ABSTRACTION ISN'T THE ENTIRE STORY

CS 045

Computer Organization and Architecture

Prof. Donald J. Patterson

Adapted from Bryant and O'Hallaron, Computer Systems:
A Programmer's Perspective, Third Edition

REALITY #4

THERE'S MORE TO PERFORMANCE THAN ASYMPTOTIC COMPLEXITY

- Constant factors matter too!
- And even exact op count does not predict performance
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- Must understand system to optimize performance
  - How programs compiled and executed
  - How to measure program performance and identify bottlenecks
- How to improve performance without destroying code modularity and generality

REALITY #4

MEMORY PERFORMANCE EXAMPLE

```
void copyij(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(i = 0; i < 1024; i++){
        for(j = 0; j < 1024; j++){
            dst[i][j] = src[i][j];
        }
    }
}
```

REALITY #4

MEMORY PERFORMANCE EXAMPLE

```
void copyij(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(i = 0; i < 1024; i++){
        for(j = 0; j < 1024; j++){
            dst[i][j] = src[i][j];
        }
    }
}
```

```
void copyji(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(j = 0; j < 1024; j++){
        for(i = 0; i < 1024; i++){
            dst[i][j] = src[i][j];
        }
    }
}
```

REALITY #4

MEMORY PERFORMANCE EXAMPLE

```
void copyij(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(i = 0; i < 1024; i++){
        for(j = 0; j < 1024; j++){
            dst[i][j] = src[i][j];
        }
    }
}
```

```
void copyji(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(j = 0; j < 1024; j++){
        for(i = 0; i < 1024; i++){
            dst[i][j] = src[i][j];
        }
    }
}
```

REALITY #4

MEMORY PERFORMANCE EXAMPLE

```
void copyij(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(i = 0; i < 1024; i++){
        for(j = 0; j < 1024; j++){
            dst[i][j] = src[i][j];
        }
    }
}
```

```
void copyji(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(j = 0; j < 1024; j++){
        for(i = 0; i < 1024; i++){
            dst[i][j] = src[i][j];
        }
    }
}
```

REALITY #4

MEMORY PERFORMANCE EXAMPLE

```
void copyij(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(i = 0; i < 1024; i++){
        for(j = 0; j < 1024; j++){
            dst[i][j] = src[i][j];
        }
    }
}
```

6ms

```
void copyji(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(j = 0; j < 1024; j++){
        for(i = 0; i < 1024; i++){
            dst[i][j] = src[i][j];
        }
    }
}
```

REALITY #4

MEMORY PERFORMANCE EXAMPLE

```
void copyij(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(i = 0; i < 1024; i++){
        for(j = 0; j < 1024; j++){
            dst[i][j] = src[i][j];
        }
    }
}
```

6ms

```
void copyji(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(j = 0; j < 1024; j++){
        for(i = 0; i < 1024; i++){
            dst[i][j] = src[i][j];
        }
    }
}
```

24ms

REALITY #4

MEMORY PERFORMANCE EXAMPLE

```
void copyij(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(i = 0; i < 1024; i++){
        for(j = 0; j < 1024; j++){
            dst[i][j] = src[i][j];
        }
    }
}
```

6ms

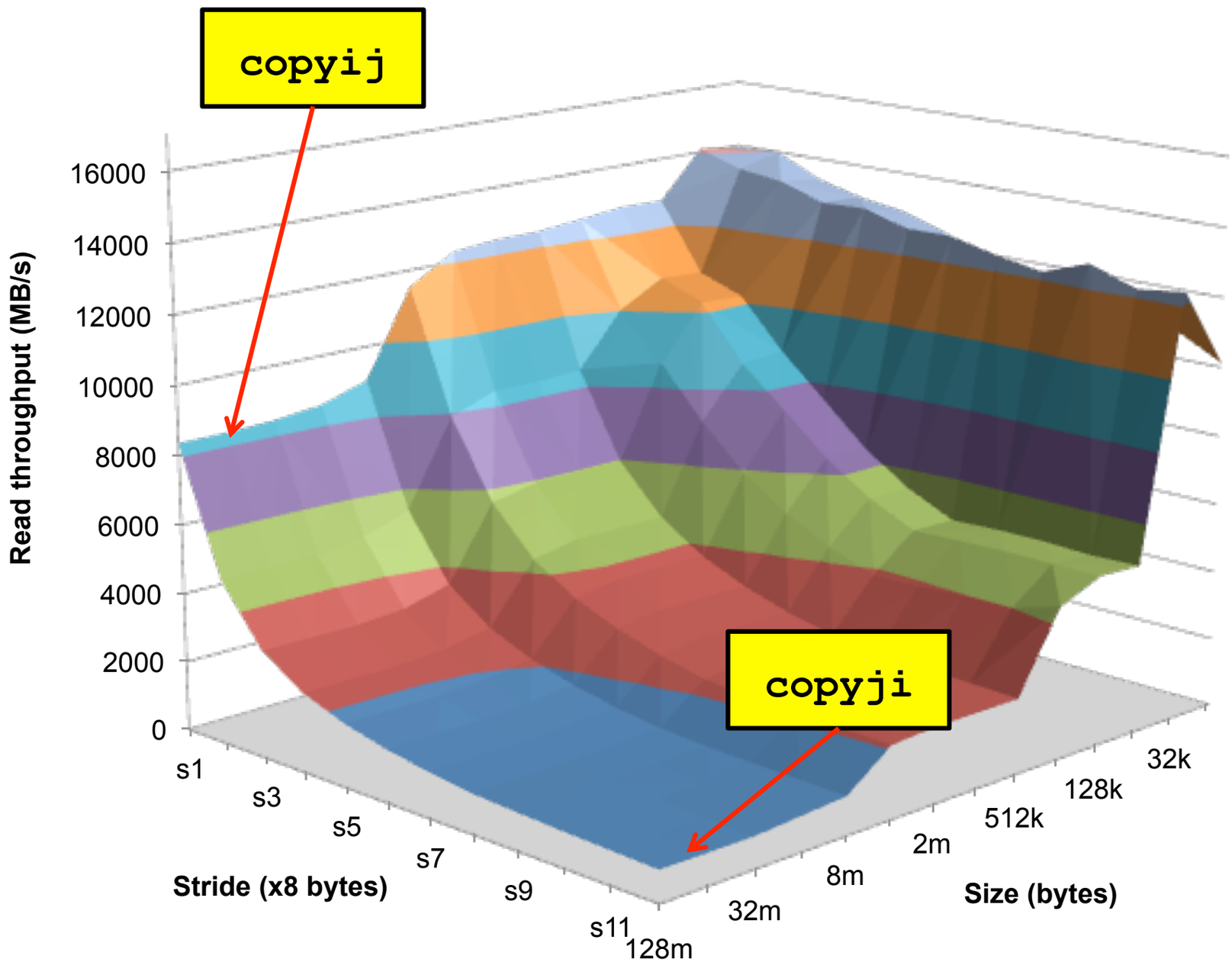
```
void copyji(int src[1024][1024],
            int dst[1024][1024]){
    int i,j;
    for(j = 0; j < 1024; j++){
        for(i = 0; i < 1024; i++){
            dst[i][j] = src[i][j];
        }
    }
}
```

