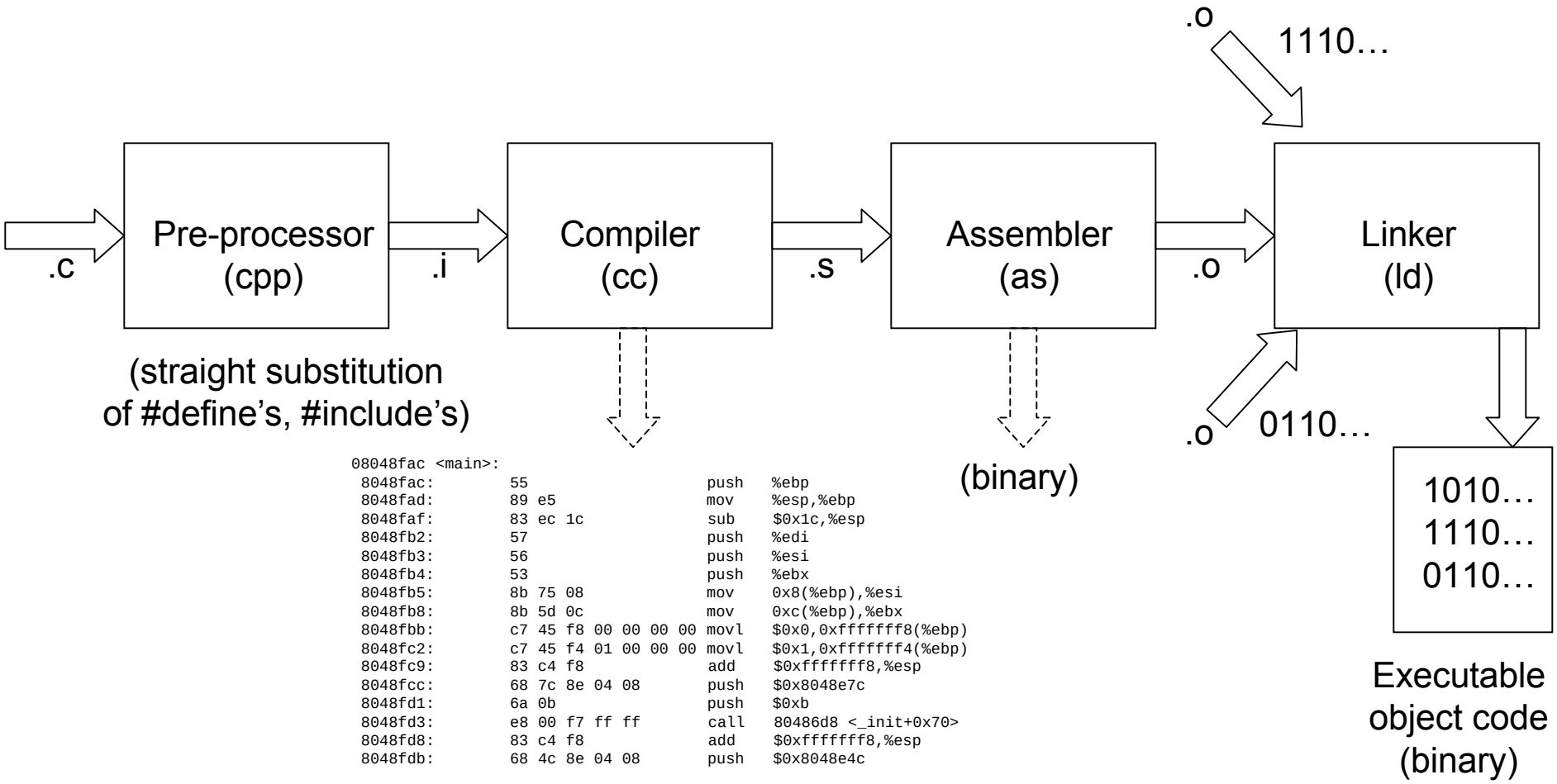
The Compilation Process




```
08048fac <main>:
 8048fac:       55                      push   %ebp
 8048fad:       89 e5                   mov    %esp,%ebp
 8048faf:       83 ec 1c                sub    $0x1c,%esp
 8048fb2:       57                      push   %edi
 8048fb3:       56                      push   %esi
 8048fb4:       53                      push   %ebx
 8048fb5:       8b 75 08                mov    0x8(%ebp),%esi
 8048fb8:       8b 5d 0c                mov    0xc(%ebp),%ebx
 8048fbb:       c7 45 f8 00 00 00 00    movl   $0x0,0xfffffff8(%ebp)
 8048fc2:       c7 45 f4 01 00 00 00    movl   $0x1,0xfffffff4(%ebp)
 8048fc9:       83 c4 f8                add    $0xfffffff8,%esp
 8048fcc:       68 7c 8e 04 08          push   $0x8048e7c
 8048fd1:       6a 0b                   push   $0xb
 8048fd3:       e8 00 f7 ff ff          call   80486d8 <_init+0x70>
 8048fd8:       83 c4 f8                add    $0xfffffff8,%esp
 8048fdb:       68 4c 8e 04 08          push   $0x8048e4c
```

