CSED211 Lab 05.

Structure, Union, GCC Optimization lab.

19. 10. 16.

Dongbin Na

Heterogeneous Data Structures

- Combining objects of different types: *structures*, *unions*

[Struct]

```
struct Node {
  int i;
  char c;
  int j;
};
```

[Union]

```
union Node {
  int i;
  char c;
  double d;
};
```

Data Alignment

- Suppose 64-bit processor always fetches 8 bytes from memory with an address that must be a multiple of 8
- Therefore, all variable can be read or written with *a single memory operation*
- Intel recommends that data be aligned to improve memory system performance

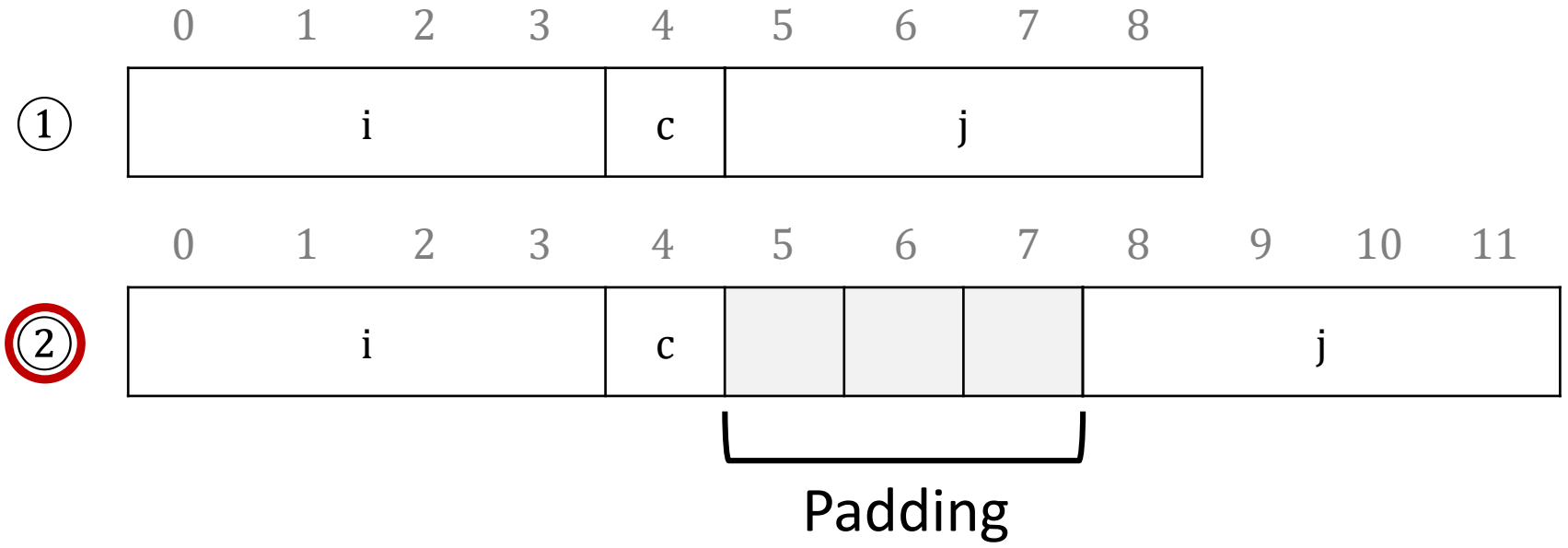
Data Alignment: Example 1)

- If primitive data type requires **K** bytes, address must be multiple of **K**

[CSED211-2019-07.pdf p.33]