24ms

- Hierarchical memory organization
  - Performance depends on access patterns
  - Including how step through multi-dimensional array

REALITY #4

WHY THE PERFORMANCE DIFFERS



REALITY #5

COURSE PERSPECTIVE

- Most Systems Courses are Builder-Centric
  - Computer Architecture
    - Design pipelined processor in Verilog
  - Operating Systems
    - Implement sample portions of operating system
  - Compilers
    - Write compiler for simple language
  - Networking
    - Implement and simulate network protocols

REALITY #5

COURSE PERSPECTIVE

- Our Course is Programmer-Centric
  - Purpose is to show that by knowing more about the underlying system, one can be more effective as a programmer
  - Enable you to
    - Write programs that are more reliable and efficient
      - Incorporate features that require hooks into OS
        - E.g., concurrency, signal handlers
  - Cover material in this course that you won't see elsewhere
  - Not just a course for dedicated hackers
    - We bring out the hidden hacker in everyone!

ABSTRACTION ISN'T THE ENTIRE STORY

OVERVIEW

- Course Theme
- Five Realities
- General System Walkthrough

ABSTRACTION ISN'T THE ENTIRE STORY

OVERVIEW

- Course Theme
- Five Realities
- General System Walkthrough

INFORMATION IS BITS + CONTEXT

HELLO WORLD

```
1    #include <stdio.h>
2
3    int main()
4    {
5        printf("hello, world\n");
6        return 0;
7    }
```

INFORMATION IS BITS + CONTEXT

HELLO WORLD

```
1   #include <stdio.h>
2
3   int main()
4   {
5       printf("hello, world\n");
6       return 0;
7   }
```

#	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	

INFOR

HELLO

#	i	n
35	105	110
h	>	\n
104	62	10
\n	*SP*	*SP*
10	32	32
l	o	,
108	111	44

Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	0000000	NUL	32	0x20	040	0100000	space	64	0x40	100	1000000	@	96	0x60	140	1100000	`
1	0x01	001	0000001	SOH	33	0x21	041	0100001	!	65	0x41	101	1000001	A	97	0x61	141	1100001	a
2	0x02	002	0000010	STX	34	0x22	042	0100010	"	66	0x42	102	1000010	B	98	0x62	142	1100010	b
3	0x03	003	0000011	ETX	35	0x23	043	0100011	#	67	0x43	103	1000011	C	99	0x63	143	1100011	c
4	0x04	004	0000100	EOT	36	0x24	044	0100100	$	68	0x44	104	1000100	D	100	0x64	144	1100100	d
5	0x05	005	0000101	ENQ	37	0x25	045	0100101	%	69	0x45	105	1000101	E	101	0x65	145	1100101	e
6	0x06	006	0000110	ACK	38	0x26	046	0100110	&	70	0x46	106	1000110	F	102	0x66	146	1100110	f
7	0x07	007	0000111	BEL	39	0x27	047	0100111	'	71	0x47	107	1000111	G	103	0x67	147	1100111	g
8	0x08	010	0001000	BS	40	0x28	050	0101000	(	72	0x48	110	1001000	H	104	0x68	150	1101000	h
9	0x09	011	0001001	TAB	41	0x29	051	0101001	)	73	0x49	111	1001001	I	105	0x69	151	1101001	i
10	0x0A	012	0001010	LF	42	0x2A	052	0101010	*	74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
11	0x0B	013	0001011	VT	43	0x2B	053	0101011	+	75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
12	0x0C	014	0001100	FF	44	0x2C	054	0101100	,	76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
13	0x0D	015	0001101	CR	45	0x2D	055	0101101	-	77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
14	0x0E	016	0001110	SO	46	0x2E	056	0101110	.	78	0x4E	116	1001110	N	110	0x6E	156	1101110	n
15	0x0F	017	0001111	SI	47	0x2F	057	0101111	/	79	0x4F	117	1001111	O	111	0x6F	157	1101111	o
16	0x10	020	0010000	DLE	48	0x30	060	0110000	0	80	0x50	120	1010000	P	112	0x70	160	1110000	p
17	0x11	021	0010001	DC1	49	0x31	061	0110001	1	81	0x51	121	1010001	Q	113	0x71	161	1110001	q
18	0x12	022	0010010	DC2	50	0x32	062	0110010	2	82	0x52	122	1010010	R	114	0x72	162	1110010	r
19	0x13	023	0010011	DC3	51	0x33	063	0110011	3	83	0x53	123	1010011	S	115	0x73	163	1110011	s
20	0x14	024	0010100	DC4	52	0x34	064	0110100	4	84	0x54	124	1010100	T	116	0x74	164	1110100	t
21	0x15	025	0010101	NAK	53	0x35	065	0110101	5	85	0x55	125	1010101	U	117	0x75	165	1110101	u
22	0x16	026	0010110	SYN	54	0x36	066	0110110	6	86	0x56	126	1010110	V	118	0x76	166	1110110	v
23	0x17	027	0010111	ETB	55	0x37	067	0110111	7	87	0x57	127	1010111	W	119	0x77	167	1110111	w
24	0x18	030	0011000	CAN	56	0x38	070	0111000	8	88	0x58	130	1011000	X	120	0x78	170	1111000	x
25	0x19	031	0011001	EM	57	0x39	071	0111001	9	89	0x59	131	1011001	Y	121	0x79	171	1111001	y
26	0x1A	032	0011010	SUB	58	0x3A	072	0111010	:	90	0x5A	132	1011010	Z	122	0x7A	172	1111010	z
27	0x1B	033	0011011	ESC	59	0x3B	073	0111011	;	91	0x5B	133	1011011	[	123	0x7B	173	1111011	{
28	0x1C	034	0011100	FS	60	0x3C	074	0111100	<	92	0x5C	134	1011100	\	124	0x7C	174	1111100	|
29	0x1D	035	0011101	GS	61	0x3D	075	0111101	=	93	0x5D	135	1011101	]	125	0x7D	175	1111101	}
30	0x1E	036	0011110	RS	62	0x3E	076	0111110	>	94	0x5E	136	1011110	^	126	0x7E	176	1111110	~
31	0x1F	037	0011111	US	63	0x3F	077	0111111	?	95	0x5F	137	1011111	_	127	0x7F	177	1111111	DEL

