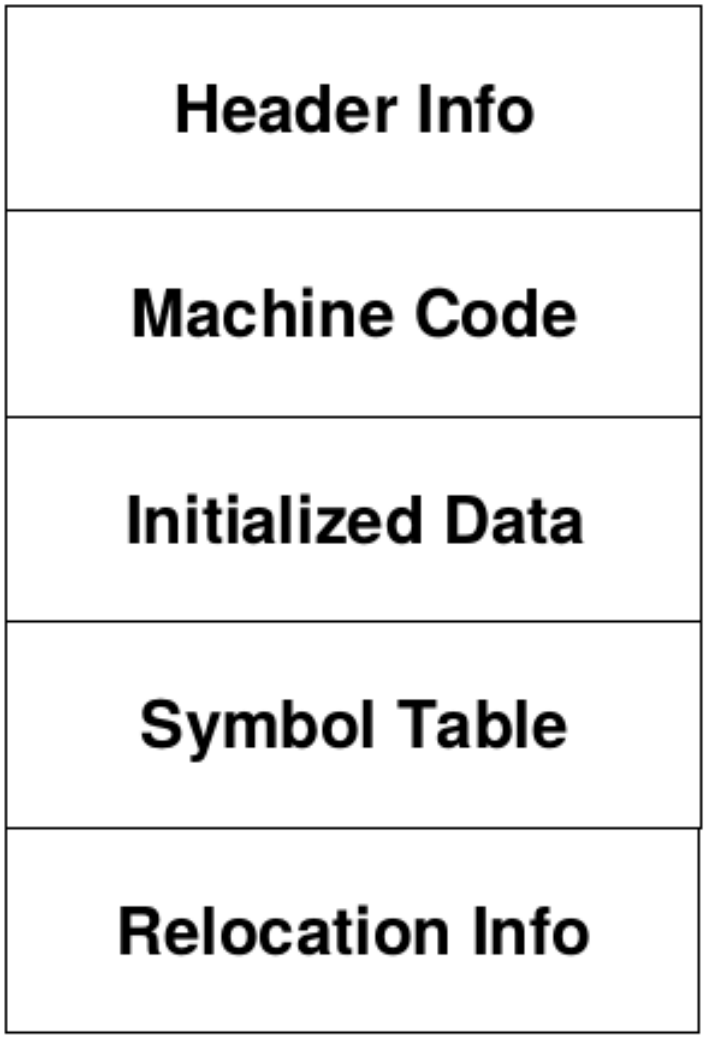
- Defined in other object files
  - Library functions, external calls to other modules
- Will exist in the symbol table entries of those obj files
- Make a list of unresolved references in the present object file
  - Relocation information

Unresolved references

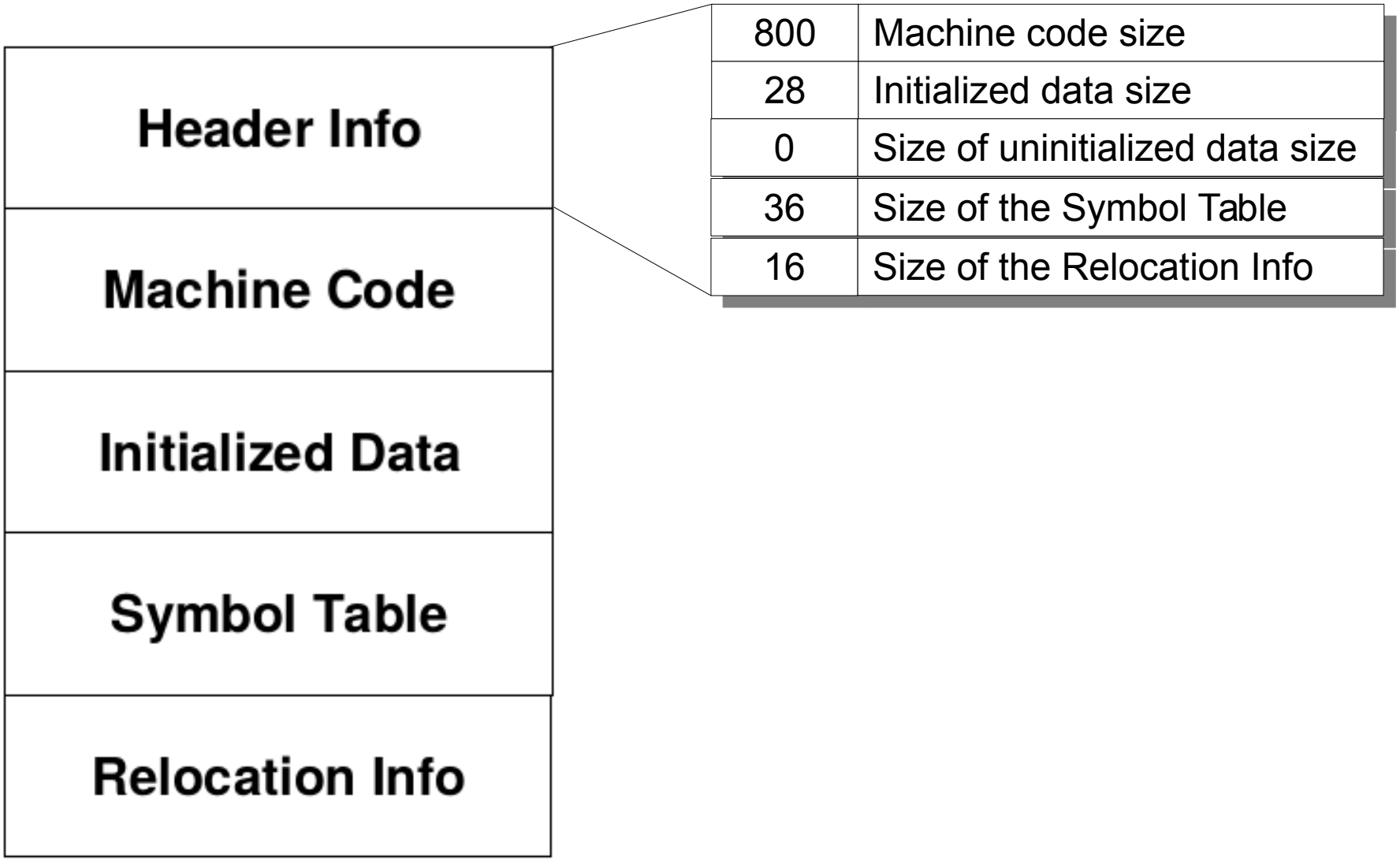
- Defined in other object files
  - Library functions, external calls to other modules
- Will exist in the symbol table entries of those obj files
- Make a list of unresolved references in the present object file
  - Relocation information

64	call	printf	??

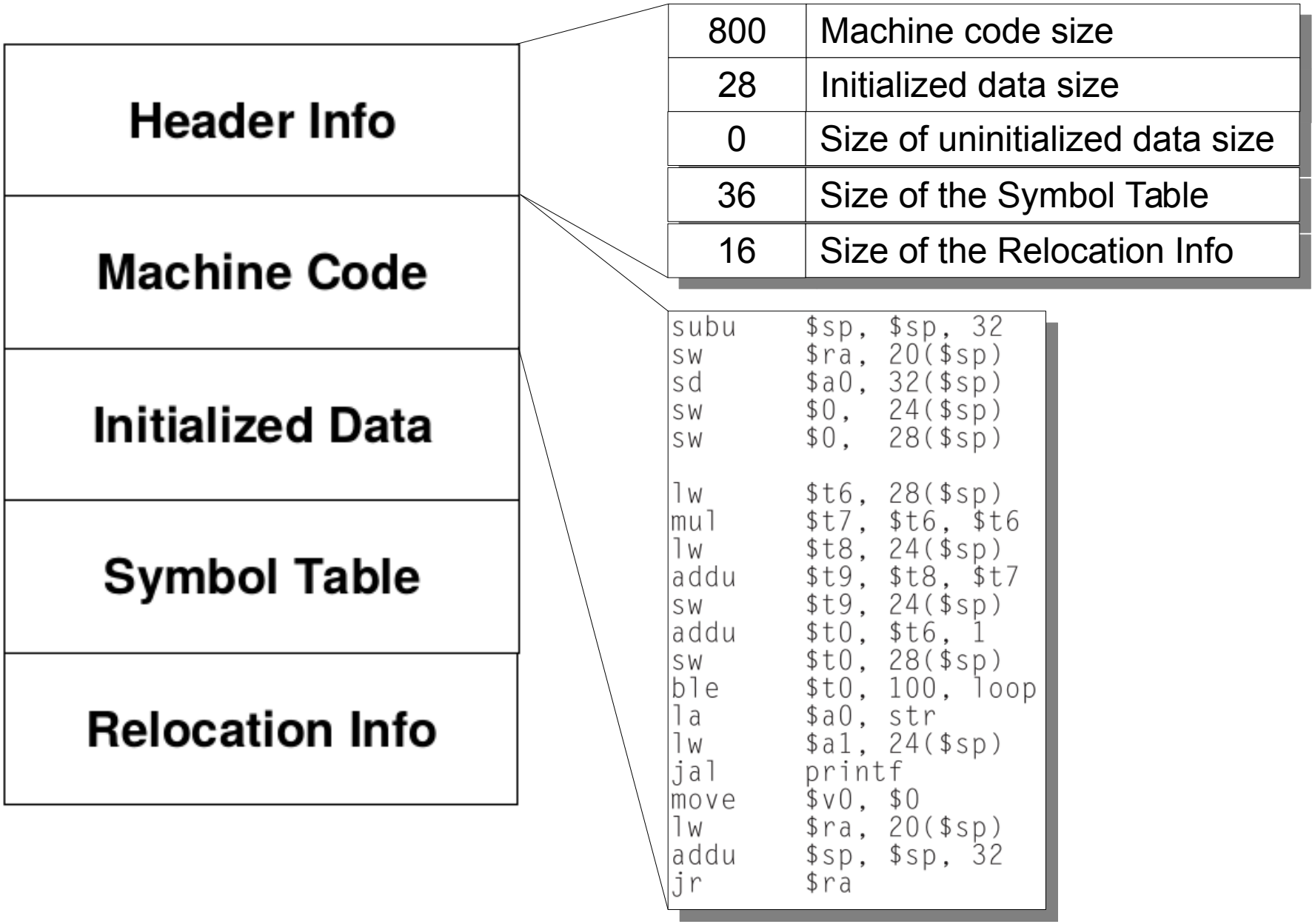
Object File Format (hello.o)