Q. Which of the two is the correct diagram of memory?

```
struct Node {
  int i;
  char c;
  int j;
};
```

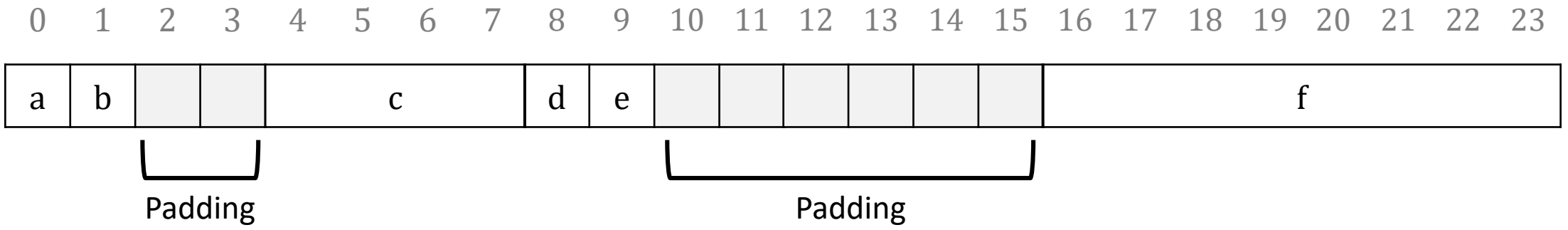


Data Alignment: Example 2)

- If primitive data type requires **K** bytes, address must be multiple of **K**

[CSED211-2019-07.pdf p.33]

```
struct Node {
  char a;
  char b;
  int c;
  char d;
  char e;
  double f;
};
```



Structure Variable Debug Example

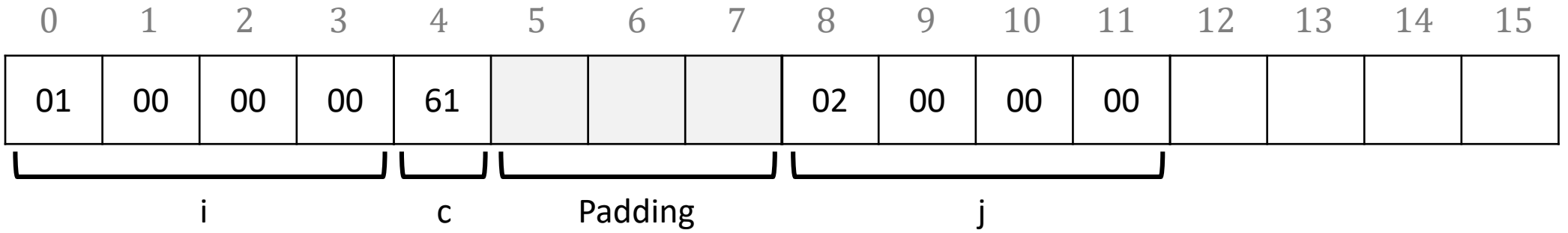
```c
#include <stdio.h>

typedef struct Node {
  int i;
  char c;
  int j;
} Node;

int main(void) {
  Node node;
  node.i = 1;
  node.c = 'a';
  node.j = 2;
  printf("%d", node.i);
  return 0;
}
```

```
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x1,-0x10(%rbp)
   0x0000000000400535 <+15>:    movb   $0x61,-0xc(%rbp)
   0x0000000000400539 <+19>:    movl   $0x2,-0x8(%rbp)
=> 0x0000000000400540 <+26>:    mov    -0x10(%rbp),%eax
   0x0000000000400543 <+29>:    mov    %eax,%esi
   0x0000000000400545 <+31>:    mov    $0x4005e4,%edi
   0x000000000040054a <+36>:    mov    $0x0,%eax
   0x000000000040054f <+41>:    callq  0x400400 <printf@plt>
   0x0000000000400554 <+46>:    mov    $0x0,%eax
   0x0000000000400559 <+51>:    leaveq
   0x000000000040055a <+52>:    retq
```