Take a moment to translate your first name into decimal and then binary

INFORMATION IS BITS + CONTEXT

HELLO WORLD

```c
#include <stdio.h>

int main()
{
    printf("hello, worl
    return 0;
}
```

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

Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	0000000	NUL	32	0x20	040	0100000	space	64	0x40	100	1000000	@	96	0x60	140	1100000	`
1	0x01	001	0000001	SOH	33	0x21	041	0100001	!	65	0x41	101	1000001	A	97	0x61	141	1100001	a
2	0x02	002	0000010	STX	34	0x22	042	0100010	"	66	0x42	102	1000010	B	98	0x62	142	1100010	b
3	0x03	003	0000011	ETX	35	0x23	043	0100011	#	67	0x43	103	1000011	C	99	0x63	143	1100011	c
4	0x04	004	0000100	EOT	36	0x24	044	0100100	$	68	0x44	104	1000100	D	100	0x64	144	1100100	d
5	0x05	005	0000101	ENQ	37	0x25	045	0100101	%	69	0x45	105	1000101	E	101	0x65	145	1100101	e
6	0x06	006	0000110	ACK	38	0x26	046	0100110	&	70	0x46	106	1000110	F	102	0x66	146	1100110	f
7	0x07	007	0000111	BEL	39	0x27	047	0100111	'	71	0x47	107	1000111	G	103	0x67	147	1100111	g
8	0x08	010	0001000	BS	40	0x28	050	0101000	(	72	0x48	110	1001000	H	104	0x68	150	1101000	h
9	0x09	011	0001001	TAB	41	0x29	051	0101001	)	73	0x49	111	1001001	I	105	0x69	151	1101001	i
10	0x0A	012	0001010	LF	42	0x2A	052	0101010	*	74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
11	0x0B	013	0001011	VT	43	0x2B	053	0101011	+	75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
12	0x0C	014	0001100	FF	44	0x2C	054	0101100	,	76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
13	0x0D	015	0001101	CR	45	0x2D	055	0101101	-	77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
14	0x0E	016	0001110	SO	46	0x2E	056	0101110	.	78	0x4E	116	1001110	N	110	0x6E	156	1101110	n
15	0x0F	017	0001111	SI	47	0x2F	057	0101111	/	79	0x4F	117	1001111	O	111	0x6F	157	1101111	o
16	0x10	020	0010000	DLE	48	0x30	060	0110000	0	80	0x50	120	1010000	P	112	0x70	160	1110000	p
17	0x11	021	0010001	DC1	49	0x31	061	0110001	1	81	0x51	121	1010001	Q	113	0x71	161	1110001	q
18	0x12	022	0010010	DC2	50	0x32	062	0110010	2	82	0x52	122	1010010	R	114	0x72	162	1110010	r
19	0x13	023	0010011	DC3	51	0x33	063	0110011	3	83	0x53	123	1010011	S	115	0x73	163	1110011	s
20	0x14	024	0010100	DC4	52	0x34	064	0110100	4	84	0x54	124	1010100	T	116	0x74	164	1110100	t
21	0x15	025	0010101	NAK	53	0x35	065	0110101	5	85	0x55	125	1010101	U	117	0x75	165	1110101	u
22	0x16	026	0010110	SYN	54	0x36	066	0110110	6	86	0x56	126	1010110	V	118	0x76	166	1110110	v
23	0x17	027	0010111	ETB	55	0x37	067	0110111	7	87	0x57	127	1010111	W	119	0x77	167	1110111	w
24	0x18	030	0011000	CAN	56	0x38	070	0111000	8	88	0x58	130	1011000	X	120	0x78	170	1111000	x
25	0x19	031	0011001	EM	57	0x39	071	0111001	9	89	0x59	131	1011001	Y	121	0x79	171	1111001	y
26	0x1A	032	0011010	SUB	58	0x3A	072	0111010	:	90	0x5A	132	1011010	Z	122	0x7A	172	1111010	z
27	0x1B	033	0011011	ESC	59	0x3B	073	0111011	;	91	0x5B	133	1011011	[	123	0x7B	173	1111011	{
28	0x1C	034	0011100	FS	60	0x3C	074	0111100	<	92	0x5C	134	1011100	\	124	0x7C	174	1111100	|
29	0x1D	035	0011101	GS	61	0x3D	075	0111101	=	93	0x5D	135	1011101	]	125	0x7D	175	1111101	}
30	0x1E	036	0011110	RS	62	0x3E	076	0111110	>	94	0x5E	136	1011110	^	126	0x7E	176	1111110	~
31	0x1F	037	0011111	US	63	0x3F	077	0111111	?	95	0x5F	137	1011111	_	127	0x7F	177	1111111	DEL

- Hello world is a source file
  - made with a text editor
  - saved as a text file (.c)
- The file is made up of bits (0 or 1)
  - 0's and 1's are grouped in bytes (8 bits in a row)
  - 1 byte can represent 256 different numbers
- In the source file each byte represents a text character
  - (International characters require multiple bytes)
- All files are made of 0's and 1's
  - Text files only have bytes that translate into characters
  - Binary files have bytes that don't translate into characters

INFORMATION IS BITS + CONTEXT

HELLO WORLD

```c
#include <stdio.h>

int main()
{
    printf("hello, worl
    return 0;
}
```

