

C

1

Changelog

Corrections made in this version not seen in first lecture:

- 24 August 2017: slide 3 (a note on next week's reading): adjust for lack of quiz
- 24 August 2017: slide 20 (logical operators): add parenthesis
- 24 August 2017: slide 27: remove extraneous * from line 4 (was not present on slide 27)

1

quiz demo

1

logistics notes

quizzes

- post-quiz: released Thursday night, due Sat
- pre-quiz: released Sat, due Tuesday morning

note on comments:

- we will eventually have TAs look at them
(for answers autograded as wrong)

question order is *randomized* —

- avoid referring to answer letters in your comments

2

a note on next week's reading

we're starting x86-64 assembly

using AT&T syntax — not what you learned in 2150
other syntax: "Intel syntax"

some important differences (described at end of section 3.3)

quiz will avoid assembly syntax issues

we will cover next week

3

lab accounts

they should work now

ignore typo of Spring for Fall in email body (subject is okay)

if you registered this week, likely today

let me know if you don't have one by Monday

4

slides + audio + feedback

slides and video and audio on website
when I don't mess up recording

anonymous feedback on Collab exists

5

office hours

TA office hours should start next week

all office hours on course website

my office hours:

Monday: 1PM-2PM

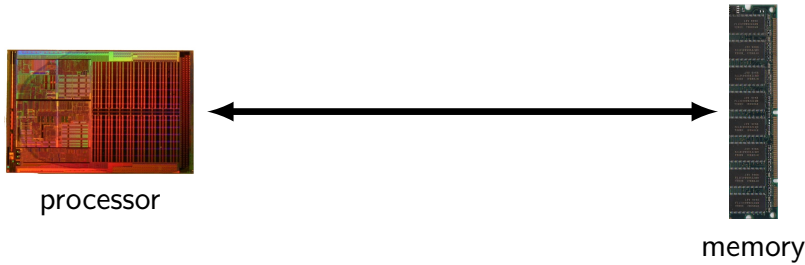
Wednesday: 2PM-3PM

Friday this week: 10AM-11AM; 1PM-2PM

Friday future weeks: TBA

6

processors and memory



Images:
Single core Opteron 8xx die: Dg2fer at the German language Wikipedia, via Wikimedia Commons
SDRAM by Arnaud 25, via Wikimedia Commons

7

memory

address	value
0xFFFFFFFF	0x14
0xFFFFFFF	0x45
0xFFFFFFF	0xDE
...	...
0x00042006	0x06
0x00042005	0x05
0x00042004	0x04
0x00042003	0x03
0x00042002	0x02
0x00042001	0x01
0x00042000	0x00
0x00041FFF	0x03
0x00041FFE	0x60
...	...
0x00000002	0xFE
0x00000001	0xE0
0x00000000	0xA0

8

memory

address	value
0xFFFFFFFF	0x14
0xFFFFFFF	0x45
0xFFFFFFF	0xDE
...	...
0x00042006	0x06
0x00042005	0x05
0x00042004	0x04
0x00042003	0x03
0x00042002	0x02
0x00042001	0x01
0x00042000	0x00
0x00041FFF	0x03
0x00041FFE	0x60
...	...
0x00000002	0xFE
0x00000001	0xE0
0x00000000	0xA0

array of bytes (byte = 8 bits)
CPU interprets based on how accessed

8

memory

address	value
0xFFFFFFFF	0x14
0xFFFFFFF	0x45
0xFFFFFFF	0xDE
...	...
0x00042006	0x06
0x00042005	0x05
0x00042004	0x04
0x00042003	0x03
0x00042002	0x02
0x00042001	0x01
0x00042000	0x00
0x00041FFF	0x03
0x00041FFE	0x60
...	...
0x00000002	0xFE
0x00000001	0xE0
0x00000000	0xA0

address	value
0x00000000	0xA0
0x00000001	0xE0
0x00000002	0xFE
...	...
0x00041FFE	0x60
0x00041FFF	0x03
0x00042000	0x00
0x00042001	0x01
0x00042002	0x02
0x00042003	0x03
0x00042004	0x04
0x00042005	0x05
0x00042006	0x06
...	...
0xFFFFFFF	0xDE
0xFFFFFFF	0x45
0xFFFFFFF	0x14

8

endianness

address	value
0xFFFFFFFF	0x14
0xFFFFFFF	0x45
0xFFFFFFF	0xDE
...	...
0x00042006	0x06
0x00042005	0x05
0x00042004	0x04
0x00042003	0x03
0x00042002	0x02
0x00042001	0x01
0x00042000	0x00
0x00041FFF	0x03
0x00041FFE	0x60
...	...
0x00000002	0xFE
0x00000001	0xE0

```
int *x = (int*)0x42000;
cout << *x << endl;
```