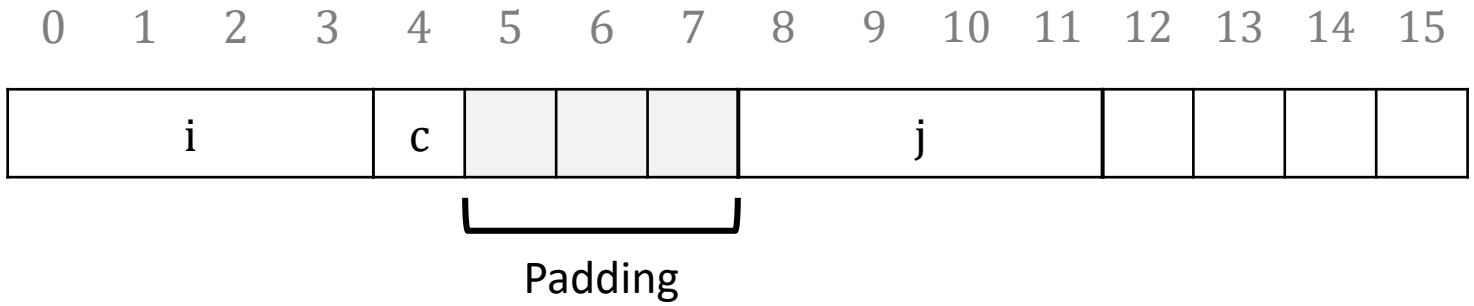
```
(gdb) x/8wx $rbp - 16
0x7ffe4229ded0: 0x00000001      0x00007f61      0x00000002      0x00000000
0x7ffe4229dee0: 0x00400560      0x00000000      0x14a47830      0x00007fee
```



Structure vs Union

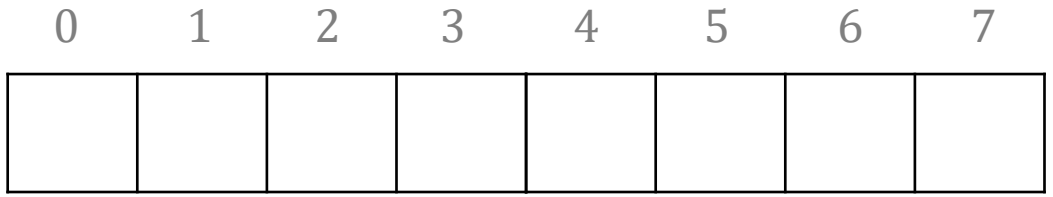
- All fields refer to different addresses in structure

```
struct Node {
  int i;
  char c;
  int j;
};
```



- All fields refer to the same address in union

```
union Node {
  int i;
  char c;
  double d;
};
```



Union Variable Debug Example: int vs char

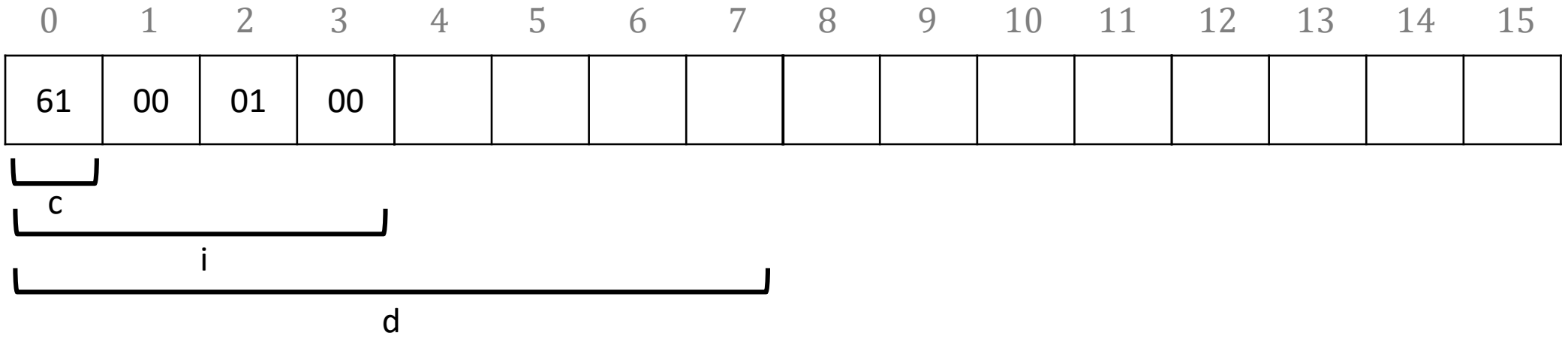
```c
#include <stdio.h>

typedef union Node {
  int i;
  char c;
  double d;
} Node;

int main(void) {
  Node node;
  node.i = 97 + 65536;
  printf("%c", node.c);
  return 0;
}
```

```
   0x0000000000400526 <+0>:     push   %rbp
   0x0000000000400527 <+1>:     mov    %rsp,%rbp
   0x000000000040052a <+4>:     sub    $0x10,%rsp
   0x000000000040052e <+8>:     movl   $0x10061,-0x10(%rbp)
   0x0000000000400535 <+15>:    movzbl -0x10(%rbp),%eax
   0x0000000000400539 <+19>:    movsbl %al,%eax
   0x000000000040053c <+22>:    mov    %eax,%edi
=> 0x000000000040053e <+24>:    callq  0x400400 <putchar@plt>
   0x0000000000400543 <+29>:    mov    $0x0,%eax
   0x0000000000400548 <+34>:    leaveq
   0x0000000000400549 <+35>:    retq
```

```
(gdb) x/8wx $rbp - 16
0x7fffd62dc5e0: 0x00010061      0x00007fff      0x00000000      0x00000000
0x7fffd62dc5f0: 0x00400550      0x00000000      0x7ebee830      0x00007f04
```