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

Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	0000000	NUL	32	0x20	040	0100000	space	64	0x40	100	1000000	@	96	0x60	140	1100000	`
1	0x01	001	0000001	SOH	33	0x21	041	0100001	!	65	0x41	101	1000001	A	97	0x61	141	1100001	a
2	0x02	002	0000010	STX	34	0x22	042	0100010	"	66	0x42	102	1000010	B	98	0x62	142	1100010	b
3	0x03	003	0000011	ETX	35	0x23	043	0100011	#	67	0x43	103	1000011	C	99	0x63	143	1100011	c
4	0x04	004	0000100	EOT	36	0x24	044	0100100	$	68	0x44	104	1000100	D	100	0x64	144	1100100	d
5	0x05	005	0000101	ENQ	37	0x25	045	0100101	%	69	0x45	105	1000101	E	101	0x65	145	1100101	e
6	0x06	006	0000110	ACK	38	0x26	046	0100110	&	70	0x46	106	1000110	F	102	0x66	146	1100110	f
7	0x07	007	0000111	BEL	39	0x27	047	0100111	'	71	0x47	107	1000111	G	103	0x67	147	1100111	g
8	0x08	010	0001000	BS	40	0x28	050	0101000	(	72	0x48	110	1001000	H	104	0x68	150	1101000	h
9	0x09	011	0001001	TAB	41	0x29	051	0101001	)	73	0x49	111	1001001	I	105	0x69	151	1101001	i
10	0x0A	012	0001010	LF	42	0x2A	052	0101010	*	74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
11	0x0B	013	0001011	VT	43	0x2B	053	0101011	+	75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
12	0x0C	014	0001100	FF	44	0x2C	054	0101100	,	76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
13	0x0D	015	0001101	CR	45	0x2D	055	0101101	-	77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
14	0x0E	016	0001110	SO	46	0x2E	056	0101110	.	78	0x4E	116	1001110	N	110	0x6E	156	1101110	n
15	0x0F	017	0001111	SI	47	0x2F	057	0101111	/	79	0x4F	117	1001111	O	111	0x6F	157	1101111	o
16	0x10	020	0010000	DLE	48	0x30	060	0110000	0	80	0x50	120	1010000	P	112	0x70	160	1110000	p
17	0x11	021	0010001	DC1	49	0x31	061	0110001	1	81	0x51	121	1010001	Q	113	0x71	161	1110001	q
18	0x12	022	0010010	DC2	50	0x32	062	0110010	2	82	0x52	122	1010010	R	114	0x72	162	1110010	r
19	0x13	023	0010011	DC3	51	0x33	063	0110011	3	83	0x53	123	1010011	S	115	0x73	163	1110011	s
20	0x14	024	0010100	DC4	52	0x34	064	0110100	4	84	0x54	124	1010100	T	116	0x74	164	1110100	t
21	0x15	025	0010101	NAK	53	0x35	065	0110101	5	85	0x55	125	1010101	U	117	0x75	165	1110101	u
22	0x16	026	0010110	SYN	54	0x36	066	0110110	6	86	0x56	126	1010110	V	118	0x76	166	1110110	v
23	0x17	027	0010111	ETB	55	0x37	067	0110111	7	87	0x57	127	1010111	W	119	0x77	167	1110111	w
24	0x18	030	0011000	CAN	56	0x38	070	0111000	8	88	0x58	130	1011000	X	120	0x78	170	1111000	x
25	0x19	031	0011001	EM	57	0x39	071	0111001	9	89	0x59	131	1011001	Y	121	0x79	171	1111001	y
26	0x1A	032	0011010	SUB	58	0x3A	072	0111010	:	90	0x5A	132	1011010	Z	122	0x7A	172	1111010	z
27	0x1B	033	0011011	ESC	59	0x3B	073	0111011	;	91	0x5B	133	1011011	[	123	0x7B	173	1111011	{
28	0x1C	034	0011100	FS	60	0x3C	074	0111100	<	92	0x5C	134	1011100	\	124	0x7C	174	1111100	|
29	0x1D	035	0011101	GS	61	0x3D	075	0111101	=	93	0x5D	135	1011101	]	125	0x7D	175	1111101	}
30	0x1E	036	0011110	RS	62	0x3E	076	0111110	>	94	0x5E	136	1011110	^	126	0x7E	176	1111110	~
31	0x1F	037	0011111	US	63	0x3F	077	0111111	?	95	0x5F	137	1011111	_	127	0x7F	177	1111111	DEL

- All information in a computer is 0's and 1's
  - "bits"
- How we interpret the bits determine if is a character, an integer, a float, a sound, an image, a program, etc. How the bits are interpreted depend on
  - "context"

INFORMATION IS BITS + CONTEXT

HELLO WORLD

```
1   #include <stdio.h>
2
3   int main()
4   {
5       printf("hello, world\n");
6       return 0;
7   }
```

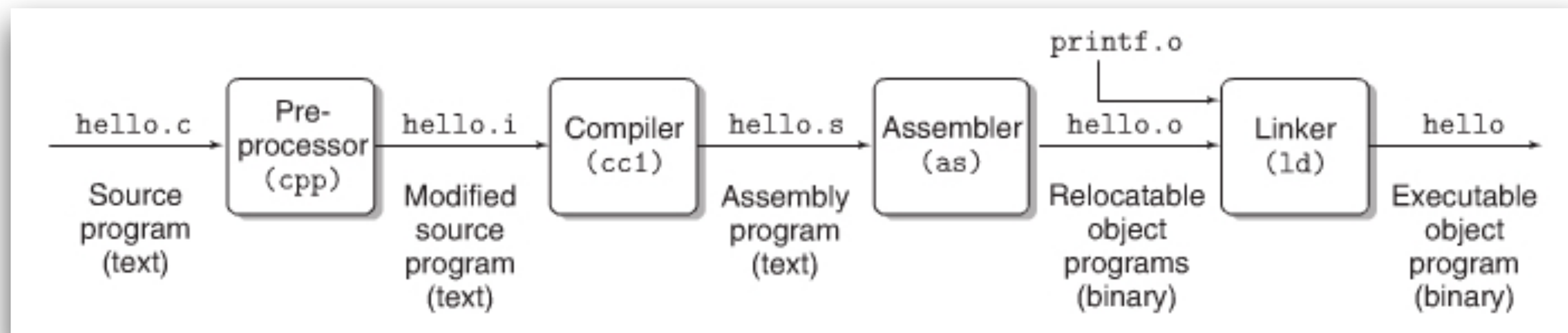
- For the bits in a source file to make sense as a program it must be translated into machine language in a multi-stage process