Making Code More Modular

Pre-processor

```
#include <stdlib.h>
#include <stdio.h>

#define  WORD_LENGTH  30
```

Prototypes

```
int insert(char **array, int *count, char *word);
void remove(char **array, int *count, int location);
int saveFile(char *outFile, char **array, int count);
```

Create a header file for related functions

Main Function

```
int main(int argc, char *argv[])
{
  ...
}
```

Leave in main .c file

Additional Functions

```
int insert(char **array, int *count, char *word)
{
  ...
}
```

Place in separate .c file

```
int saveFile(char *outFile, char **array, int count)
{
  ...
}
```

Making Code More Modular

labX.c

```
#include <stdlib.h>
#include <stdio.h>
#include “fileIO.h”
#include “dictlib.h”

#define  WORD_LENGTH  30

int
main (int argc, char *argv[])
{
  ...
}
```

dictlib.h

```
int insert(char **array, int *count, char *word);
void remove(char **array, int *count, int location);
```

fileIO.h

```
int saveFile(char *outFile, char **array, int count);
```

dictlib.c

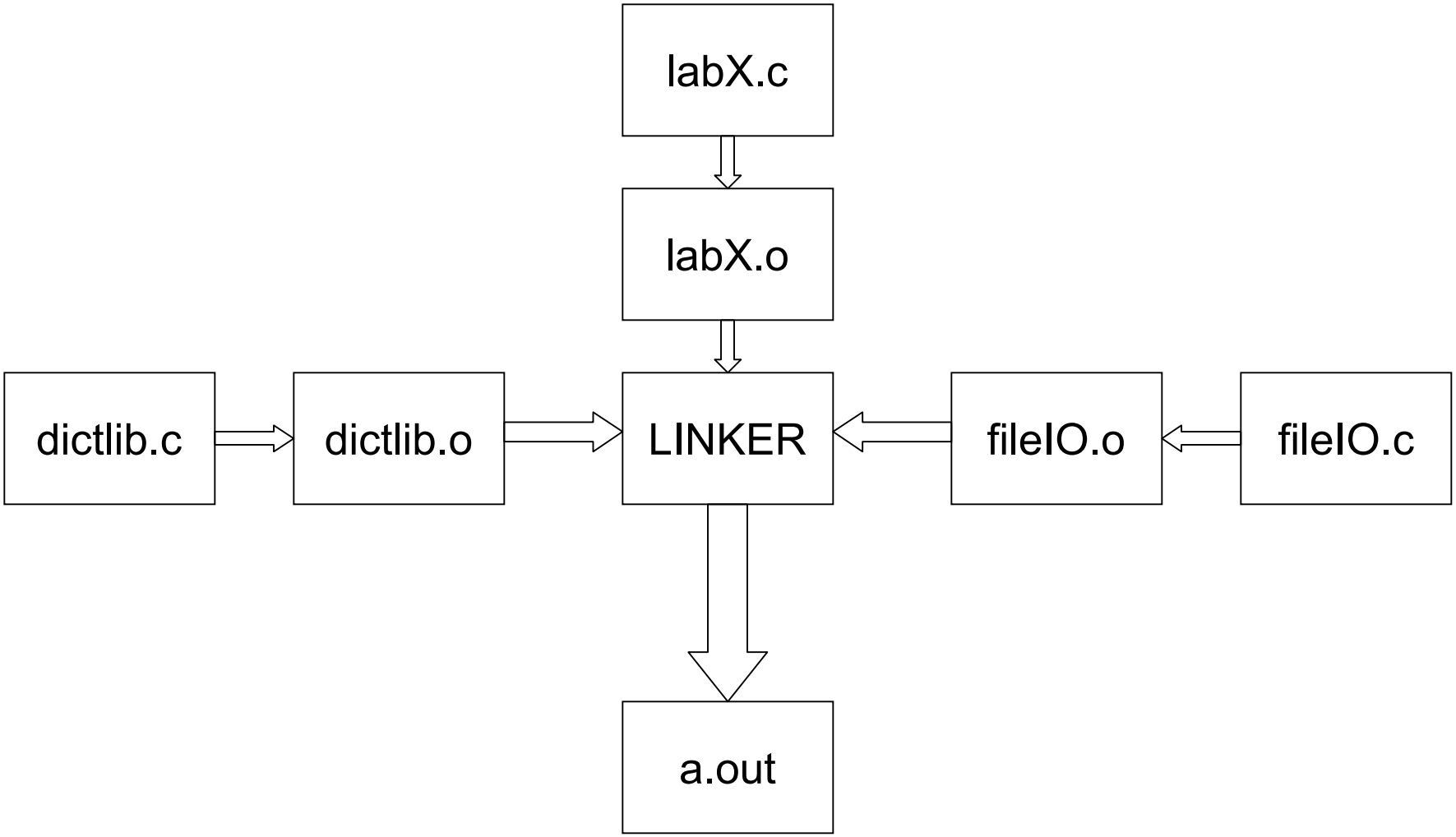
```
int insert(char **array, int *count, char
    *word)
{
  ...
}

void remove(char **array, int *count, int
    location)
{
  ...
}
```

fileIO.c

```
int saveFile(char *outFile, char **array,
    int count)
{
  ...
}
```

Putting the Modules Together