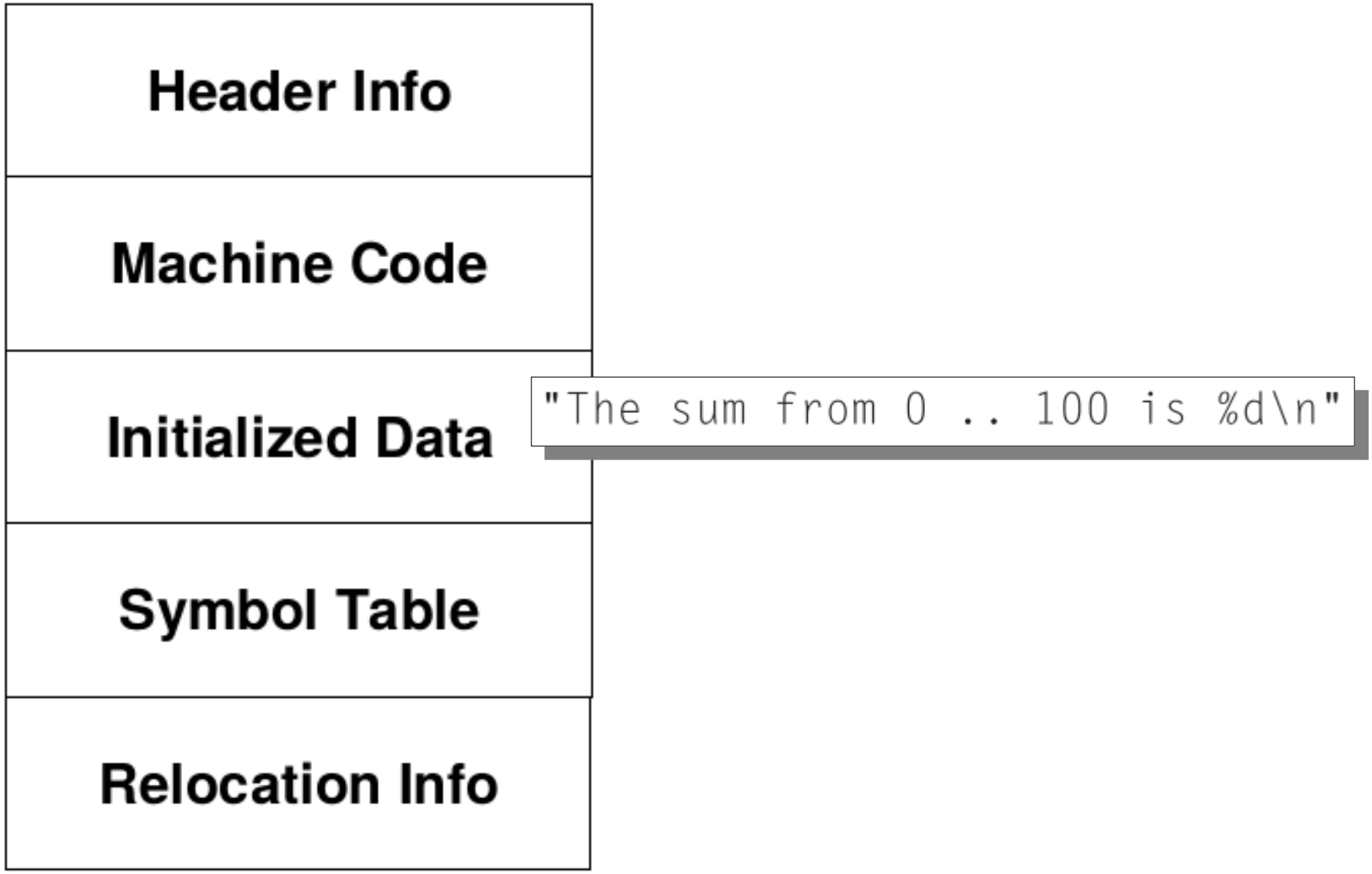
Object File Format (hello.o)



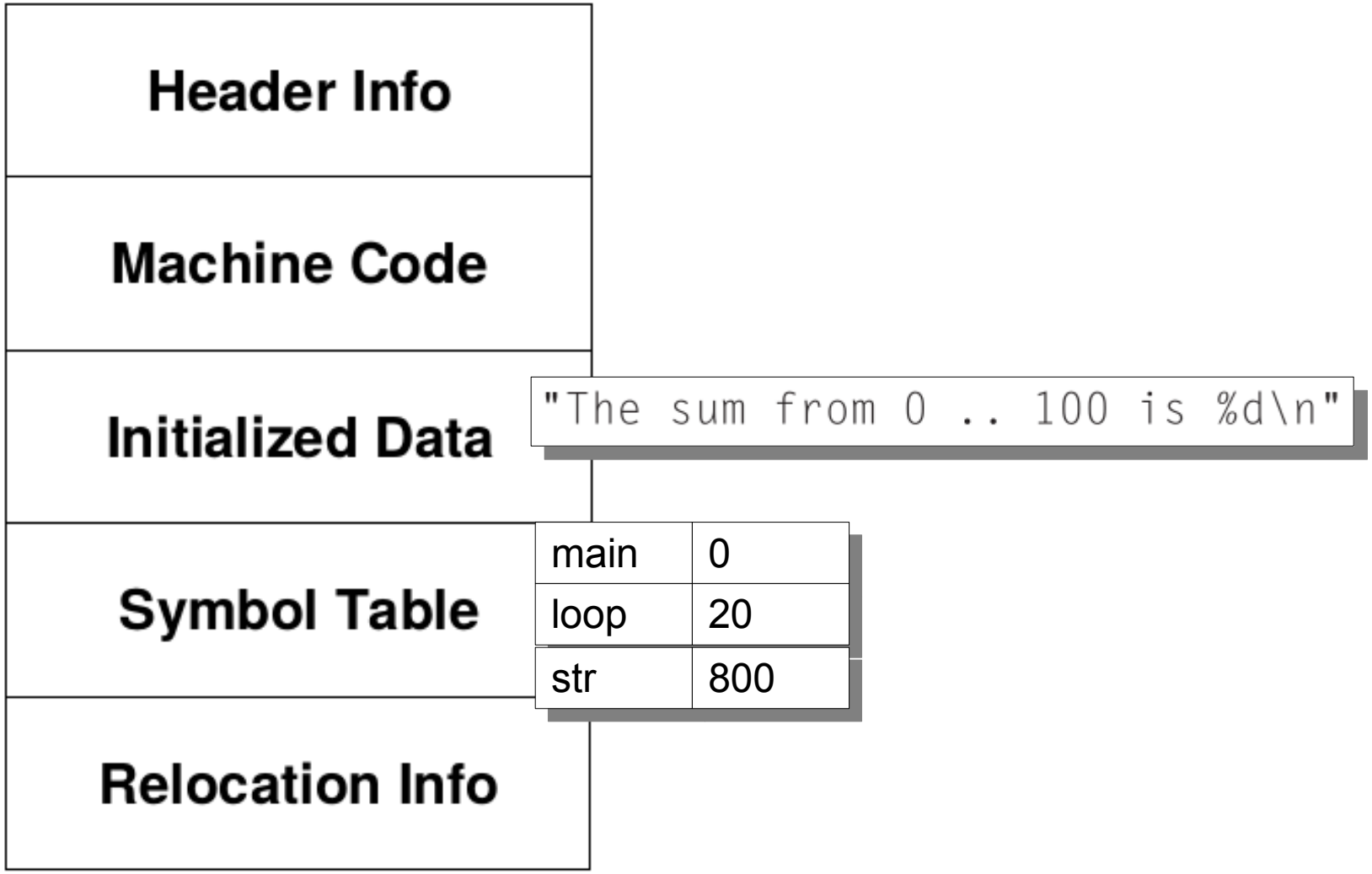
Object File Format (hello.o)



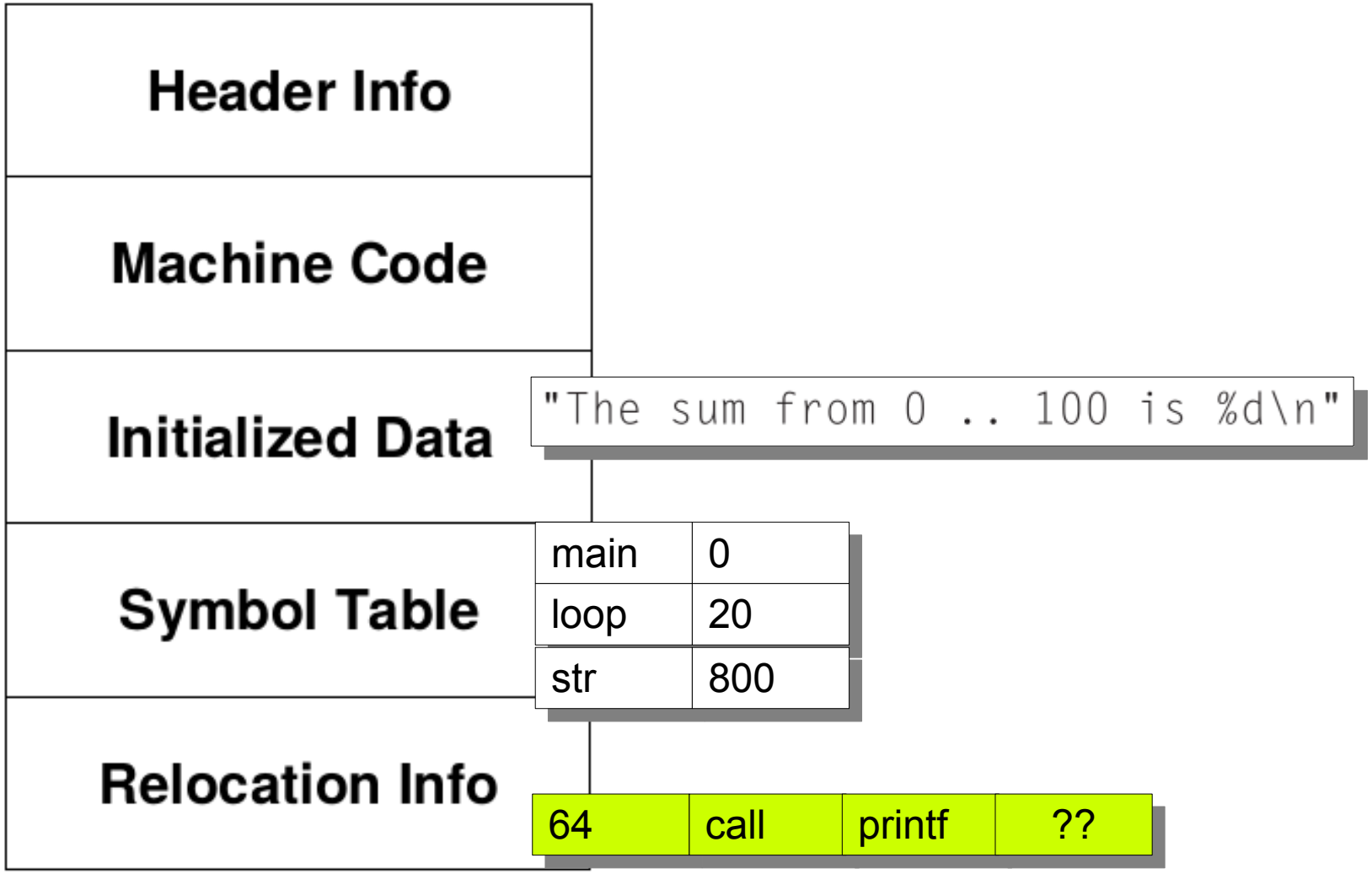
Object File Format (hello.o)