INFORMATION IS BITS + CONTEXT

HELLO WORLD

```
1   #include <stdio.h>
2
3   int main()
4   {
5       printf("hello, world\n");
6       return 0;
7   }
```

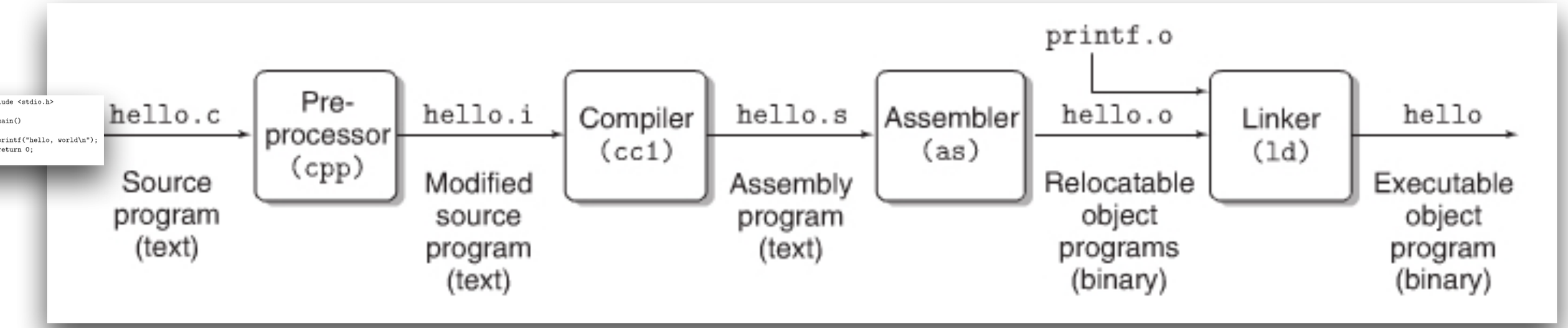
- For the bits in a source file to make sense as a program it must be translated into machine language in a multi-stage process



INFORMATION IS BITS + CONTEXT

HELLO WORLD

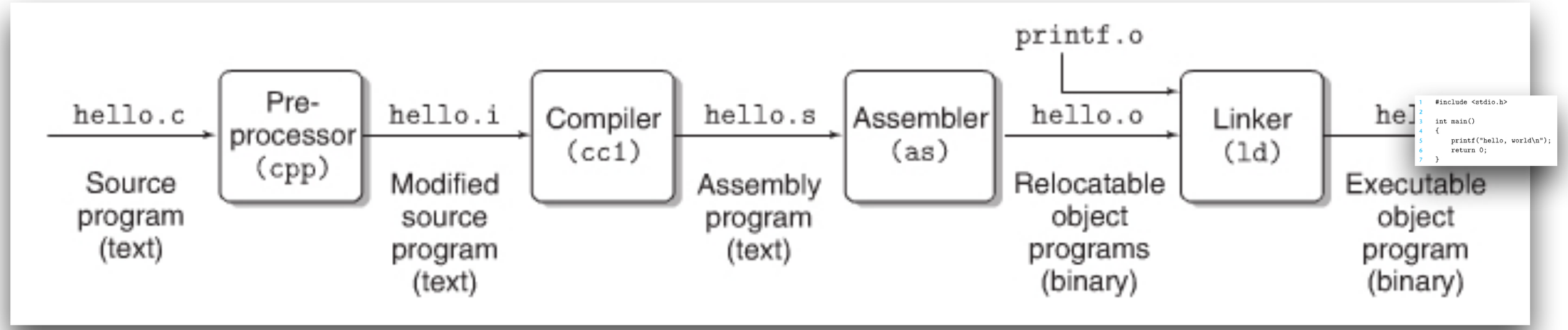
- For the bits in a source file to make sense as a program it must be translated into machine language in a multi-stage process