9

endianness

address	value
0xFFFFFFFF	0x14
0xFFFFFFF	0x45
0xFFFFFFF	0xDE
...	...
0x00042006	0x06
0x00042005	0x05
0x00042004	0x04
0x00042003	0x03
0x00042002	0x02
0x00042001	0x01
0x00042000	0x00
0x00041FFF	0x03
0x00041FFE	0x60
...	...
0x00000002	0xFE
0x00000001	0xE0

```
int *x = (int*)0x42000;
cout << *x << endl;
```

9

endianness

address	value
0xFFFFFFFF	0x14
0xFFFFFFF	0x45
0xFFFFFFF	0xDE
...	...
0x00042006	0x06
0x00042005	0x05
0x00042004	0x04
0x00042003	0x03
0x00042002	0x02
0x00042001	0x01
0x00042000	0x00
0x00041FFF	0x03
0x00041FFE	0x60
...	...
0x00000002	0xFE
0x00000001	0xE0

```
int *x = (int*)0x42000;
cout << *x << endl;
```

0x03020100 = 50462976

0x00010203 = 66051

9

endianness

address	value
0xFFFFFFFF	0x14
0xFFFFFFF	0x45
0xFFFFFFF	0xDE
...	...
0x00042006	0x06
0x00042005	0x05
0x00042004	0x04
0x00042003	0x03
0x00042002	0x02
0x00042001	0x01
0x00042000	0x00
0x00041FFF	0x03
0x00041FFE	0x60
...	...
0x00000002	0xFE
0x00000001	0xE0

```
int *x = (int*)0x42000;
cout << *x << endl;
```

0x03020100 = 50462976

little endian
(least significant byte has lowest address)

0x00010203 = 66051

big endian
(most significant byte has lowest address)

9

endianness

address	value	
0xFFFFFFFF	0x14	
0xFFFFFFF	0x45	
0xFFFFFFF	0xDE	
...	...	
0x00042006	0x06	
0x00042005	0x05	
0x00042004	0x04	
0x00042003	0x03	
0x00042002	0x02	
0x00042001	0x01	
0x00042000	0x00	
0x00041FFF	0x03	
0x00041FFE	0x60	
...	...	
0x00000002	0xFE	
0x00000001	0xE0	

```
int *x = (int*)0x42000;
cout << *x << endl;
```

0x03020100 = 50462976

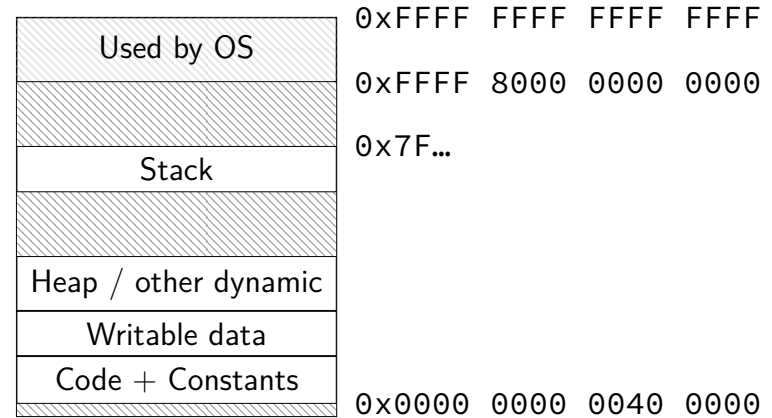
little endian
(least significant byte has lowest address)

0x00010203 = 66051

big endian
(most significant byte has lowest address)

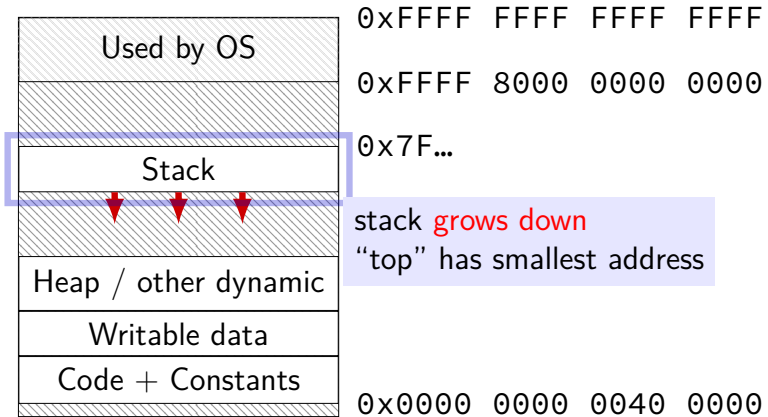
9

program memory (x86-64 Linux)



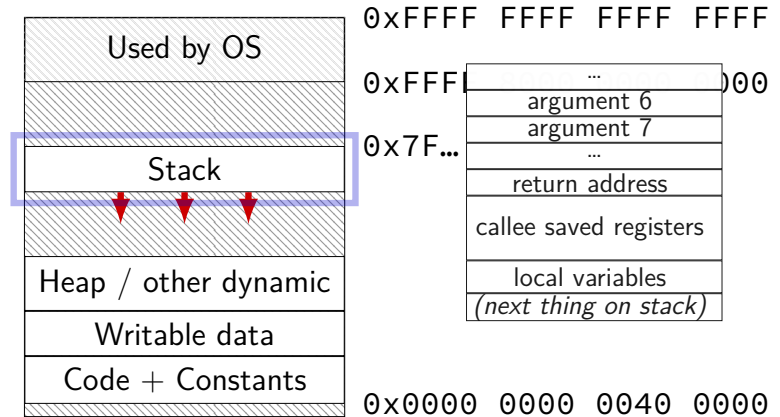
10

program memory (x86-64 Linux)