Union Variable Debug Example: float vs int

- 31.625 (Float)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…	31
0	1	0	0	0	0	0	1	1	1	1	1	1	1	0	1	…	0

- 1,107,099,648 (Integer)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…	31
0	1	0	0	0	0	0	1	1	1	1	1	1	1	0	1	…	0

```
(gdb) x/8wx $rbp - 16
0x7ffcc52b1e80: 0x41fd0000      0x00007ffc      0x00000000      0x00000000
0x7ffcc52b1e90: 0x00400560      0x00000000      0x969eb830      0x00007f7d
```

0	1	2	3
00	00	fd	41

- So, 31.625 (Float) = 1,107,099,648 (Integer) in Union

GCC Optimization

- -O Level Option
  - To Turn on compiler optimization
  - Increase the compilation time
    - As compiler tries to either improve performance or reduce the size of the output binary
- Not specifying any optimization option
  - Reduce the compilation time, but …

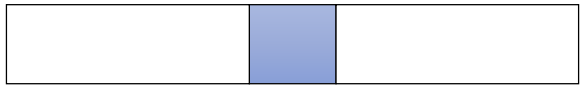
GCC Optimization Option

Option	Optimization Level	Execution Time	Code Size	Memory Usage	Compile Time
-O0	Optimization for compilation time (default)	+	+	-	-
-O1 or -O	Optimization for code size and execution time	-	-	+	+
-O2	Optimization more for code size and execution time	- -		+	+ +
-O3	Optimization more for code size and execution time	- - -		+	+ + +
-Os	Optimization for code size		- -		+ +
-Ofast	O3 with fast none accurate math calculations	- - -		+	+ + +

GCC Optimization Detail Options

- -funroll-loops
  - Unroll loops whose number of iterations can be determined at compile time or upon entry to the loop
- -fprefetch-loop-arrays
  - If supported by the target machine, generate instructions to prefetch memory to improve the performance of loops that access large arrays
- Further options are in here: https://gcc.gnu.org/onlinedocs/gcc-4.5.2/gcc/Optimize-Options.html

Locality



- Temporal Locality
  - Recently referenced items are likely to be referenced again in the near future
  - After accessing address X in memory, save the bytes in cache for future access

- Spatial Locality
  - Items with nearby addresses tend to be referenced close together in time
  - After accessing address X, save the block of memory around X in cache for future access

How to write assembly code in C

- To write assembly code in C, programmer can use '**__asm()**' function.
- Inline assembler can be used anywhere in which C or C++ can be used.

```
(c code)
__asm (
    …
    (assembly code)
    …
    : OutputOperands
    : InputOperands
)
(c code)
```

- Usage of Inline Assembly
  - Writing functions in assembly language.
  - Spot-optimizing speed-critical sections of code.
  - Making direct hardware access for device drivers.