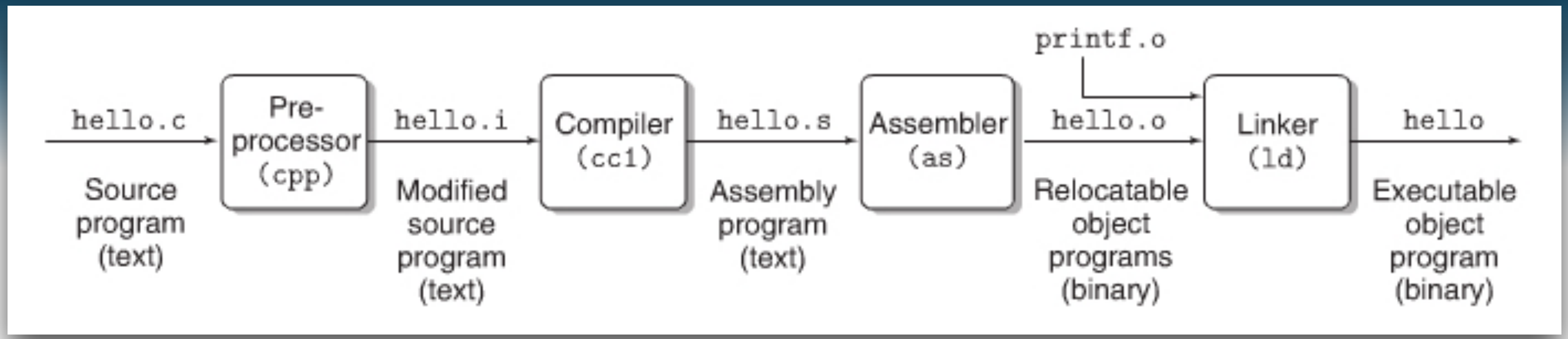


INFORMATION IS BITS + CONTEXT

HELLO WORLD

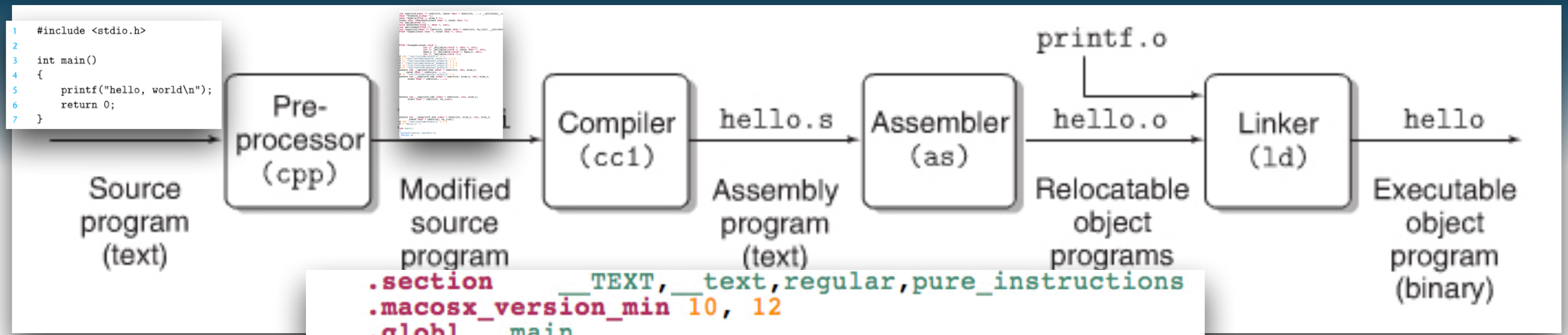
- For the bits in a source file to make sense as a program it must be translated into machine language in a multi-stage process



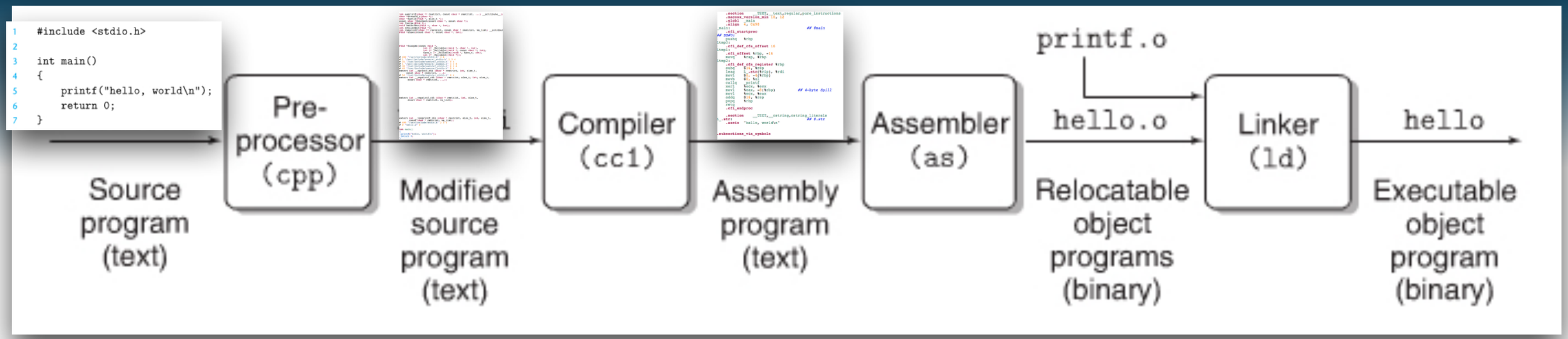



```c
#include <stdio.h>

int main()
{
    printf("hello, world\n");
    return 0;
}
```

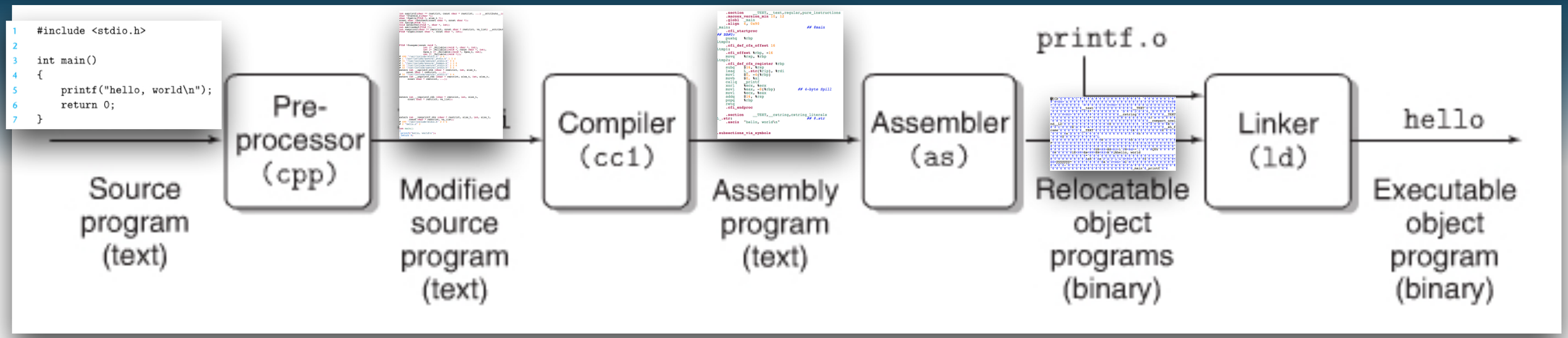
```
# 498 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/secure/_stdio.h" 1 3 4
# 31 "/usr/include/secure/_stdio.h" 3 4
# 1 "/usr/include/secure/_common.h" 1 3 4
# 32 "/usr/include/secure/_stdio.h" 2 3 4
# 42 "/usr/include/secure/_stdio.h" 3 4
extern int __sprintf_chk (char * restrict, int, size_t,
      const char * restrict, ...);
# 52 "/usr/include/secure/_stdio.h" 3 4
extern int __snprintf_chk (char * restrict, size_t, int, size_t,
      const char * restrict, ...);




extern int __vsprintf_chk (char * restrict, int, size_t,
      const char * restrict, va_list);




extern int __vsnprintf_chk (char * restrict, size_t, int, size_t,
      const char * restrict, va_list);
# 499 "/usr/include/stdio.h" 2 3 4
# 2 "hello.c" 2

int main()
{
 printf("hello, world\n");
 return 0;
}
```

