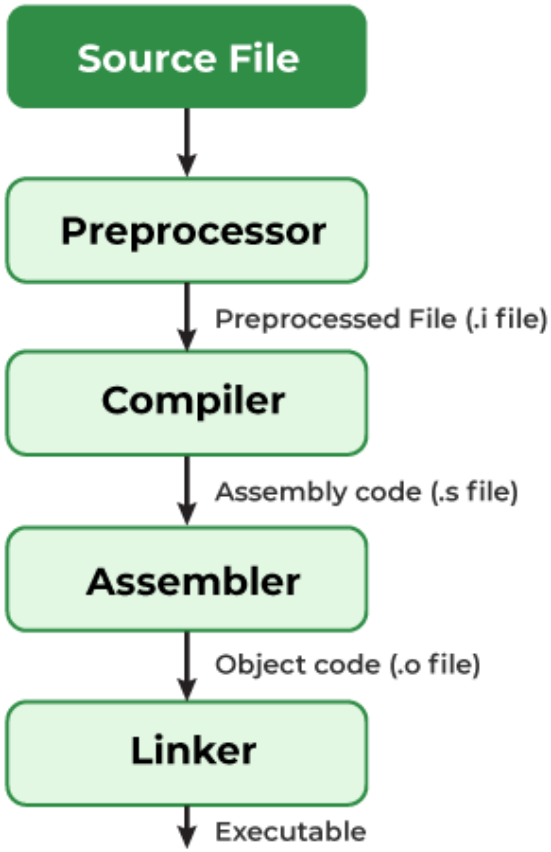
Fasi della Compilazione

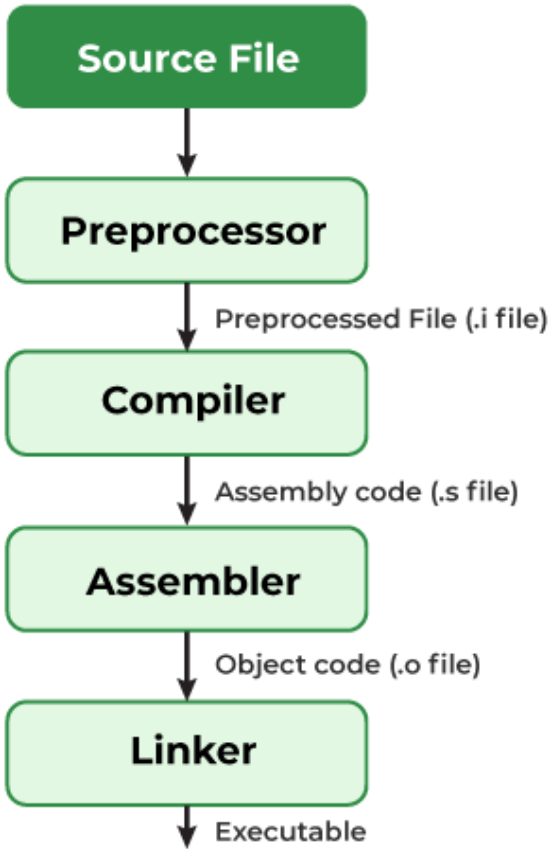
Trasformare C in Codice Macchina

Fasi compilazione




```
$gcc -Wall -save-temps filename.c -o filename
```

Fasi compilazione




```
$gcc -Wall -save-temps filename.c –o filename
```

Pre-processore produce file .i:

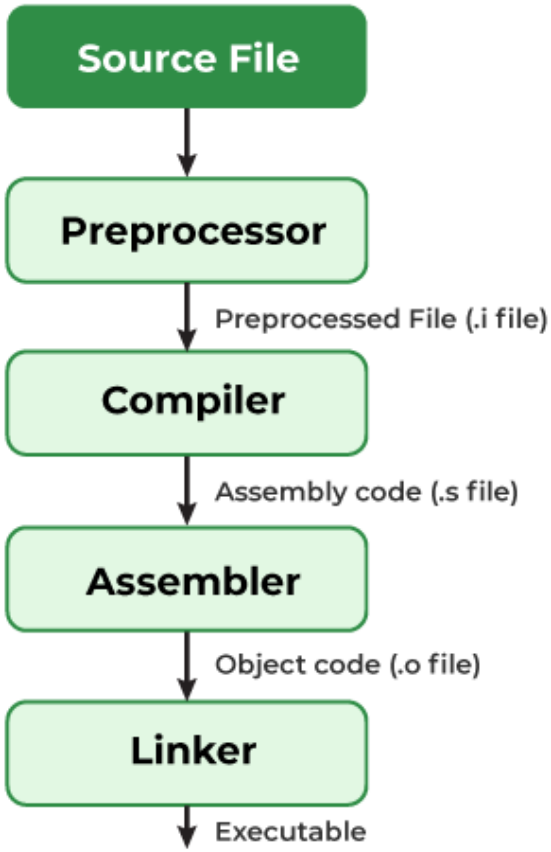
- rimuove commenti
- Espande le macro
- Espande file inclusi

Esempio output.i:

```
#include<stdio.h> manca, ora vediamo molto codice al
suo posto (solo header)
```

```
#define PI -> sostituito
```