Confused??? Enter make…

- rtfm (i.e., man make)
- Located in the same directory with your code
- File (usually) named *Makefile*
- Syntax
  - Comments:

    any line that begins with #

    # anything on the line after the # is ignored by make
  - Macros:

    NAME = data

    $(NAME) is replaced by data

    CC = gcc # CC has special meaning

    CFLAGS = –Wall –pedantic –g # as does CFLAGS

    SRC = labX.c fileIO.c dictlib.c

Syntax continued…

- Explicit Rules (which files depend on others and the commands required to successfully compile them):

  target : dependent files

  <tab> command # the <tab> is REALLY important!

  <tab> command

  labX.exe : labX.c dictlib.c fileIO.c dictlib.h fileIO.h

  gcc –o labX.exe labX.c dictlib.c fileIO.c

  means that in order to create the *labX.exe* target, those 5 files need to be in the directory and *make* should use the command *gcc –o labX.exe labX.c dictlib.c fileIO.c* to create it

Syntax continued…

- Implicit Rules (same idea as explicit rules, but without an explicit command list; *make* does the right thing depending on the extension of the file):

  target : dependent files

  dictlib.o : dictlib.c dictlib.h fileIO.h

  means that in order to create the *dictlib.o* target, *make* will execute the command $(CC) $CFLAGS –*c dictlib.c*