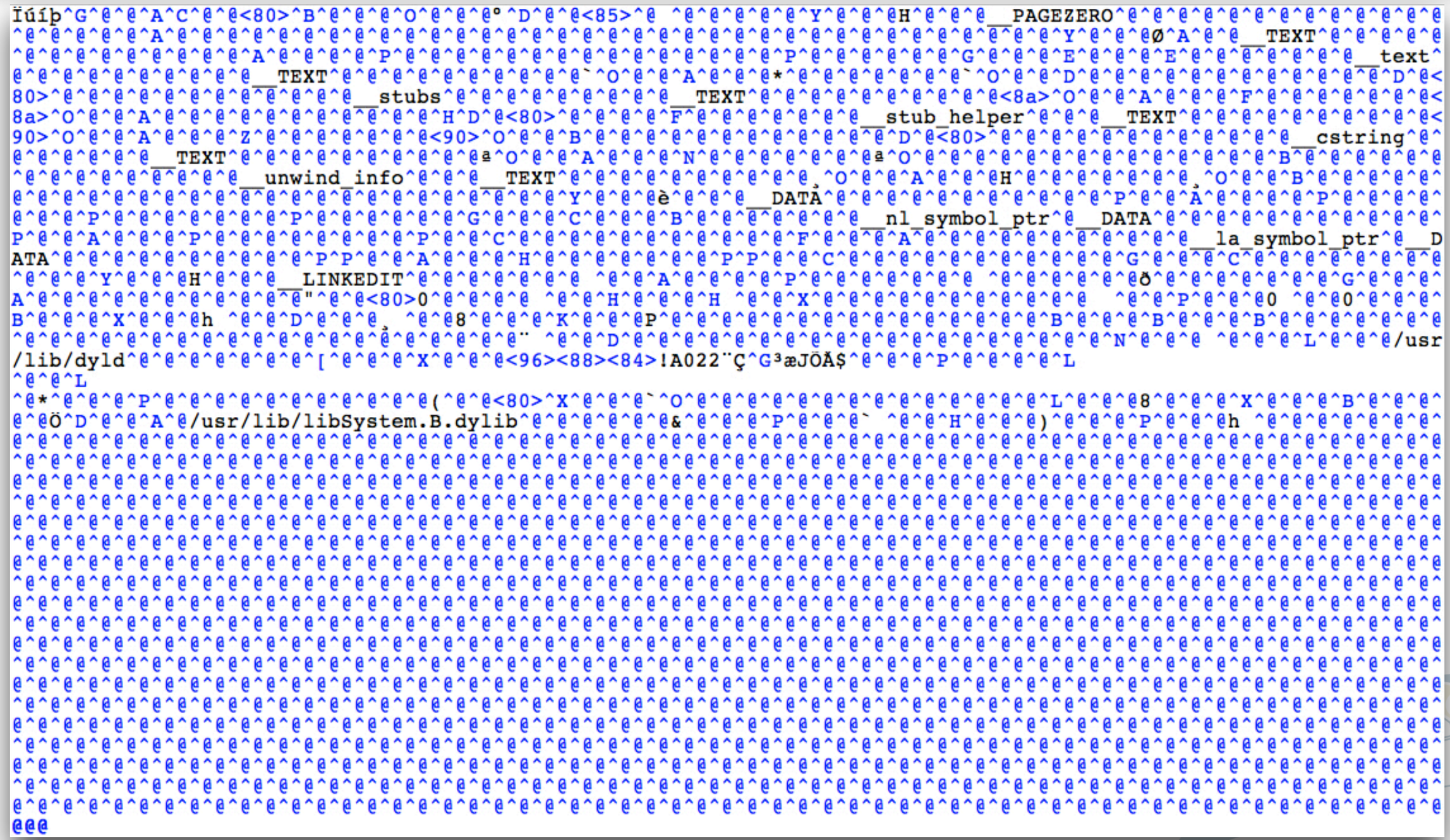


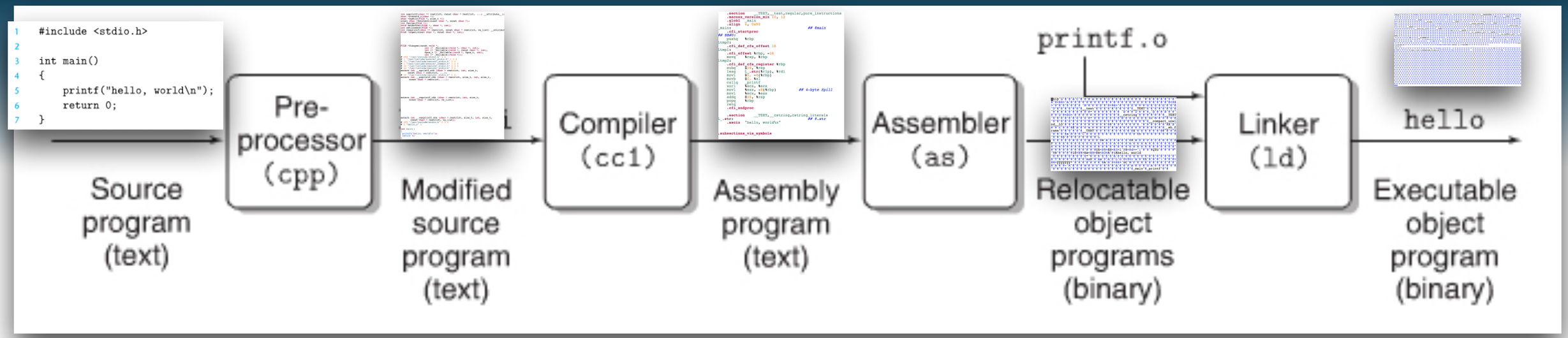

```
    .section    __TEXT,__text,regular,pure_instructions
    .macosx_version_min 10, 12
    .globl  _main
    .align  4, 0x90
_main:                                  ## @main
    .cfi_startproc
## BB#0:
    pushq   %rbp
Ltmp0:
    .cfi_def_cfa_offset 16
Ltmp1:
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
Ltmp2:
    .cfi_def_cfa_register %rbp
    subq    $16, %rsp
    leaq    L_.str(%rip), %rdi
    movl    $0, -4(%rbp)
    movb    $0, %al
    callq   _printf
    xorl    %ecx, %ecx
    movl    %eax, -8(%rbp)          ## 4-byte Spill
    movl    %ecx, %eax
    addq    $16, %rsp
    popq    %rbp
    retq
    .cfi_endproc

    .section    __TEXT,__cstring,cstring_literals
L_.str:                                 ## @.str
    .asciz  "hello, world\n"


.subsections_via_symbols
```