Fasi compilazione




```
$gcc -Wall -save-temps filename.c –o filename
```

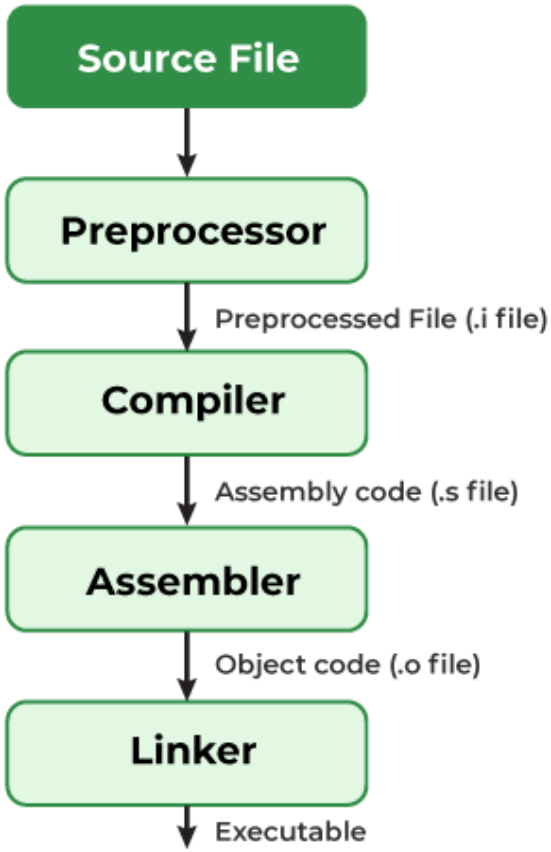
Compilatore produce file .s:

- File intermedio con istruzioni assembly

Esempio output.s:

```
…
stp         x29, x30, [sp, #32]             ; 16-byte Folded Spill
            add         x29, sp, #32
            .cfi_def_cfa w29, 16
            .cfi_offset w30, -8
            .cfi_offset w29, -16
            mov         w8, #0                          ; =0x0
            str         w8, [sp, #16]                   ; 4-byte Folded Spill
            stur        wzr, [x29, #-4]
            mov         w8, #10                         ; =0xa
            stur        w8, [x29, #-8]
            mov         w8, #4                          ; =0x4
…
```

Fasi compilazione




```
$gcc -Wall -save-temps filename.c –o filename
```

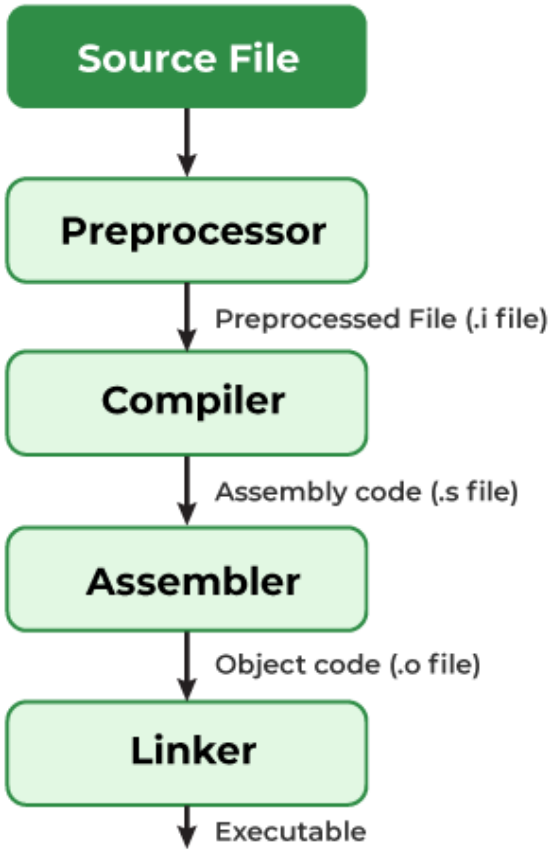
Assembler produce file .o:

- Istruzioni linguaggio macchina
- Solo il codice esistente, funzioni di libreria non ancora risolte (e.g., funzioni usate della libreria stdio.h)

Esempio output.o:

```
cffa edfe 0c00 0001 0000 0000 0100 0000
0400 0000 0802 0000 0020 0000 0000 0000
1900 0000 8801 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
f800 0000 0000 0000 2802 0000 0000 0000
f800 0000 0000 0000 0700 0000 0700 0000
0400 0000 0000 0000 5f5f 7465 7874 0000
0000 0000 0000 0000 5f5f 5445 5854 0000
```

Fasi compilazione




```
$gcc -Wall -save-temps filename.c –o filename
```

Linker produce file eseguibile:

- Fase finale in cui le librerie usate sono effettivamnete incluse. Il Linker sa dove sono le librerie e le definizioni delle funzioni usate.
- Aggiunge codice extra per configurare ambiente (esempio passaggio argomenti da linea di commando)

Se verifichiamo con size, il file .o e il file eseguibile hanno dimensioni differenti.

Linux size command

Size (without the -m option) prints the (decimal) number of bytes required by the __TEXT, __DATA and __OBJC segments. All other segments are totaled and that size is listed in the `others' column.

The final two columns is the sum in decimal and hexadecimal. If no file is specified, a.out is used.

```
(base) cristina@Mac behindTheScene % size -m main.o
Segment : 248
        Section (__TEXT, __text): 152
        Section (__TEXT, __literal8): 8
        Section (__TEXT, __cstring): 22
        Section (__LD, __compact_unwind): 64
        total 246
total 248
(base) cristina@Mac behindTheScene % size -m main
Segment __PAGEZERO: 4294967296 (zero fill)
Segment __TEXT: 16384
        Section __text: 152
        Section __stubs: 12
        Section __const: 8
        Section __cstring: 22
        Section __unwind_info: 96
        total 290
Segment __DATA_CONST: 16384
        Section __got: 8
        total 8
Segment __LINKEDIT: 16384
total 4295016448
```