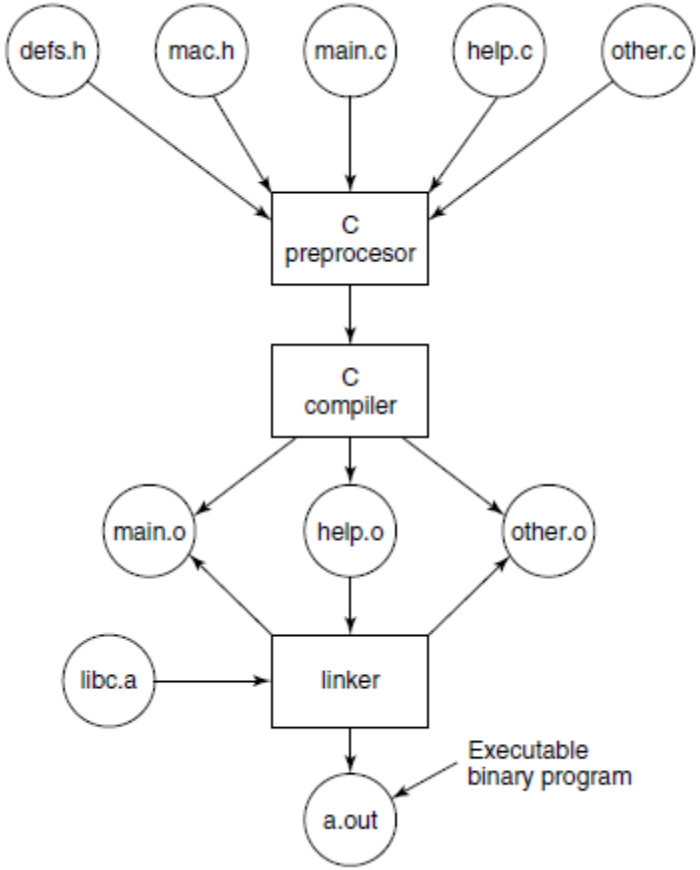