Object File Format (hello.o)



Object File Format (hello.o)



Assembler

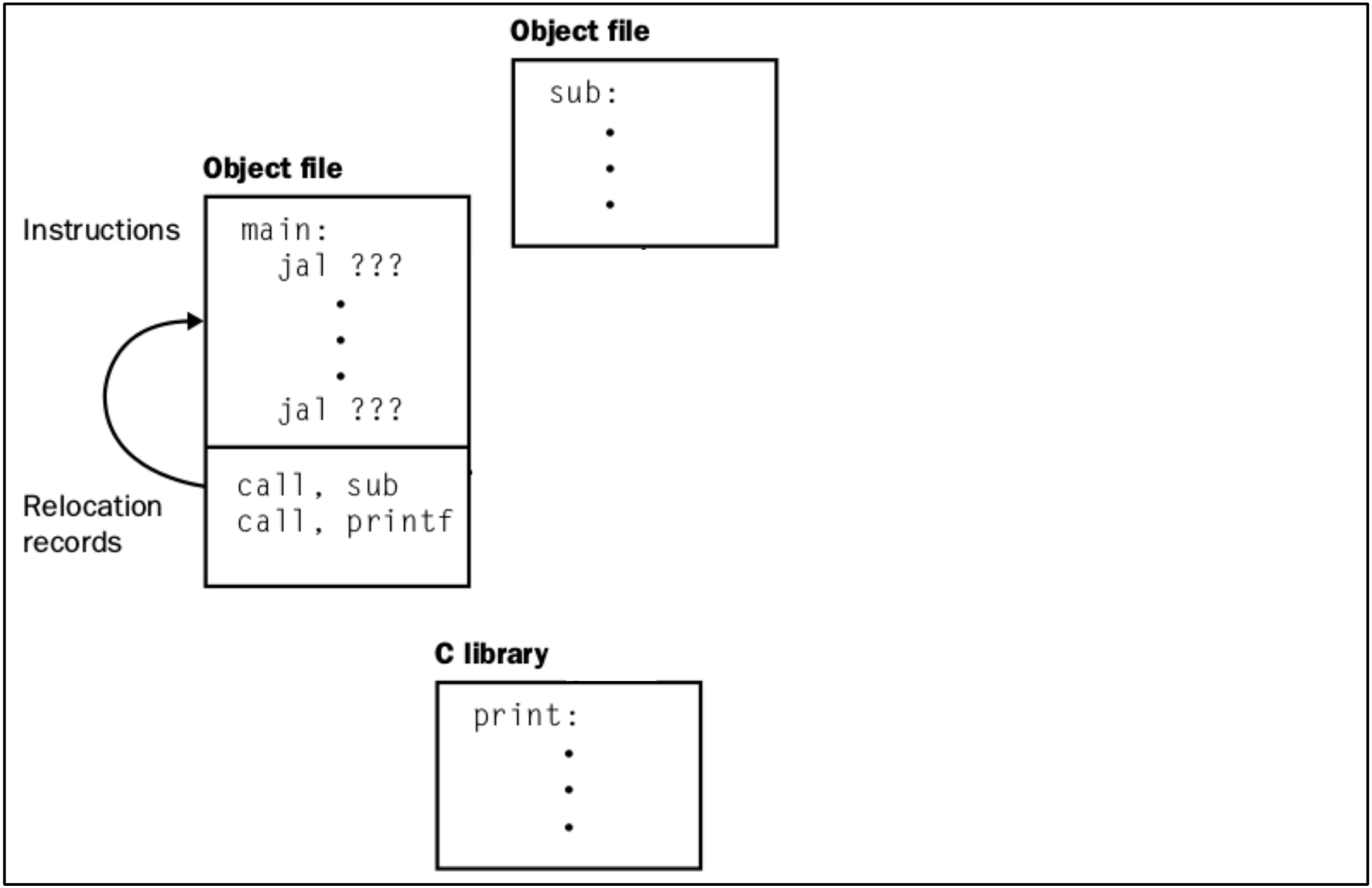
Assembler

- Pass 1: Finds instructions with labels. Records their memory locations and labels so that the relationship between symbolic names and addresses is known when instructions are translated.

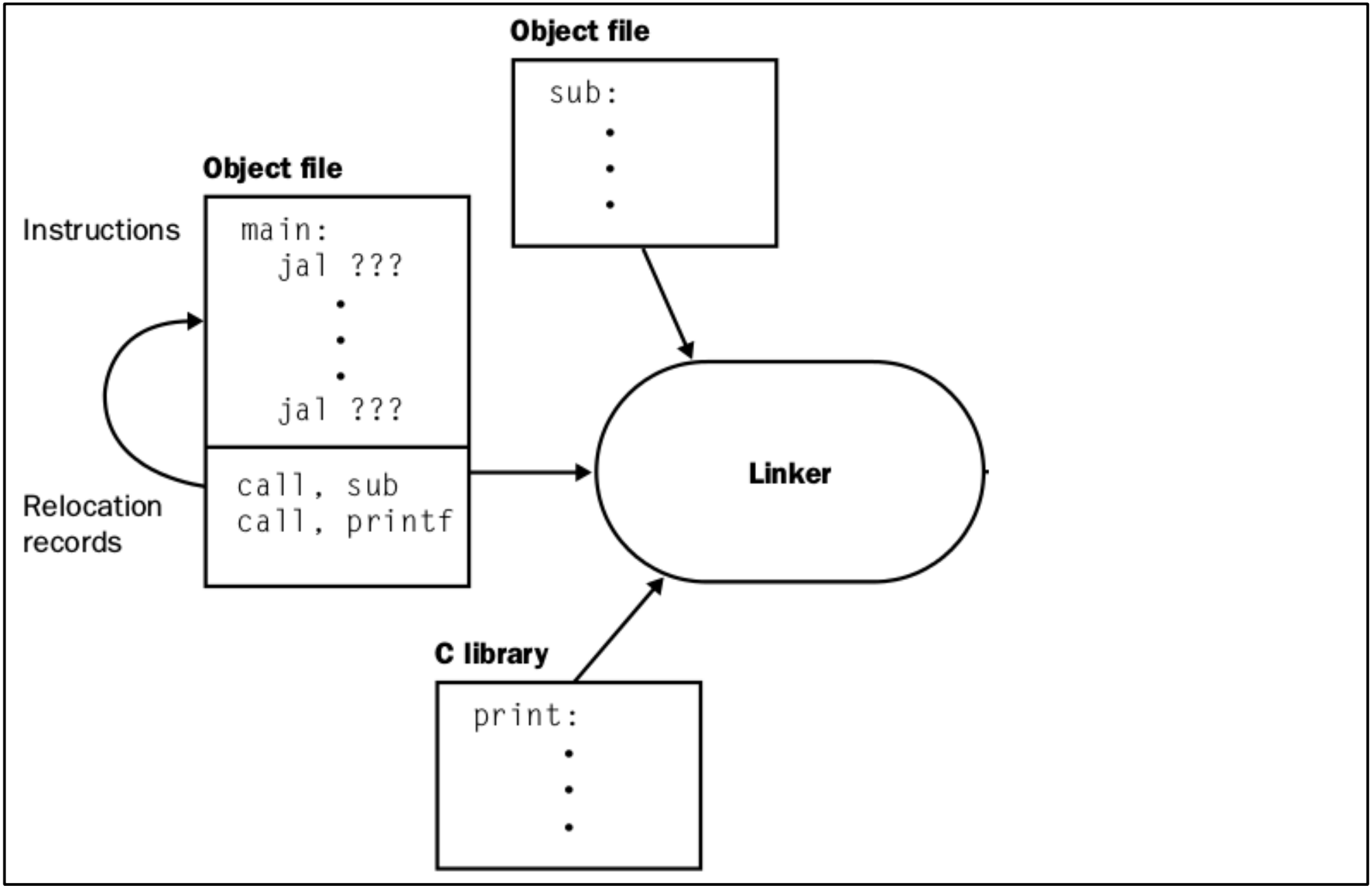
Assembler

- Pass 1: Finds instructions with labels. Records their memory locations and labels so that the relationship between symbolic names and addresses is known when instructions are translated.
- Pass 2: Translate each assembly statement by combining the numeric equivalents of opcodes, register specifiers, and labels into a legal instruction.