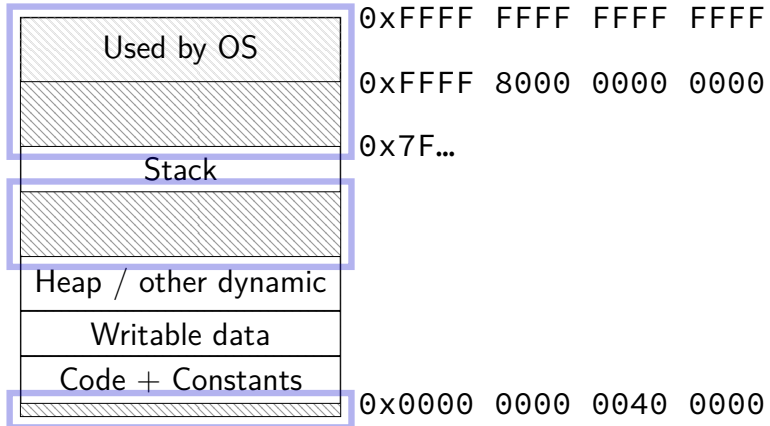
10

program memory (x86-64 Linux)



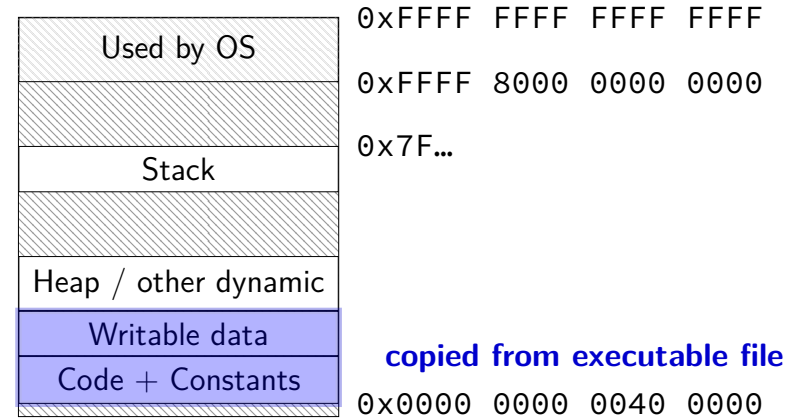
10

program memory (x86-64 Linux)



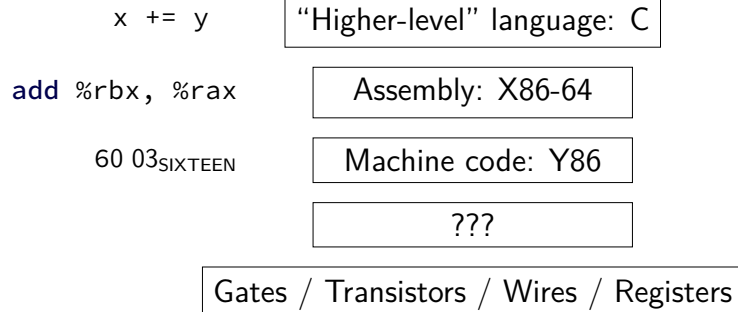
10

program memory (x86-64 Linux)



10

layers of abstraction



11

compilation commands

compile: gcc -S file.c ⇒ file.s (assembly)
assemble: gcc -c file.s ⇒ file.o (object file)
link: gcc -o file file.o ⇒ file (executable)

c+a: gcc -c file.c ⇒ file.o
 c+a+l: gcc -o file file.c ⇒ file
 ...

12

what's in those files?

hello.c

```
#include <stdio.h>
int main(void) {
    puts("Hello, World!");
    return 0;
}
```

13

what's in those files?

hello.c

```
#include <stdio.h>
int main(void) {
    puts("Hello, World!");
    return 0;
}
```

hello.s

```
.text
main:
    sub    $8, %rsp
    mov    $.Lstr, %rdi
    call   puts
    xor    %eax, %eax
    add    $8, %rsp
    ret

.data
.Lstr: .string "Hello,_World!"
```

13

what's in those files?

hello.c

```
#include <stdio.h>
int main(void) {
    puts("Hello, World!");
    return 0;
}
```

hello.s

```
.text
main:
    sub    $8, %rsp
    mov    $.Lstr, %rdi
    call   puts
    xor    %eax, %eax
    add    $8, %rsp
    ret

.data
.Lstr: .string "Hello,_World!"
```

hello.o

text (code) segment:
48 83 EC 08 BF 00 00 00 00 E8 00 00
00 00 31 C0 48 83 C4 08 C3