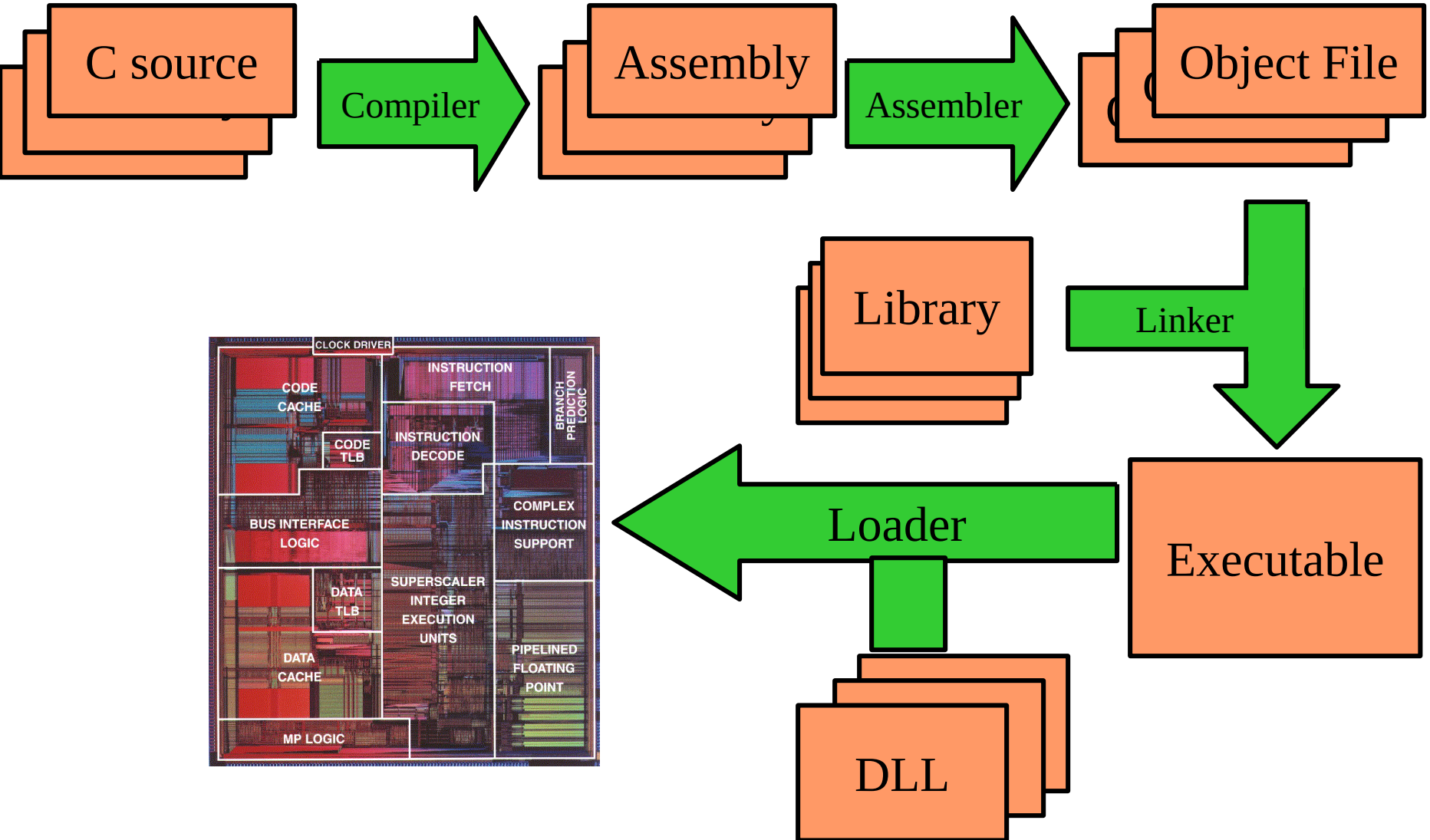
LAB assignment #1