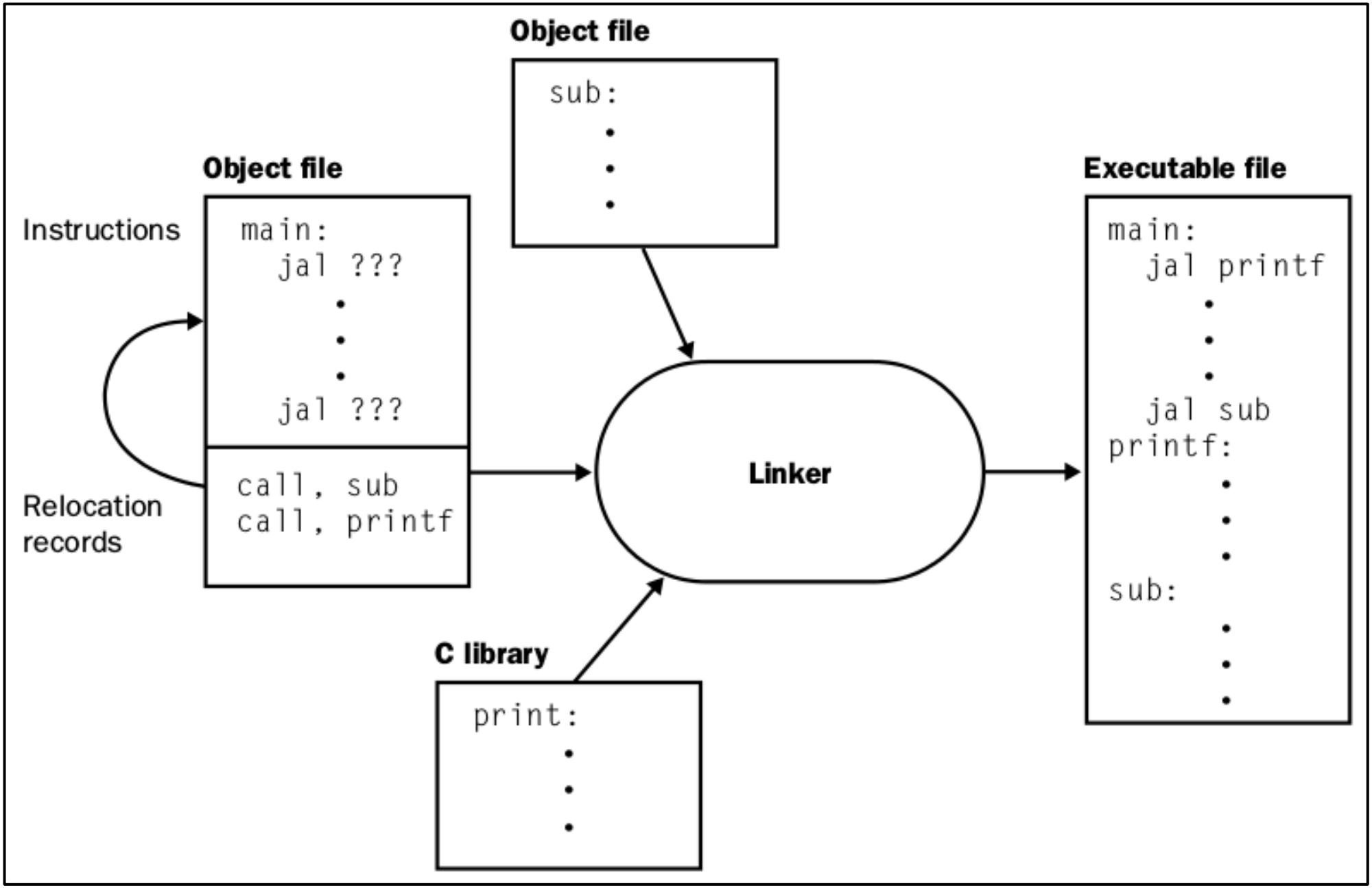
Linking Multiple Modules



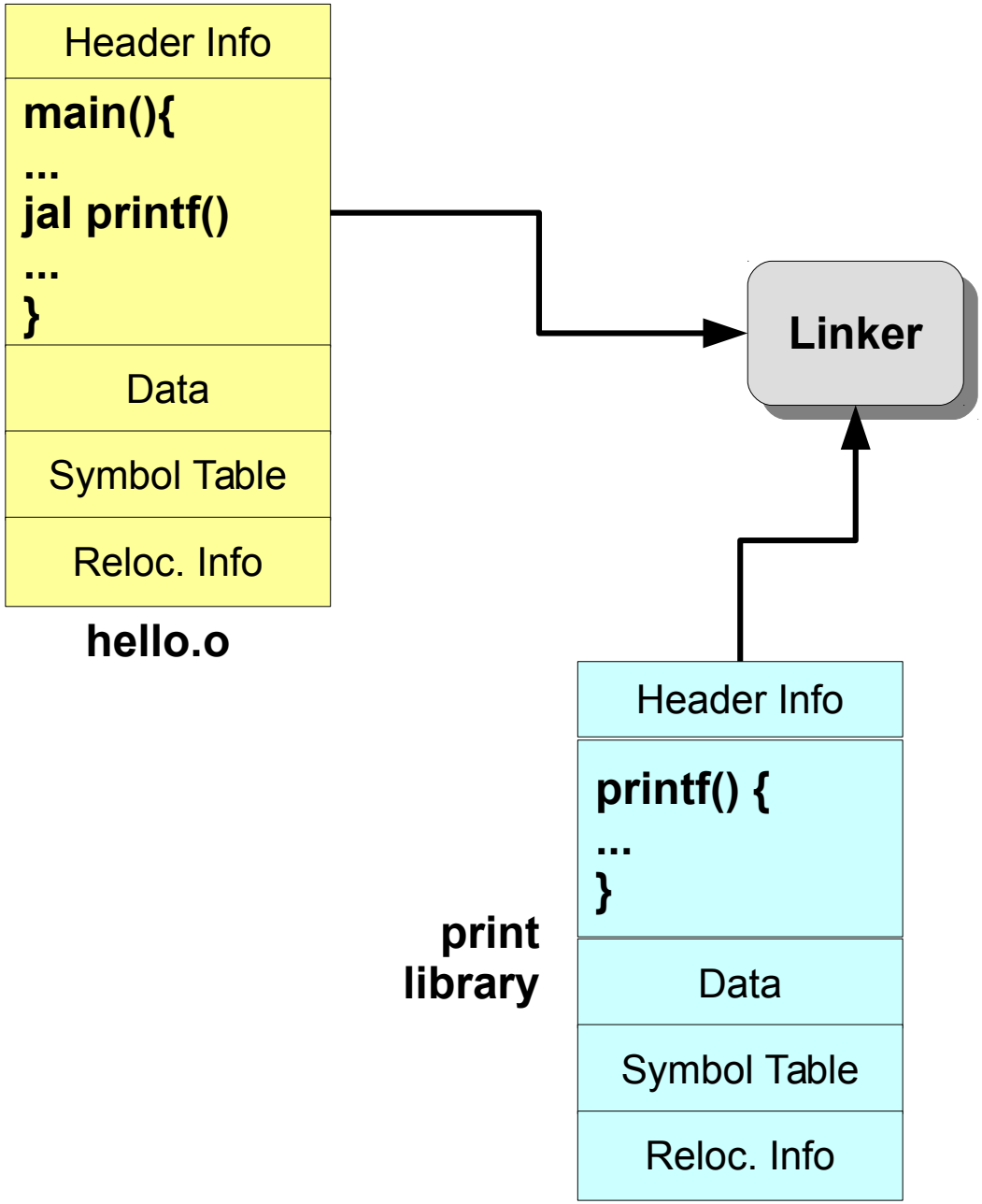
Linking Multiple Modules



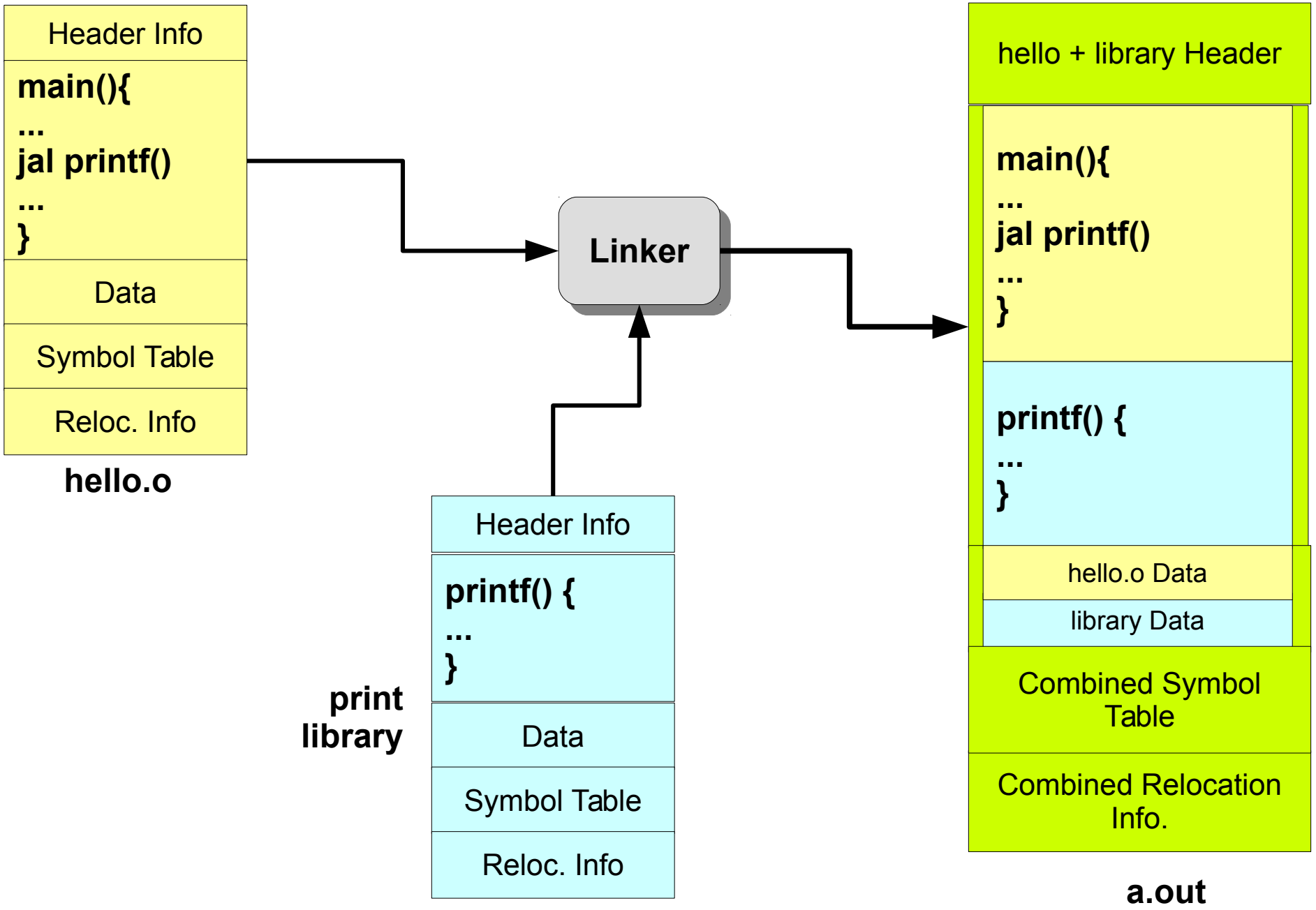
Linking Multiple Modules



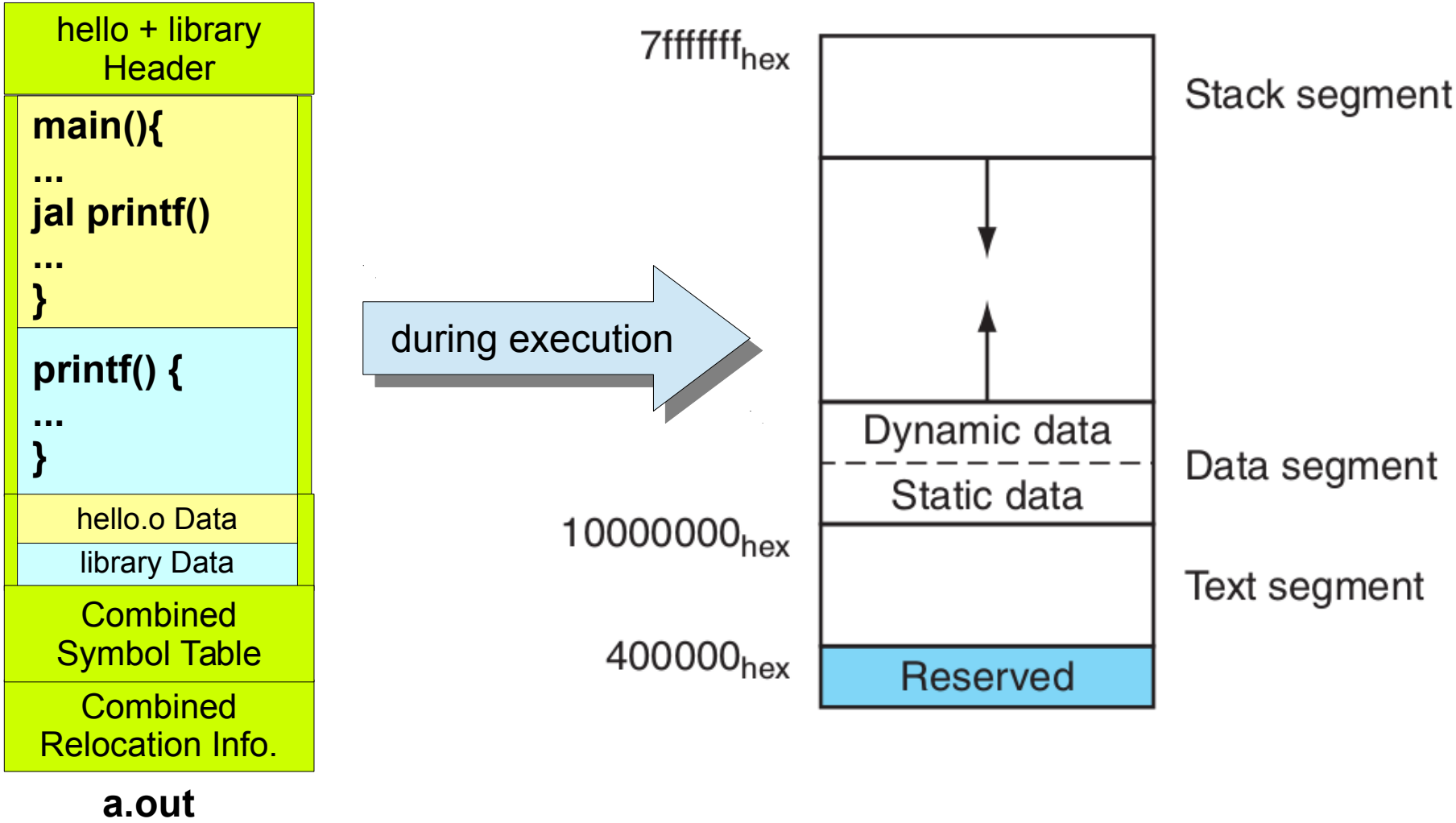
Linking Multiple Modules



Linking Multiple Modules



Memory Layout of a.out



Linking Example

- Link modules procedureA and procedureB

```
procedureA:
 ...
  la $gp, X
  lw $a0, 0($gp)
  jal B
 ...

.data
 .word X
```

```
procedureB:
 ...
  la $gp, Y
  sw $a0, 0($gp)
  jal A
 ...

.data
 .word Y
```

Linking Example – Procedure A

Object file header			
Text segment	Address	Instruction	
	0	lw $a0, 0($gp)	
	4	jal 0	
	...	...	
Data segment			
Relocation information			
Symbol table			

Linking Example – Procedure A

Object file header			
Text segment	Address	Instruction	
	0	lw $a0, 0($gp)	
	4	jal 0	
	...	...	
Data segment	0	(X)	
	...	...	
Relocation information			
Symbol table			

Linking Example – Procedure A

Object file header			
Text segment	Address	Instruction	
	0	lw $a0, 0($gp)	
	4	jal 0	
	...	...	
Data segment	0	(X)	
	...	...	
Relocation information			
Symbol table	Label	Address	
	X	–	
	B	–	