13

what's in those files?

hello.c

```
#include <stdio.h>
int main(void) {
    puts("Hello, World!");
    return 0;
}
```

hello.s

```
.text
main:
    sub    $8, %rsp
    mov    $.Lstr, %rdi
    call   puts
    xor    %eax, %eax
    add    $8, %rsp
    ret

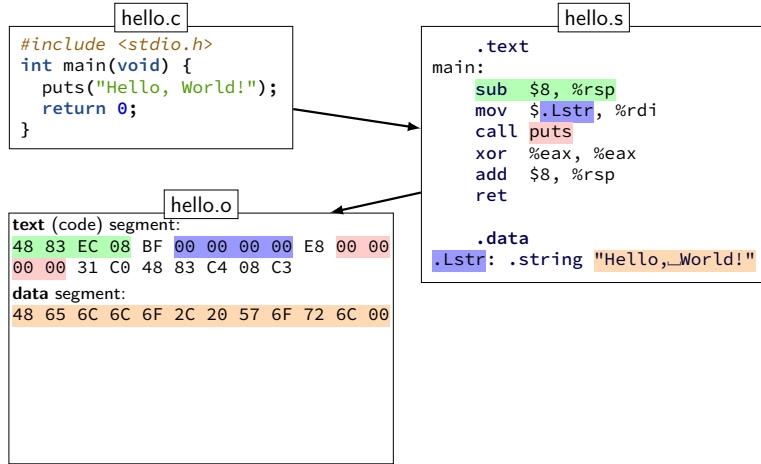
.data
.Lstr: .string "Hello,_World!"
```

hello.o

text (code) segment:
48 83 EC 08 BF 00 00 00 00 E8 00 00
00 00 31 C0 48 83 C4 08 C3
data segment:
48 65 6C 6C 6F 2C 20 57 6F 72 6C 00

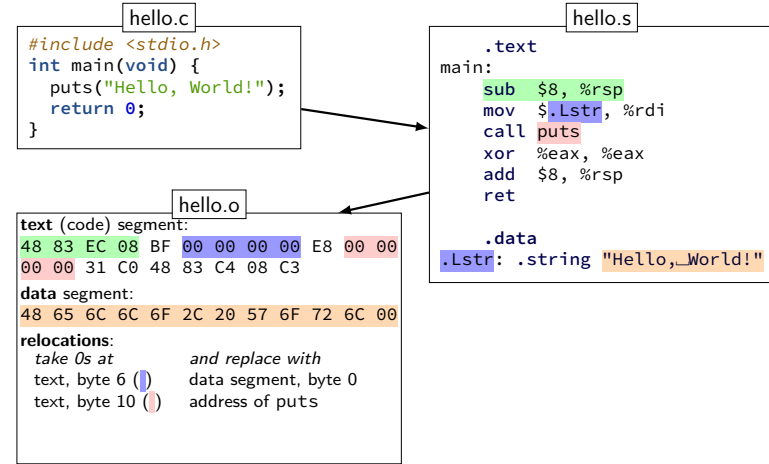
13

what's in those files?



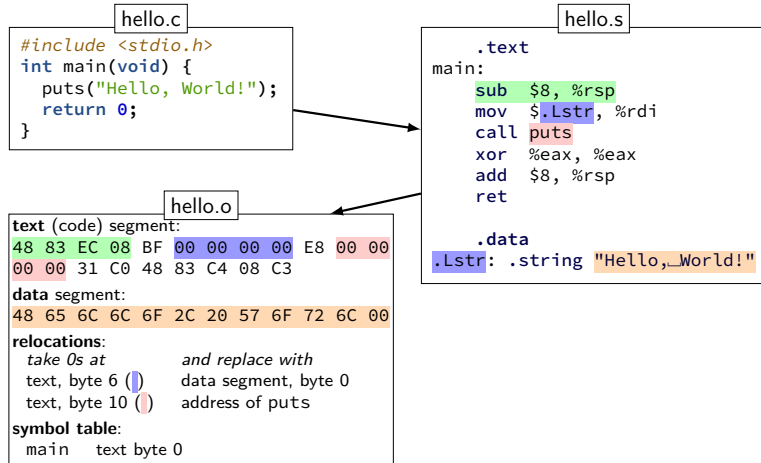
13

what's in those files?