Large Programming Projects



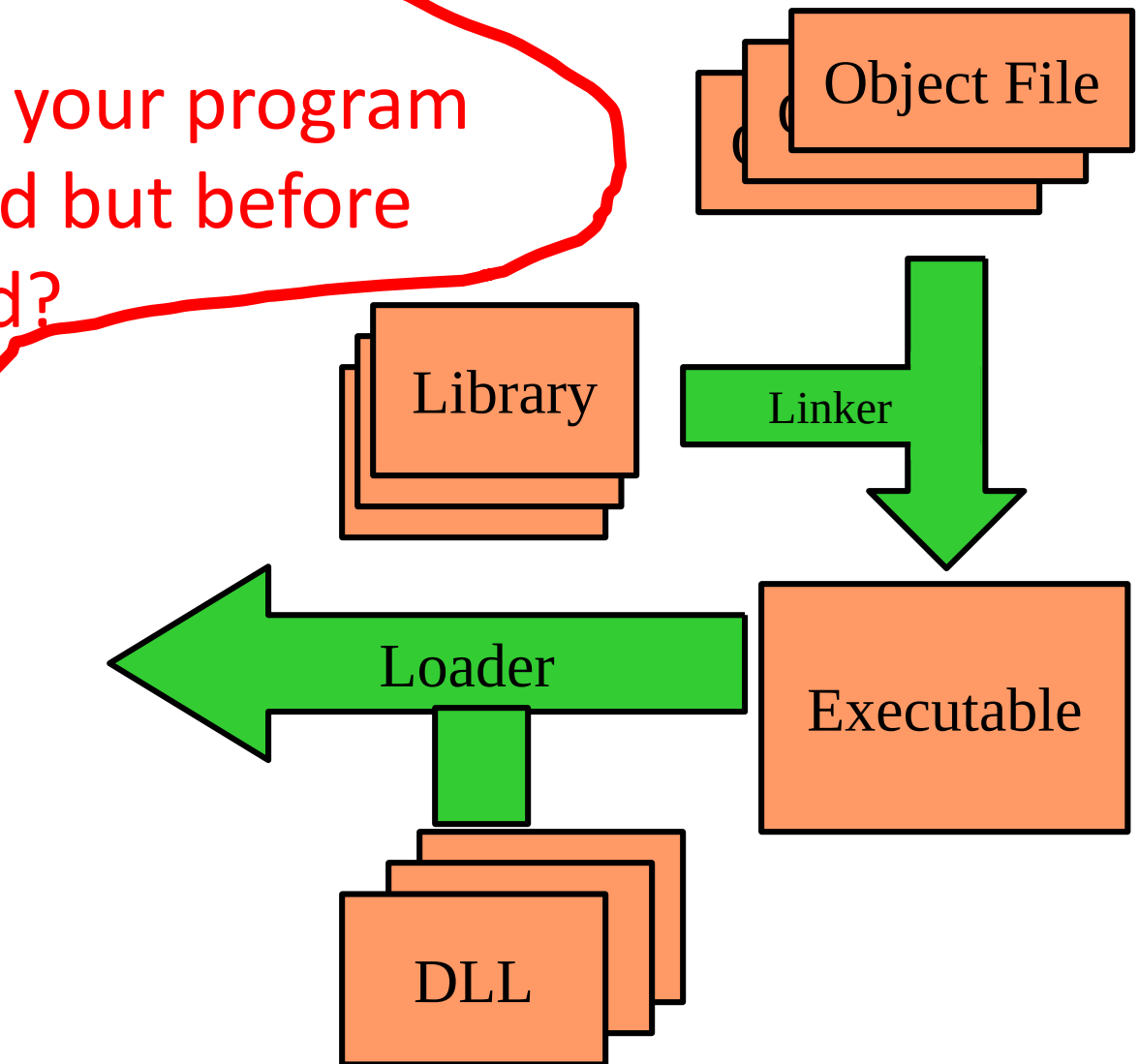
The process of compiling C and header files to make an executable.

Source Code to Execution



Source Code to Execution

What happens to your program after it is compiled but before it can be executed?



The OS Expectation

- The OS expects executable files to have a specific format
  - *Header info*
    - Code locations and size
    - Data locations and size
  - Code & data
  - *Symbol Table*
    - List of names of things defined in your program and where they are defined
    - List of names of things defined elsewhere that are used by your program, and where they are used.

Example of Things

```
#include <stdio.h>
extern int errno;

int main () {

  printf ("hello,
  world\n")

  <check errno for
  errors>
}
```

- Symbol defined in your program and used elsewhere
  - `main`
- Symbol defined elsewhere and used by your program
  - `printf`
  - `errno`

Two Steps Operation: Parts of OS

Linking

- Stitches independently created object files into a single executable file (i.e., a.out)
- Resolves cross-file references to labels
- Listing symbols needing to be resolved by loader

Loading

- copying a program image from hard disk to the main memory in order to put the program in a ready-to-run state
- Maps addresses within file to physical memory addresses
- Resolves names of dynamic library items
- schedule program as a new process

A Bit About Relocation

- modifies the object program so that it can be loaded at an address different from the location originally specified
- The compiler and assembler (mistakenly) treat each module as if it will be loaded at location zero

(e.g. ***jump 120***
is used to indicate a jump to location 120 of the current module)

A Bit About Relocation

- To convert this relative address to an absolute address, the linker adds the base address of the module to the relative address.
- The base address is the address at which this module will be loaded.