Linking Example – Procedure A

Object file header			
Text segment	Address	Instruction	
	0	lw $a0, 0($gp)	
	4	jal 0	
	...	...	
Data segment	0	(X)	
	...	...	
Relocation information	Address	Instruction type	Dependency
	0	lw	X
	4	jal	B
Symbol table	Label	Address	
	X	–	
	B	–	

Linking Example – Procedure A

Object file header			
	Name	Procedure A	
	Text size	100_{hex}	
	Data size	20_{hex}	
Text segment	Address	Instruction	
	0	`lw $a0, 0($gp)`	
	4	`jal 0`	
	...	...	
Data segment	0	(X)	
	...	...	
Relocation information	Address	Instruction type	Dependency
	0	`lw`	X
	4	`jal`	B
Symbol table	Label	Address	
	X	–	
	B	–	

Linking Example – Procedure B

Object file header			
Text segment	Address	Instruction	
	0	sw $a1, 0($gp)	
	4	jal 0	
	...	...	
Data segment			
Relocation information	Address	Instruction type	Dependency
	0	sw	Y
	4	jal	A
Symbol table			

Linking Example – Procedure B

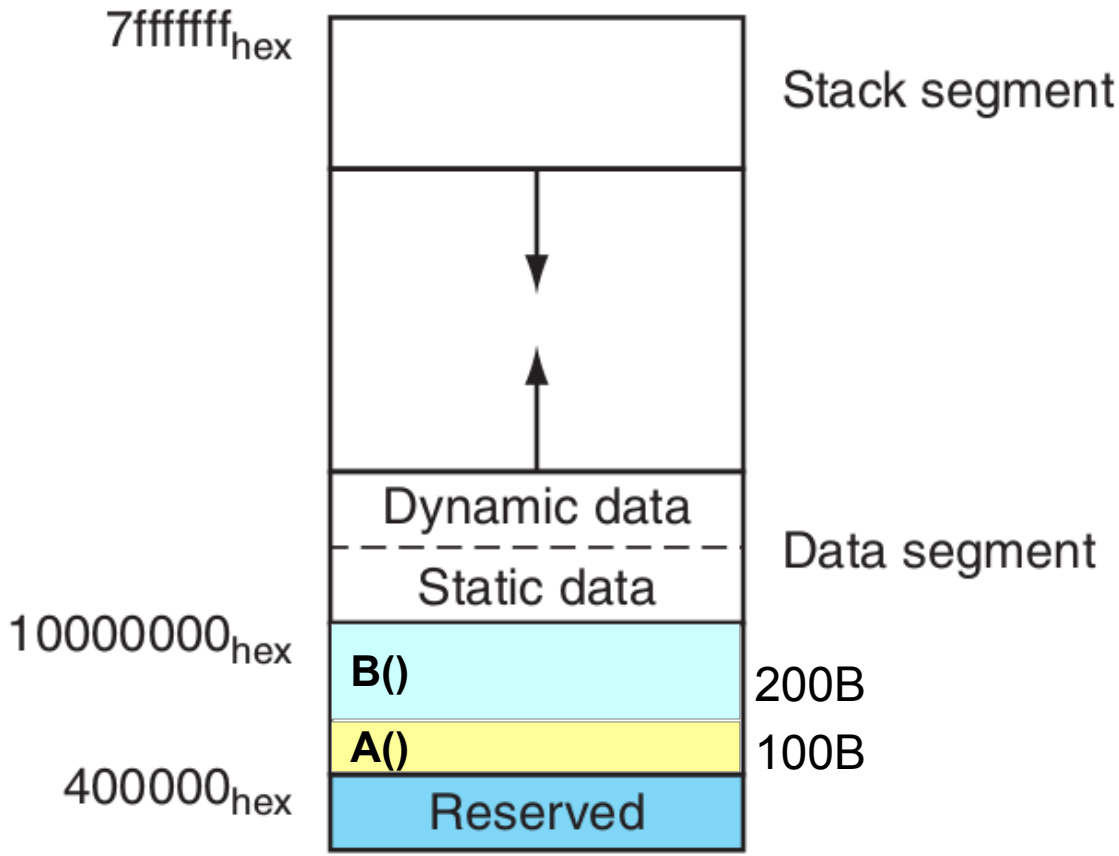
Object file header			
Text segment	Address	Instruction	
	0	sw $a1, 0($gp)	
	4	jal 0	
	...	...	
Data segment	0	(Y)	
	...	...	
Relocation information	Address	Instruction type	Dependency
	0	sw	Y
	4	jal	A
Symbol table	Label	Address	
	Y	–	
	A	–	