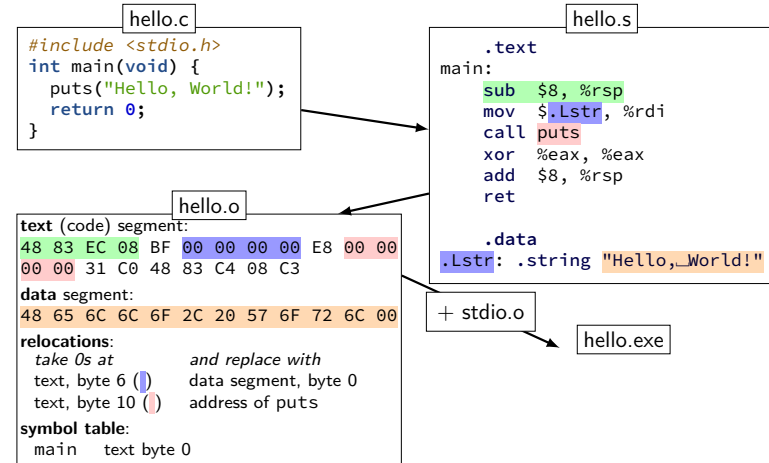
13

what's in those files?



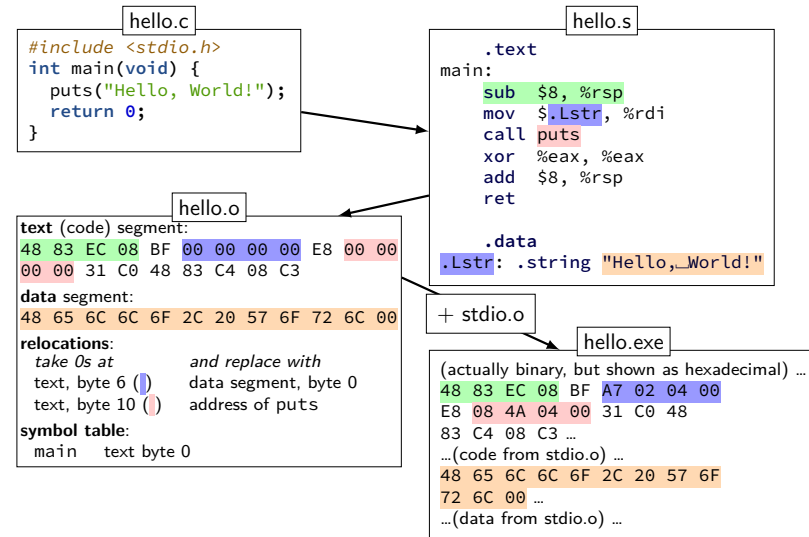
13

what's in those files?



13

what's in those files?



13

exercise (1)

main.c:

```
1 #include <stdio.h>
2 void sayHello(void) {
3     puts("Hello, World!");
4 }
5 int main(void) {
6     sayHello();
7 }
```

Which files contain the **memory address** of sayHello?

- A. main.s (assembly) D. B and C
- B. main.o (object) E. A, B and C
- C. main.exe (executable) F. something else

14

exercise (2)

main.c:

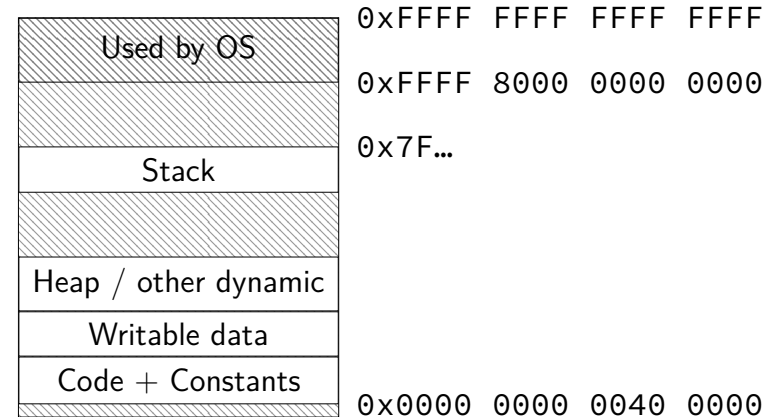
```
1 #include <stdio.h>
2 void sayHello(void) {
3     puts("Hello, World!");
4 }
5 int main(void) {
6     sayHello();
7 }
```

Which files contain the **literal ASCII string** of Hello, World!?

- A. main.s (assembly) D. B and C
- B. main.o (object) E. A, B and C
- C. main.exe (executable) F. something else

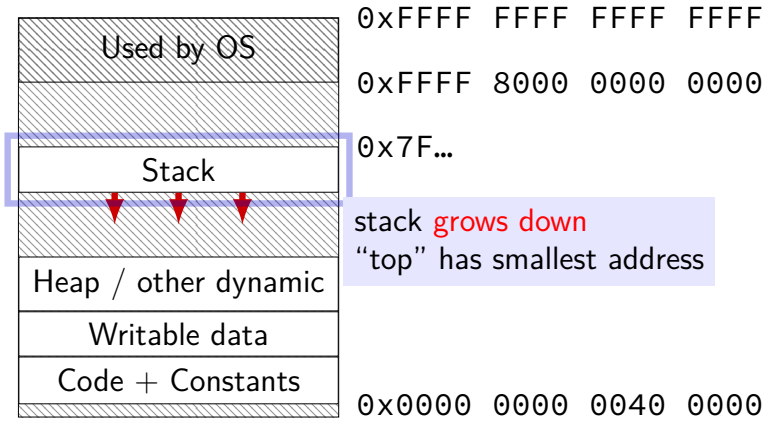
15

program memory (x86-64 Linux)