- Further explanation is in here: https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html

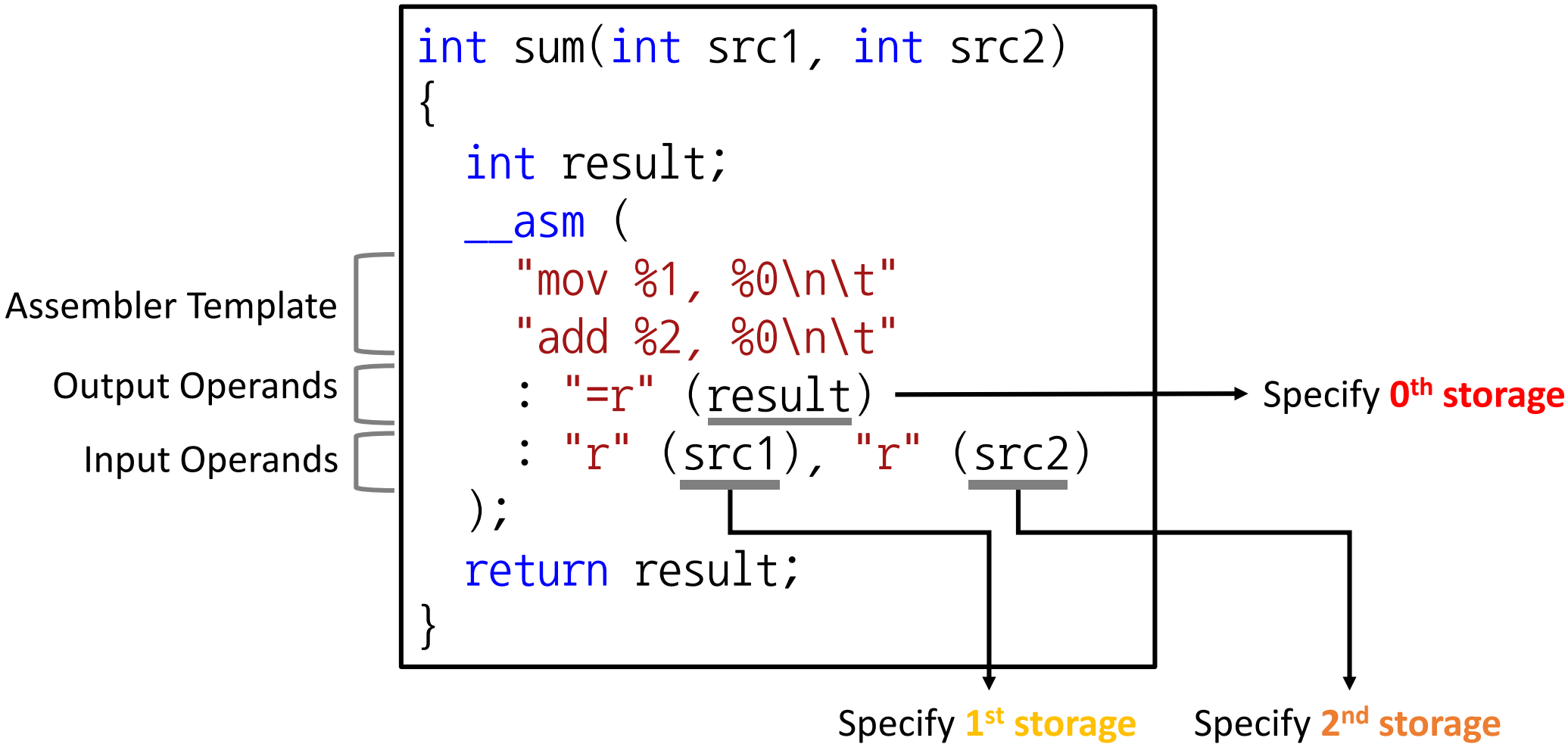
ASM Parameters: Assembler Template

- A literal string containing assembler instructions
- Assembly codes are divided by *'\n\t'*
- %n: Use the register mapped to the n^{th} argument

ASM Parameters: Operands

- Comma-separated list of the C variables used for instructions
- Common constraints include 'r' for register and 'm' for memory
- Output constraints must begin with either '=' (for overwriting) or '+' (for reading and writing)

__asm Example


int sum(int src1, int src2)
{
int result;
__asm (
"mov %1, %0\n\t"
"add %2, %0\n\t"
: "=r" (result)
: "r" (src1), "r" (src2)
);
return result;
}
Assembler Template
Output Operands
Input Operands
Specify 0th storage
Specify 1st storage
Specify 2nd storage

Q & A

- No Lab homework in this session, get ready for your midterms!