Linking Example – Procedure B

Object file header			
	Name	Procedure B	
	Text size	200_{hex}	
	Data size	30_{hex}	
Text segment	Address	Instruction	
	0	sw $a1, 0($gp)	
	4	jal 0	
	...	...	
Data segment	0	(Y)	
	...	...	
Relocation information	Address	Instruction type	Dependency
	0	sw	Y
	4	jal	A
Symbol table	Label	Address	
	Y	–	
	A	–	

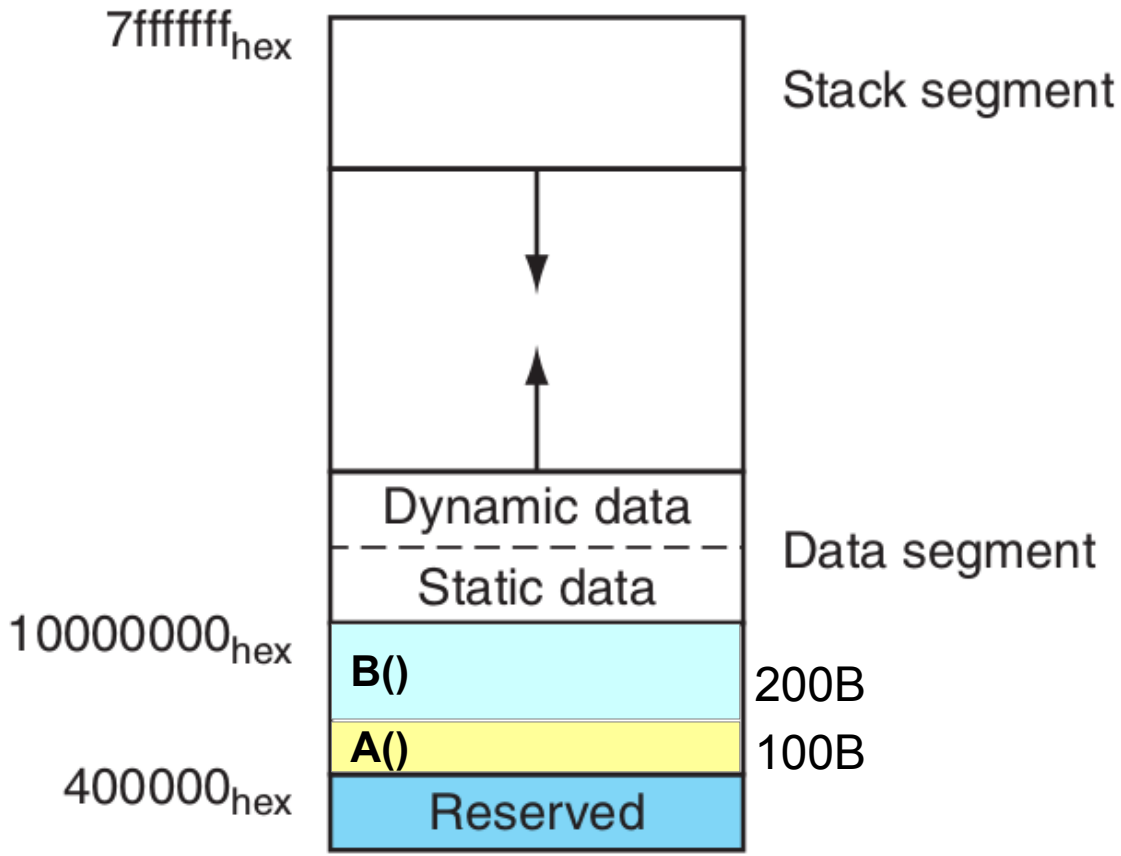
Linking Example

- Address of Procedure A?
- Address of Procedure B?



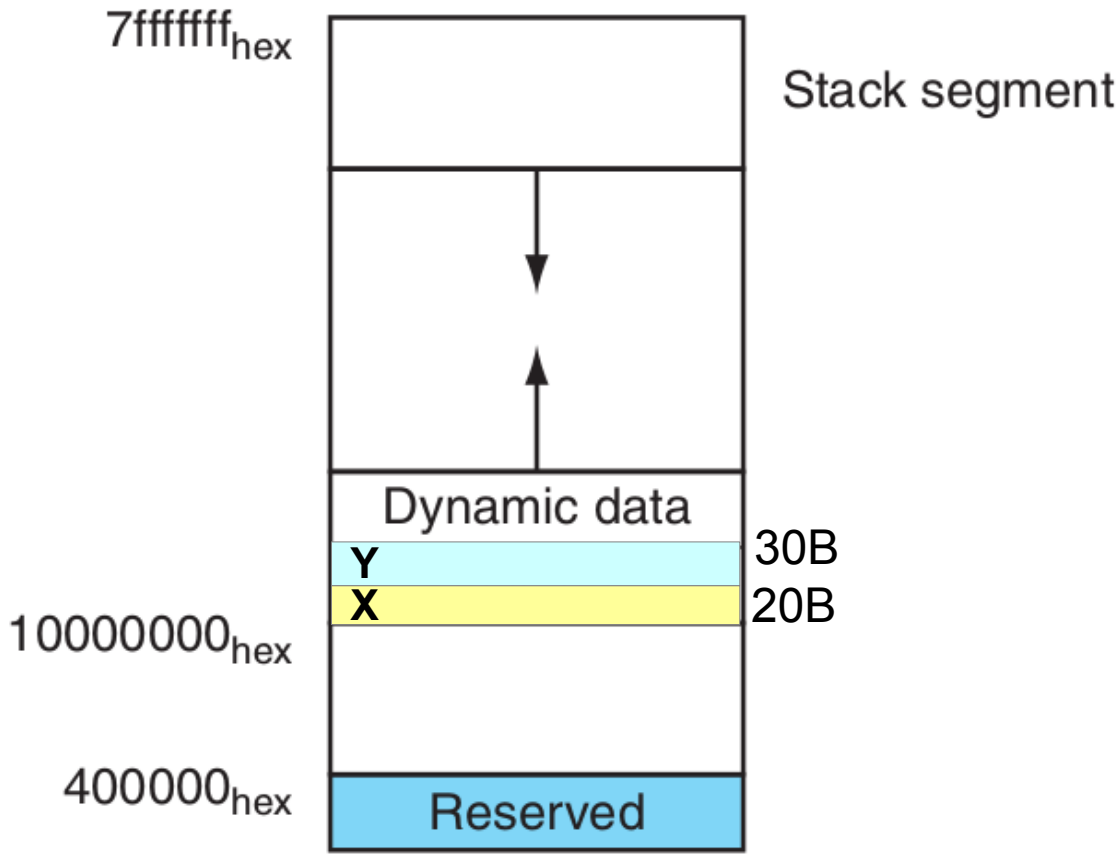
Linking Example

- Procedure A starts at 0x400000
- Procedure B starts at 0x400100



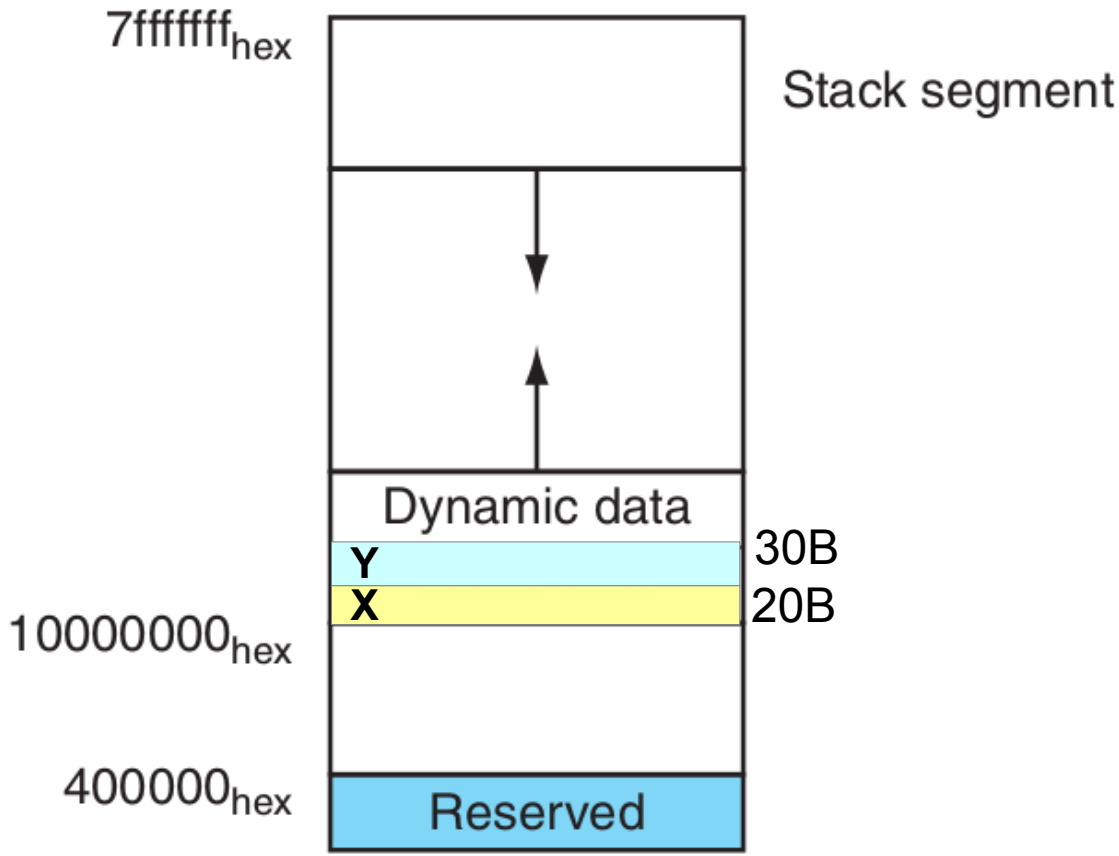
Linking Example

- Value of $gp?
- Addresses of X & Y?