Example: Module A is to be loaded starting at location 2300 and contains the instruction

jump 120

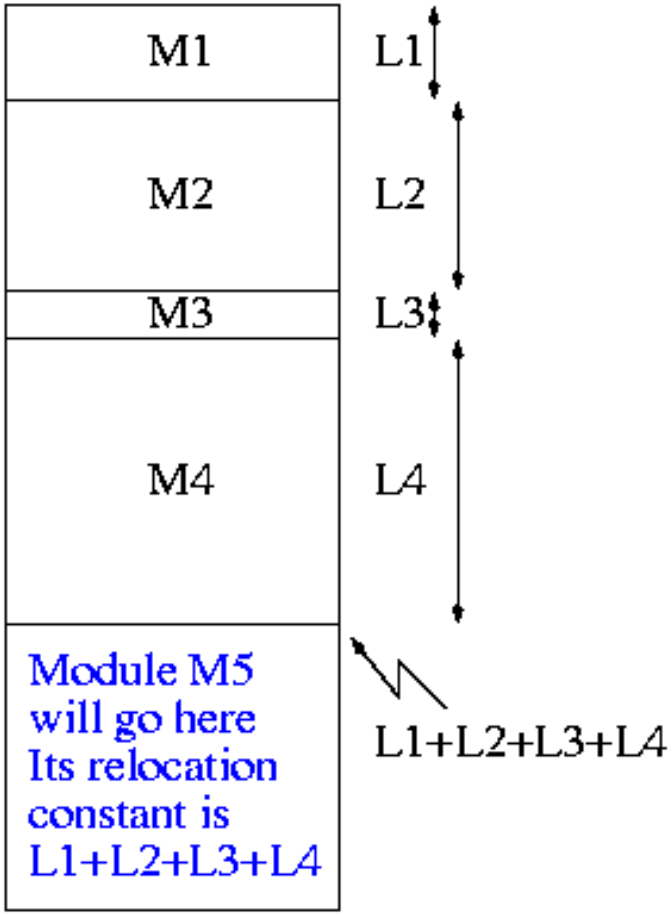
The linker changes this instruction to

jump 2420

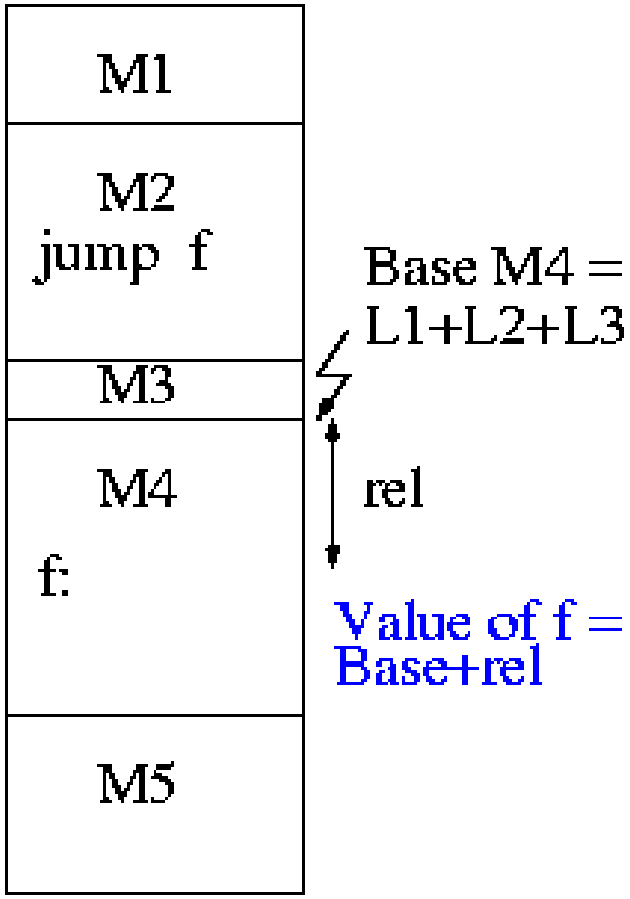
A Bit About Relocation

- How does the linker know that Module A is to be loaded starting at location 2300?
  - It processes the modules one at a time. The first module is to be loaded at location zero. So relocating the first module is trivial (adding zero). We say that the relocation constant is zero.
  - After processing the first module, the linker knows its length (say that length is L1).
  - Hence the next module is to be loaded starting at L1, i.e., the relocation constant is L1.
  - In general the linker keeps the sum of the lengths of all the modules it has already processed; this sum is the relocation constant for the next module.