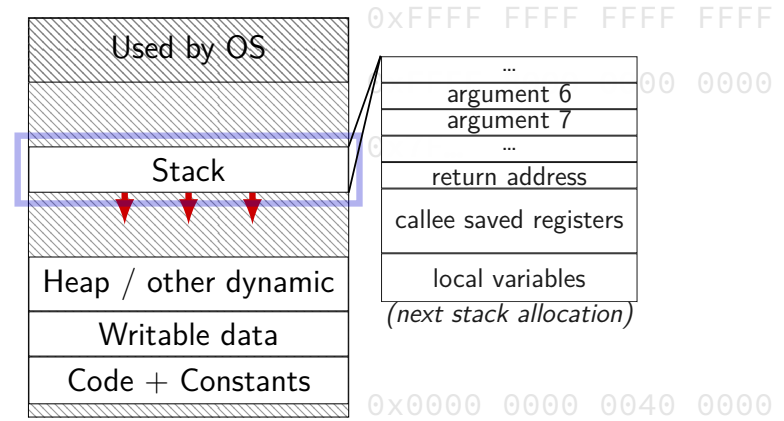
16

program memory (x86-64 Linux)



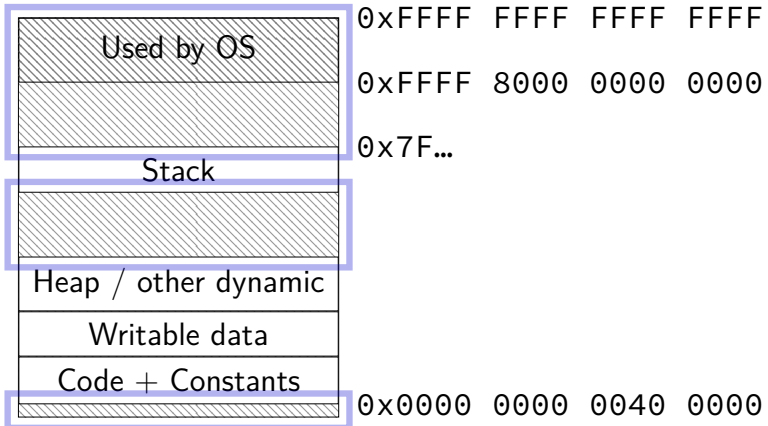
16

program memory (x86-64 Linux)



16

program memory (x86-64 Linux)



16

C Data Types

Varies between machines(!). For **this course**:

type	size (bytes)
char	1
short	2
int	4
long	8

17

C Data Types

Varies between machines(!). For **this course**:

type	size (bytes)
char	1
short	2
int	4
long	8
float	4
double	8

17

C Data Types

Varies between machines(!). For **this course**:

type	size (bytes)
char	1
short	2
int	4
long	8
float	4
double	8
void *	8
<i>anything</i> *	8

17

truth

bool

18

truth

bool

x **==** 4 is an **int**
1 if true; 0 if false

18

false values in C

0

including null pointers — 0 cast to a pointer

19

logical operators

return 1 for true or 0 for false

(1 && 1) == 1	(1 1) == 1
(2 && 4) == 1	(2 4) == 1
(1 && 0) == 0	(1 0) == 1
(0 && 0) == 0	(0 0) == 0
(-1 && -2) == 1	(-1 -2) == 1
("" && "") == 1	("" "") == 1
! 1 == 0	
! 4 == 0	
! -1 == 0	
! 0 == 1	

20

short-circuit (||)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() || one());
6     printf(">_%d\n", one() || zero());
7     return 0;
8 }
```

zero()

one()

> 1

one()

> 1

OR	false	true
false	false	true
true	true	true

21

short-circuit (||)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() || one());
6     printf(">_%d\n", one() || zero());
7     return 0;
8 }
```

zero()

one()

> 1

one()

> 1

OR	false	true
false	false	true
true	true	true

21

short-circuit (||)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() || one());
6     printf(">_%d\n", one() || zero());
7     return 0;
8 }
```

zero()

one()

> 1

one()

> 1

	OR	false	true
false	false	false	true
true	true	true	true

21

short-circuit (||)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() || one());
6     printf(">_%d\n", one() || zero());
7     return 0;
8 }
```

zero()

one()

> 1

one()

> 1

	OR	false	true
false	false	false	true
true	true	true	true

21

short-circuit (||)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() || one());
6     printf(">_%d\n", one() || zero());
7     return 0;
8 }
```

zero()

one()

> 1

one()

> 1

	OR	false	true
false	false	false	true
true	true	true	true

21

short-circuit (&&)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() && one());
6     printf(">_%d\n", one() && zero());
7     return 0;
8 }
```

zero()

> 0

one()

zero()