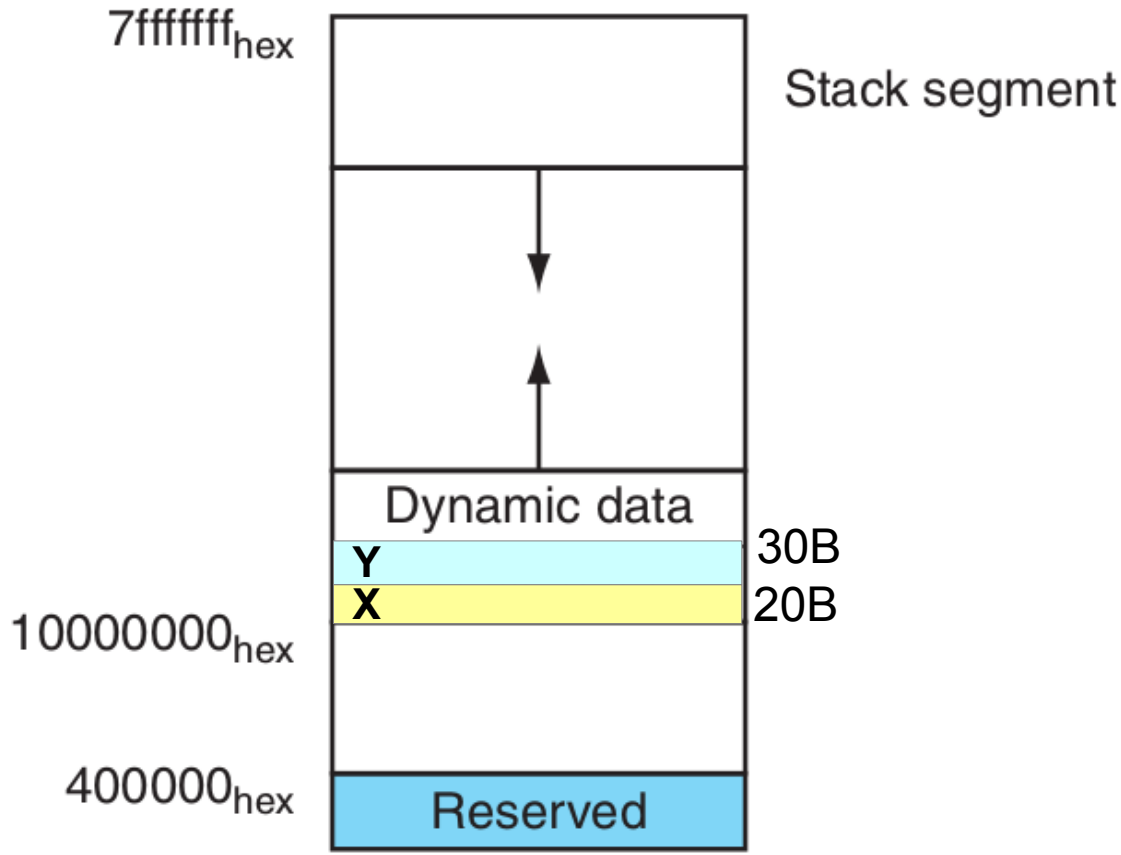
Linking Example

- Value of $gp?
- Addresses of X & Y?



Linking Example

- Value of $gp = 0x10008000
- X = 0x10000000
- Y = 0x10000020



Linking Example – a.out

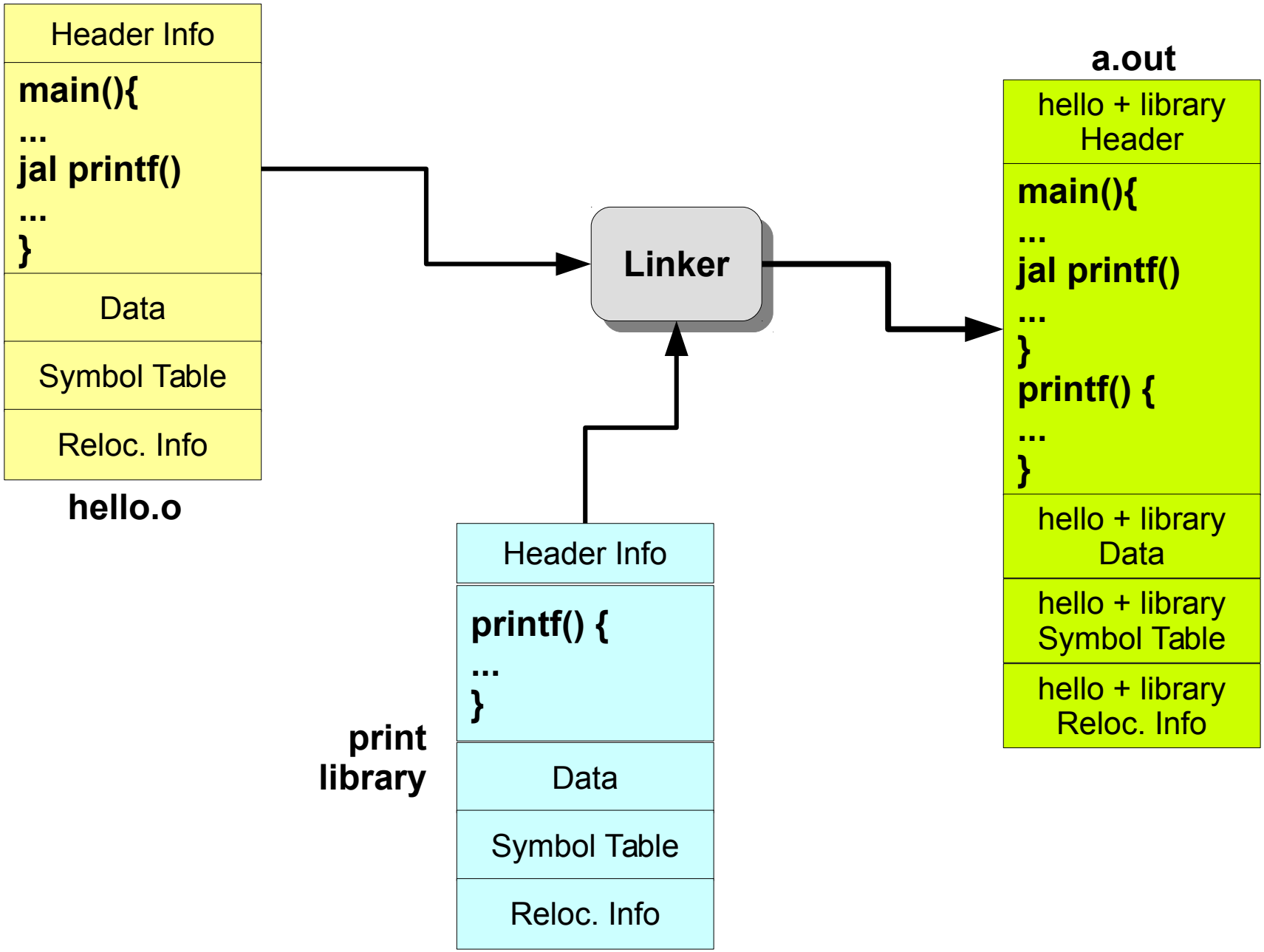
Executable file header		
	Text size	300_{hex}
	Data size	50_{hex}
Text segment	Address	Instruction
	$0040\ 0000_{hex}$	lw $a0, 8000_{hex}($gp)
	$0040\ 0004_{hex}$	jal $40\ 0100_{hex}$
	...	...
	$0040\ 0100_{hex}$	sw $a1, 8020_{hex}($gp)
	$0040\ 0104_{hex}$	jal $40\ 0000_{hex}$
	...	...
Data segment	Address	
	$1000\ 0000_{hex}$	(X)
	...	...
	$1000\ 0020_{hex}$	(Y)
	...	...

Module Outline

- Addressing modes. Instruction classes.
- MIPS-I ISA.
- Translating and starting a program.
- High level languages, Assembly languages and object code.
- Subroutine and subroutine call. Use of stack for handling subroutine call and return.

