Lifetime of HelloWorld Program

CSE 238/2038/2138: Systems Programming

Instructor:

Fatma CORUT ERGİN

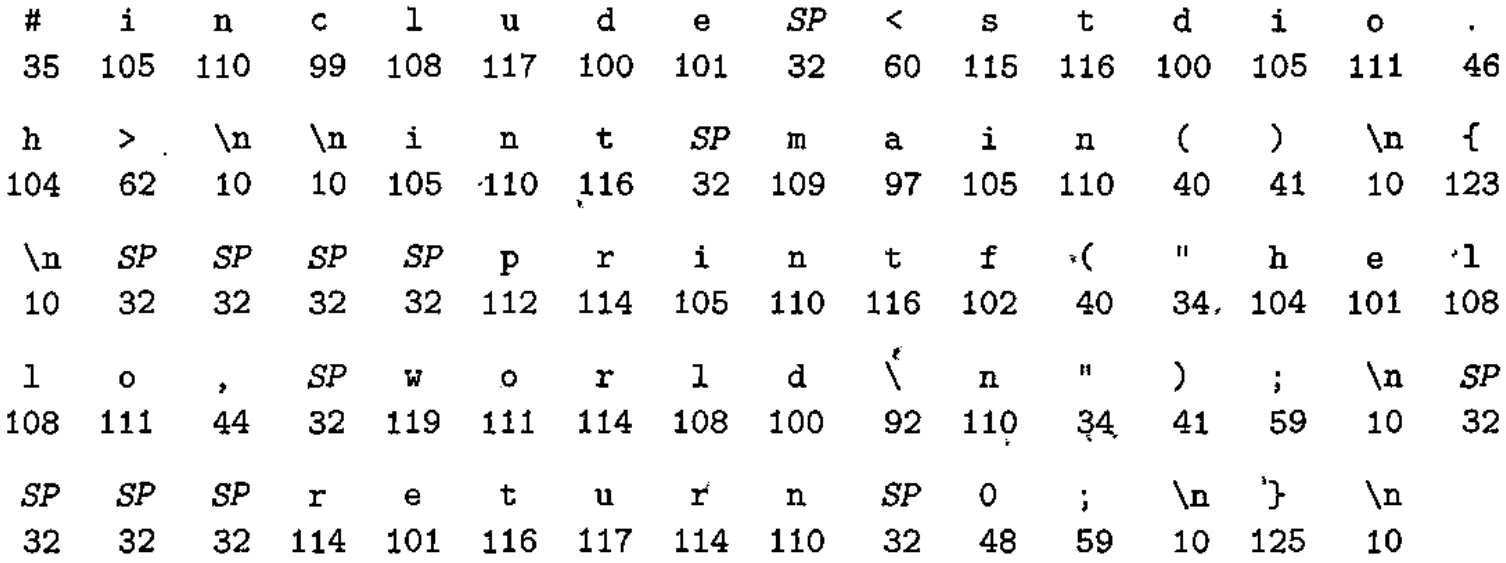
Hello World Program in C

```
#include <stdio.h>

int main(){
       printf(“Hello World! \n”);

       return 0;
}
```

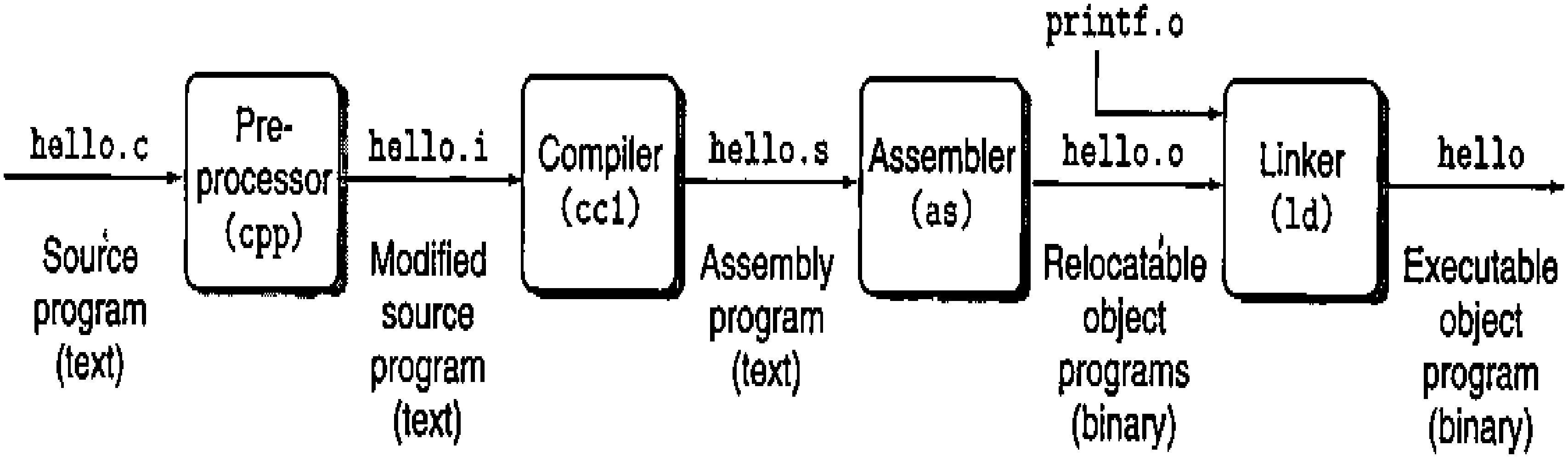
The program → hello.c



#	i	n	c	l	u	d	e	*SP*	<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46
h	>	\n	\n	i	n	t	*SP*	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123
\n	*SP*	*SP*	*SP*	*SP*	p	r	i	n	t	f	(	"	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108
l	o	,	*SP*	w	o	r	l	d	\	n	"	)	;	\n	*SP*
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	32
SP	*SP*	*SP*	r	e	t	u	r	n	*SP*	0	;	\n	}	\n	
32	32	32	114	101	116	117	114	110	32	48	59	10	125	10	