> 0

	AND	false	true
false	false	false	false
true	true	false	true

22

short-circuit (&&)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() && one());
6     printf(">_%d\n", one() && zero());
7     return 0;
8 }
```

zero()

> 0

one()

zero()

> 0

AND	false	true
false	false	false
true	false	true

22

short-circuit (&&)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() && one());
6     printf(">_%d\n", one() && zero());
7     return 0;
8 }
```

zero()

> 0

one()

zero()

> 0

AND	false	true
false	false	false
true	false	true

22

short-circuit (&&)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() && one());
6     printf(">_%d\n", one() && zero());
7     return 0;
8 }
```

zero()

> 0

one()

zero()

> 0

AND	false	true
false	false	false
true	false	true

22

short-circuit (&&)

```
1 #include <stdio.h>
2 int zero() { printf("zero()\n"); return 0; }
3 int one() { printf("one()\n"); return 1; }
4 int main() {
5     printf(">_%d\n", zero() && one());
6     printf(">_%d\n", one() && zero());
7     return 0;
8 }
```

zero()

> 0

one()

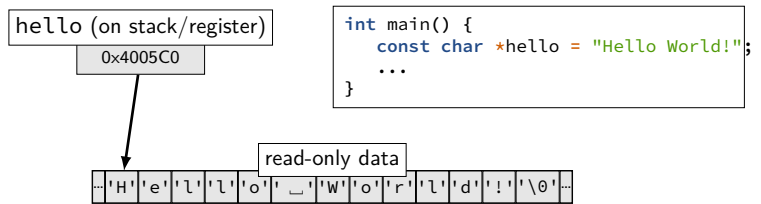
zero()

> 0

AND	false	true
false	false	false
true	false	true

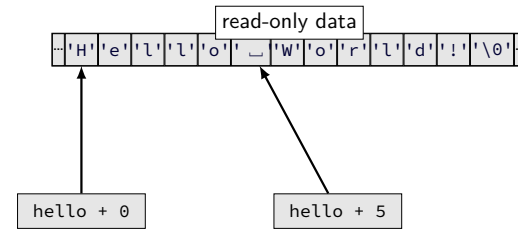
22

strings in C



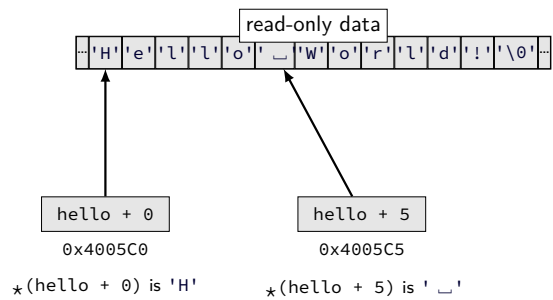
23

pointer arithmetic



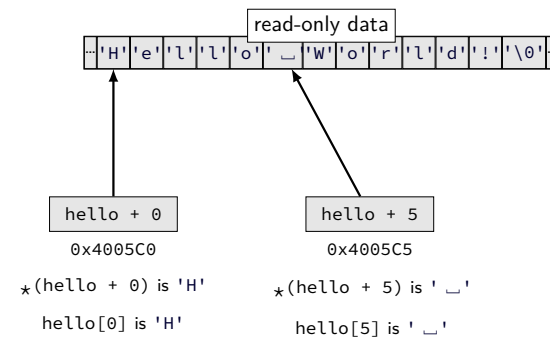
24

pointer arithmetic



24

pointer arithmetic



24

arrays and pointers

`*(foo + bar)` exactly the same as `foo[bar]`

arrays 'decay' into pointers

25

exercise

```
1 char foo[4] = "foo";
2    // {'f', 'o', 'o', '\0'}
3 char *pointer;
4 pointer = foo;
5 *pointer = 'b';
6 pointer = pointer + 2;
7 pointer[0] = 'z';
8 *(foo + 1) = 'a';
```

Final value of foo?

- A. "fao" D. "bao"
- B. "zao" E. something else/crash
- C. "baz"

26

exercise

```
1 char foo[4] = "foo";
2    // {'f', 'o', 'o', '\0'}
3 char *pointer;
4 pointer = foo;
5 *pointer = 'b';
6 pointer = pointer + 2;
7 pointer[0] = 'z';
8 *(foo + 1) = 'a';
```

Final value of foo?

- A. "fao" D. "bao"
- B. "zao" E. something else/crash
- C. "baz"

26

exercise explanation

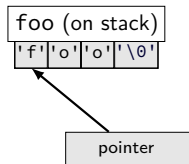
```
1 char foo[4] = "foo";
2    // {'f', 'o', 'o', '\0'}
3 char *pointer;
4 pointer = foo;
5 *pointer = 'b';
6 pointer = pointer + 2;
7 pointer[0] = 'z';
8 *(foo + 1) = 'a';
```

foo (on stack)			
'f'	'o'	'o'	'\0'

27

exercise explanation

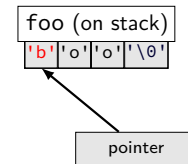
```
1 char foo[4] = "foo";
2   // {'f', 'o', 'o', '\0'}
3 char *pointer;
4 pointer = foo;
5 *pointer = 'b';
6 pointer = pointer + 2;
7 pointer[0] = 'z';
8 *(foo + 1) = 'a';
```



27

exercise explanation

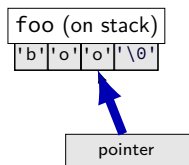
```
1 char foo[4] = "foo";
2   // {'f', 'o', 'o', '\0'}
3 char *pointer;
4 pointer = foo;
5 *pointer = 'b';
6 pointer = pointer + 2;
7 pointer[0] = 'z';
8 *(foo + 1) = 'a';
```



27

exercise explanation