Backup

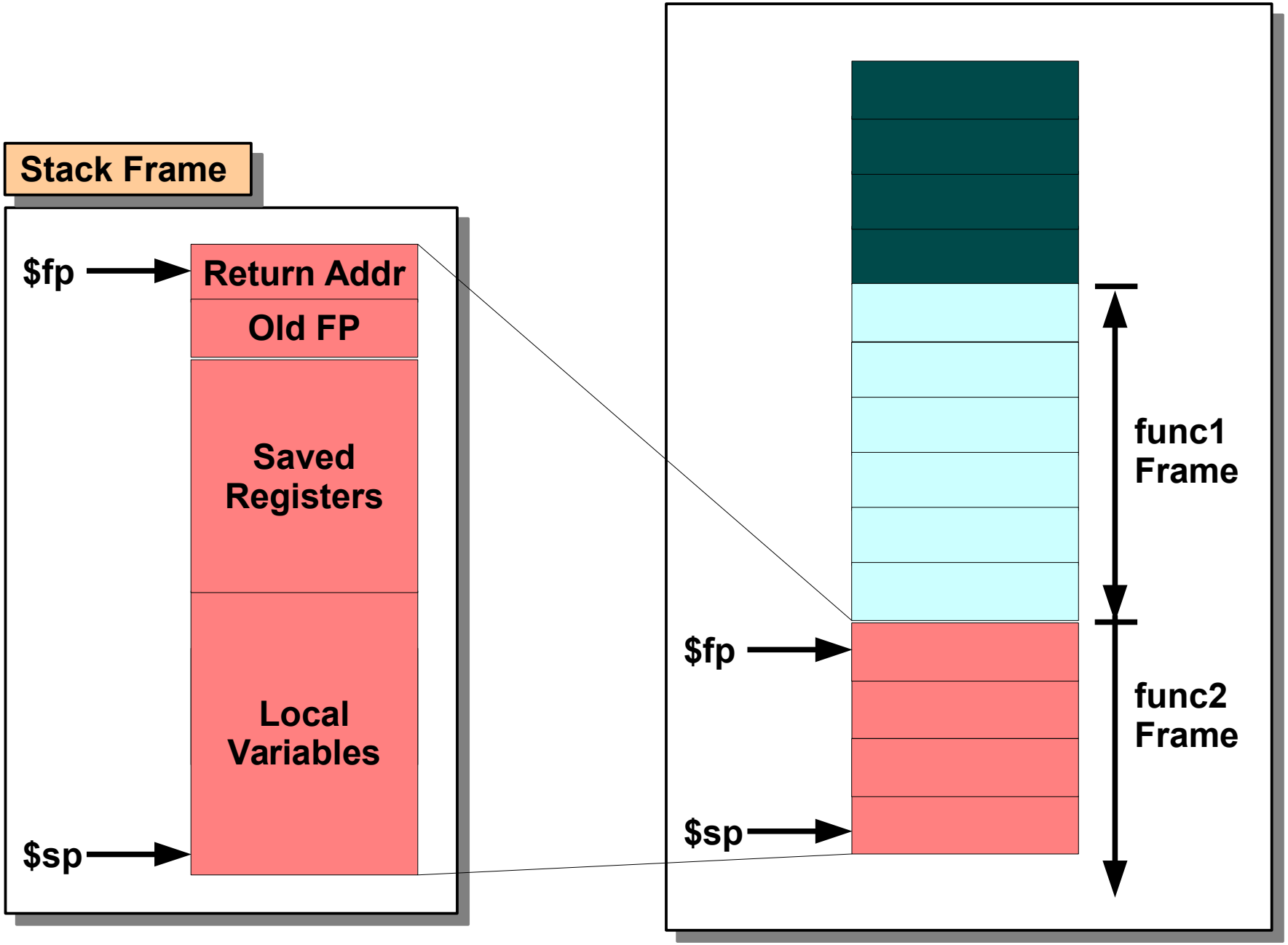
Linking Multiple Modules



The a.out executable

- What does the a.out file contain?
  - Program “code” (machine instructions)
  - Data values (values, size of arrays)
- Other information that is needed for
  - execution
  - debugging
    - Debugging: The stage in program development where mistakes (“bugs”) in the program are identified

Stack Frame – Recall



Saved Registers

- Registers 16 – 23 are saved across function calls

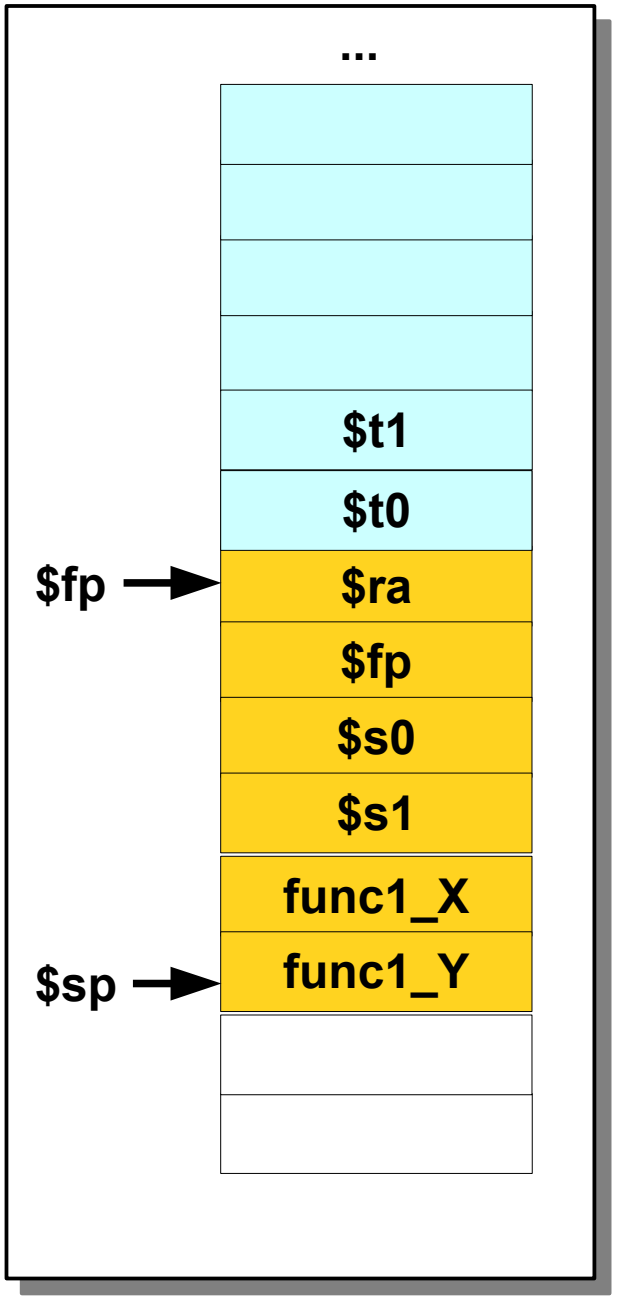
Name	Register number	Usage	Preserved on call?
$zero	0	The constant value 0	n.a.
$v0–$v1	2–3	Values for results and expression evaluation	no
$a0–$a3	4–7	Arguments	no
$t0–$t7	8–15	Temporaries	no
$s0–$s7	16–23	Saved	yes
$t8–$t9	24–25	More temporaries	no
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

Saved Registers

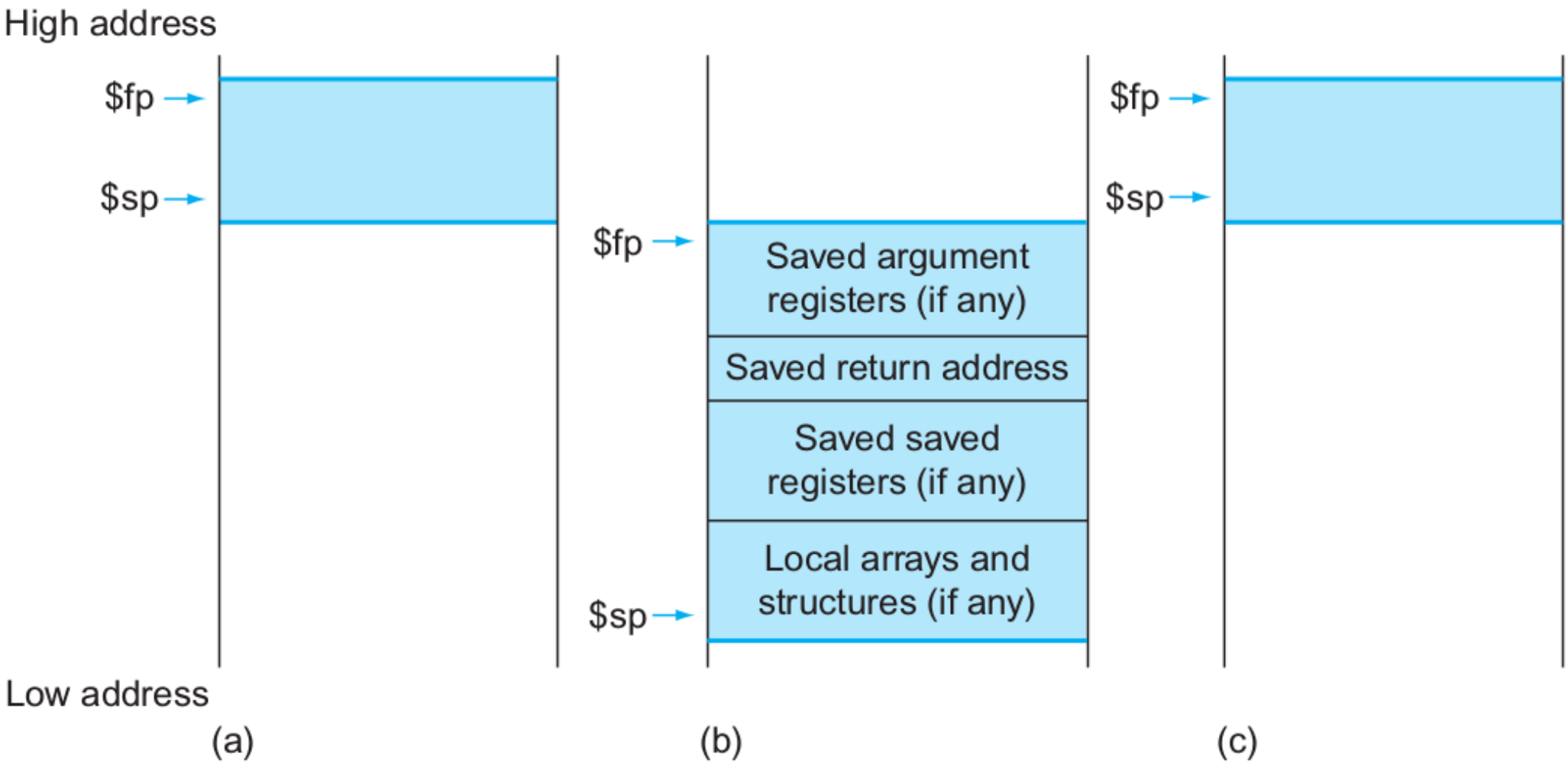
- Registers 16 – 23 are saved across function calls
- Save registers $s0 - $s7 if used by the callee
- Example: $s0, $s1 are saved

Stack Frame

- Local variables are allocated on the stack after the saved registers



Stack Frame


High address
$fp
$sp
Low address
(a)
$fp
Saved argument registers (if any)
Saved return address
Saved saved registers (if any)
Local arrays and structures (if any)
$sp
(b)
$fp
$sp
(c)