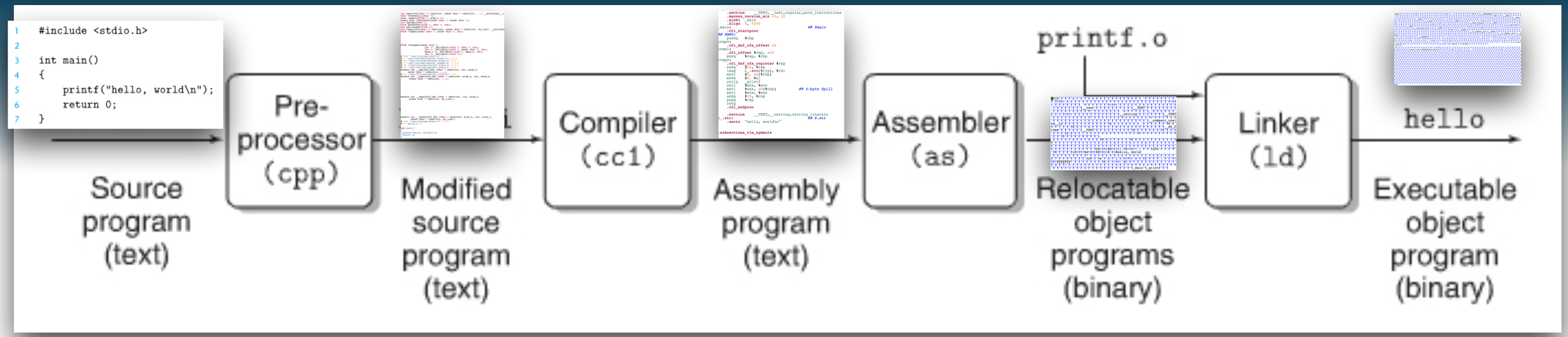

```
Îúíþ^G^@^@^A^C^@^@^@^A^@^@^@^D^@^@^@^@^B^@^@^@ ^@^@^@^@^@^@^Y^@^
@^@<88>^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@<98
>^@^@^@^@^@^@^@ ^B^@^@^@^@^@^@<98>^@^@^@^@^@^@^@^@^G^@^@^@^G^@^@^@
^D^@^@^@^@^@^@^@__text^@^@^@^@^@^@^@^@^@^@__TEXT^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@*^@^@^@^@^@^@^@ ^B^@^@^D^@^@^@ ^B^@^@^B^@^@^
@^@^D^@<80>^@^@^@^@^@^@^@^@^@^@^@^@__cstring^@^@^@^@^@^@^@__TEXT
^@^@^@^@^@^@^@^@^@^@*^@^@^@^@^@^@^@^N^@^@^@^@^@^@^@J^B^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@__compact_unwi
nd__LD^@^@^@^@^@^@^@^@^@^@^@^@^@8^@^@^@^@^@^@^@ ^@^@^@^@^@^@^@^@X^B^
@^@^C^@^@^@È^B^@^@^A^@^@^@^@^@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@__eh_f
rame^@^@^@^@^@^@__TEXT^@^@^@^@^@^@^@^@^@^@X^@^@^@^@^@^@^@@^@^@^@
^@^@^@^@^@x^B^@^@^C^@^@^@^@^@^@^@^@^@^@^@^@^K^@^@h^@^@^@^@^@^@^@^@^@
^@^@^@$^@^@^@^P^@^@^@^@^L
^@^@^@^@^@^@^B^@^@^@^X^@^@^@Ð^B^@^@^B^@^@^@ð^B^@^@^P^@^@^@^K^@^@^@
P^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^A^@^@^@^A^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@UH<89>åH<83>ì^PH<8d>=^[^@^@^@ÇEü^@^@^@^@º
^@è^@^@^@^@1É<89>Eø<89>ÈH<83>Ä^P]Ähello, world
^@^@^@^@^@^@^@^@^@^@*^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^T^@^@^@^@^@^@^@^@^AzR^@^Ax^P^A^P^L^G^H<90>^A^@^@$^@^@^@^\^@^@^@<
88>ÿÿÿÿÿÿÿÿ*^@^@^@^@^@^@^@^@^@A^N^P<86>^BC^M^F^@^@^@^@^@^@^@^Y^@^@^
@^A^@^@-^K^@^@^@^B^@^@^U^@^@^@^@^@^A^@^@^F^A^@^@^@^O^A^@^@^@^@^@^@
^@^@^@^@^@^G^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@_main^@_printf^@^@
```


#include <stdio.h>
int main()
{
printf("hello, world\n");
return 0;
}
Source program (text)
Pre-processor (cpp)
Modified source program (text)
Compiler (cc1)
Assembly program (text)
Assembler (as)
printf.o
Relocatable object programs (binary)
Linker (ld)
hello
Executable object program (binary)


__PAGEZERO
__TEXT
__text
__stubs
__stub_helper
__cstring
__unwind_info
__DATA
__nl_symbol_ptr
__la_symbol_ptr
__LINKEDIT
/usr/lib/dyld
/usr/lib/libSystem.B.dylib




```
[562]djp3@Codex-Perductum: ~/Documents/ClassResources/2017_01_CS045/website/
Lectures/Lecture_02
[$ ls
SourceGraphics.graffle/  hello.i  memory_bug.c          precision2.c
hello*                   hello.o  memory_performance.c  source_hello.sh
hello.c                  hello.s  precision1.c
[562]djp3@Codex-Perductum: ~/Documents/ClassResources/2017_01_CS045/website/
Lectures/Lecture_02
[$ ./hello
hello, world
```




```
#Run just the C-preprocess hello.c -> hello.i
gcc -Wall -E -o hello.i hello.c

#Run to the assembly code hello.c -> hello.s
gcc -Wall -S -o hello.s hello.c

#Run to the object code hello.c -> hello.o
gcc -Wall -c -o hello.o hello.c

#Run the whole tool chain and remove the intermediate files
gcc -Wall -o hello hello.c
```

LET'S GET IT TO WORK

TASKS

- Run a hello world program on wcpkneel
- Output your name on wcpkneel
- Output the values of the characters of your name on wcpkneel

LET'S GET IT TO WORK

OUTCOMES

- I expect that after today,
  - you don't have problems logging on to wcpkneel with ssh
  - that you can write at least one C program on wcpkneel
  - that you can compile a C program on wcpkneel
  - that you can run a program on wcpkneel
  - that you understand that you can interpret bits as a character or the same bits as an integer and the result is different

WESTMONT INSPIRED
COMPUTING LAB