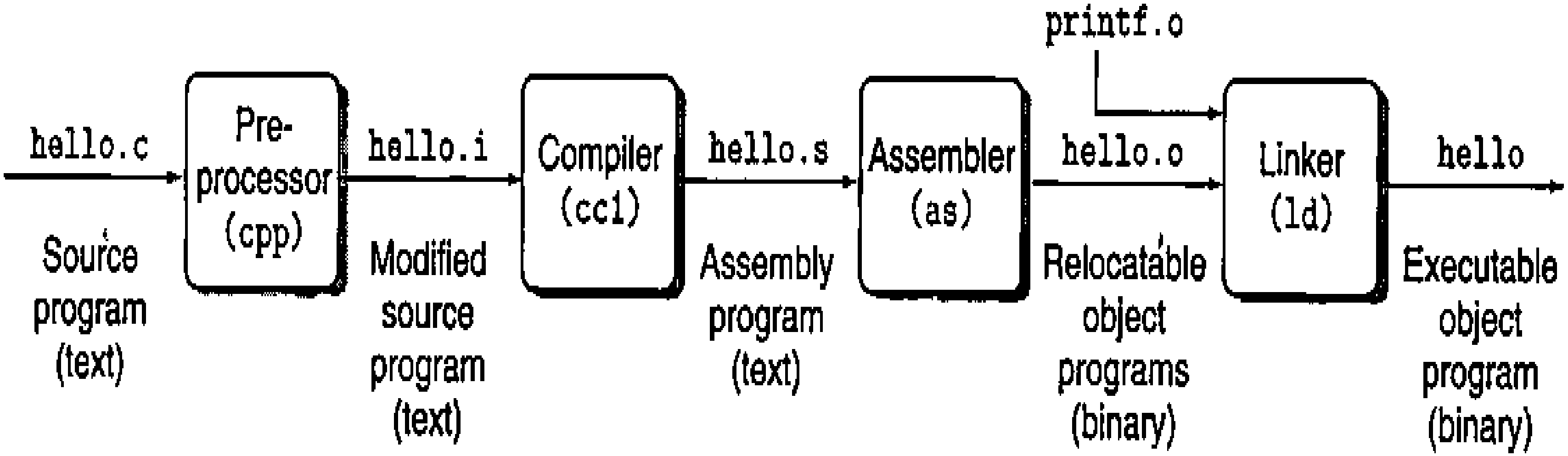
The ASCII text representation of hello.c

The Compilation System



- The gcc compiler driver reads source file “hello.c” and translates it into an executable object file “hello”
- The translation is performed in the sequence of four phases
  - preprocessor,
  - compiler,
  - assembler, and
  - linker

The Compilation System



Preprocessing phase

- The preprocessor (cpp) modifies the original C program according to directives that begin with the '#' character.
- For example, the #include <stdio.h> command in line 1 of hello. c tells the pre-processor to
  - read the contents of the system header file stdio.h and
  - insert it directly into the program text.
  - The result is another C program, typically with the .i suffix → *hello.i*

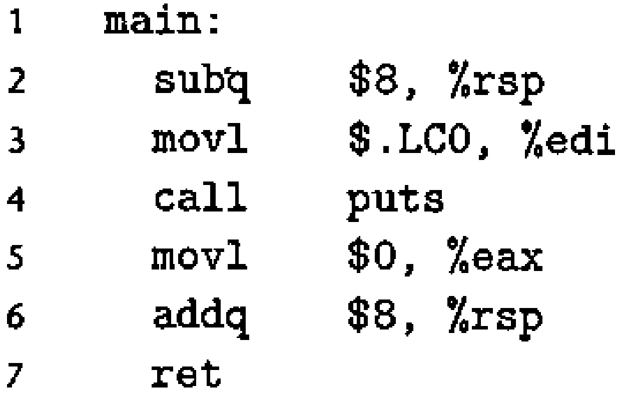
The Compilation System

Compilation phase

- The compiler (cc1) translates the text file hello. i into the text file hello. s, which contains an *assembly-language program* → *hello.s*

- This program includes the definition of function main

- Each of lines 2-7 in definition describes one low-level machine-language instruction in a textual form.



```
1   main:
2       subq    $8, %rsp
3       movl    $.LC0, %edi
4       call    puts
5       movl    $0, %eax
6       addq    $8, %rsp
7       ret
```

- Assembly language is useful because it provides a common output language for different compilers for different high-level languages.

- For example, C compilers and Fortran compilers both generate output files in the same assembly language.

The Compilation System

- **Assembly phase**
  - Next ,the assembler (as)
    - translates hello.s into machine language instructions,
    - packages them in a form known as a *relocatable object program,* and
    - stores the result in the object file *hello. o*.
  - This file is a binary file containing 17 bytes to encode the instructions for function main.
  - If we want to view hello. o with a text editor, it would appear to be nonsense.

The Compilation System

- **Linking phase**
  - Notice that our hello program calls the printf function, which is part of the *standard C library* provided by every C compiler.
  - The printf function resides in a separate precompiled object file called printf.o, which must somehow be merged with our hello. o program.
  - The linker (ld) handles this merging.
  - The result is the *hello* file, which is an executable object file (or simply *executable)* that is ready to be loaded into memory and executed by the system.