```
1 char foo[4] = "foo";
2   // {'f', 'o', 'o', '\0'}
3 char *pointer;
4 pointer = foo;
5 *pointer = 'b';
6 pointer = pointer + 2;
7 pointer[0] = 'z';
8 *(foo + 1) = 'a';
```

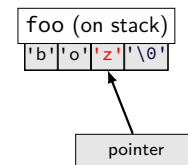


27

exercise explanation

```
1 char foo[4] = "foo";
2   // {'f', 'o', 'o', '\0'}
3 char *pointer;
4 pointer = foo;
5 *pointer = 'b';
6 pointer = pointer + 2;
7 pointer[0] = 'z';
8 *(foo + 1) = 'a';
```

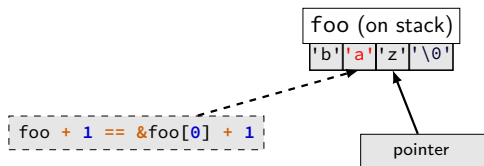
better style: `*pointer = 'z';`



27

exercise explanation

```
1 char foo[4] = "foo";
2   // {'f', 'o', 'o', '\0'}
3 char *pointer;
4 pointer = foo;
5 *pointer = 'b';
6 pointer = pointer + 2;
7 pointer[0] = 'z';    better style: *pointer = 'z';
8 *(foo + 1) = 'a';    better style: foo[1] = 'a';
```



27

example: C that is not C++

valid C and invalid C++:

```
char *str = malloc(100);
```

valid C and valid C++:

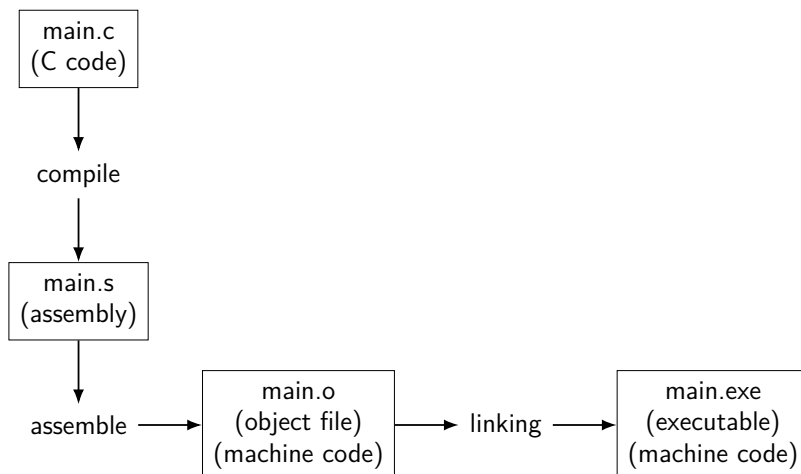
```
char *str = (char *) malloc(100);
```

valid C and invalid C++:

```
int class = 1;
```

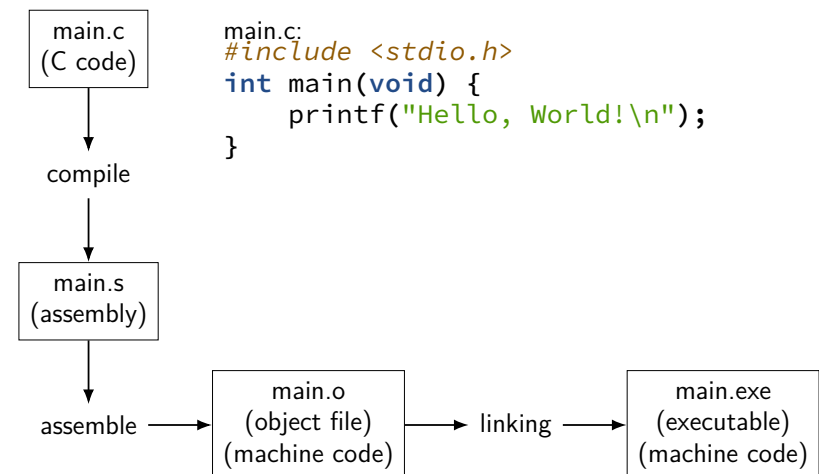
28

compilation pipeline



29

compilation pipeline



29

compilation pipeline