A Bit About Relocation



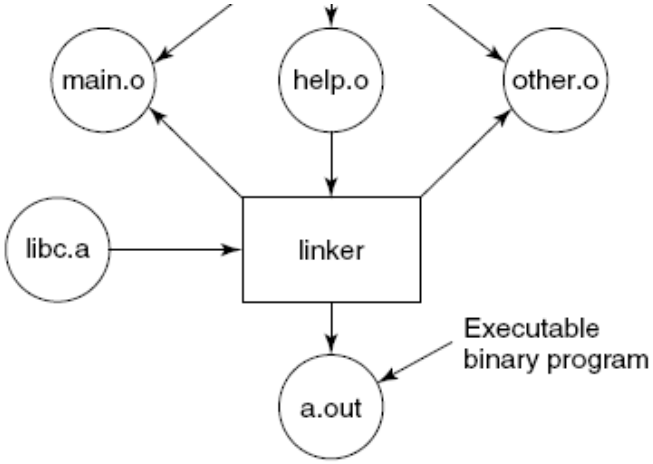
A Bit About Relocation



LAB #1: Write a Linker

- Link "==merge" together multiple parts of a program

- What problem is solved?
  - External references need to be resolved
  - Module relative addressing needs to be fixed



Cygwin

```
289:~/NYU/lab1/example>cat print.c
#include <stdio.h>

void print_hello()
{
     printf("Hello world\n");
}
```

Cygwin

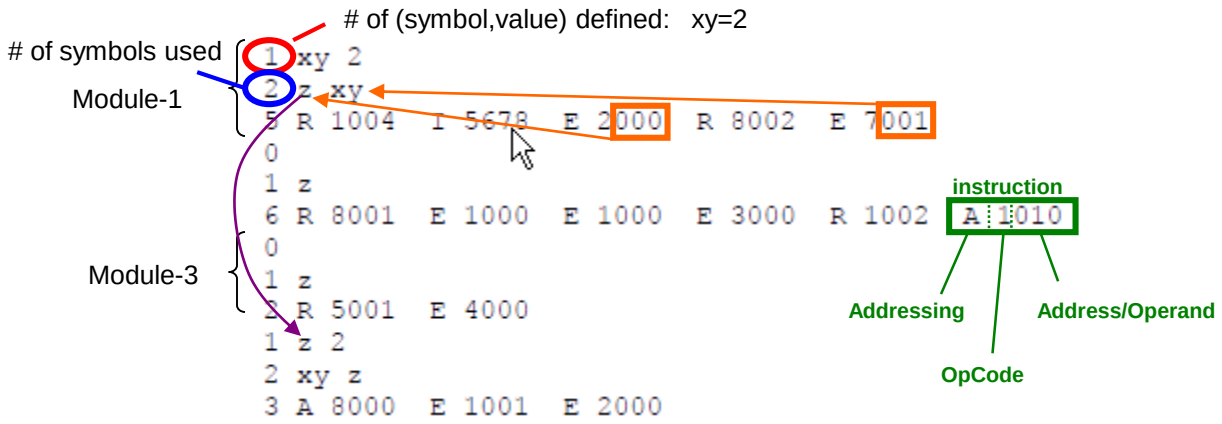
```
291:~/NYU/lab1/example>cat main.c
#include <stdio.h>

extern void print_hello();

int main(int argc, char **argv)
{
     print_hello();
}
```

LAB #1: Write a Linker

- Simplified module specification
  - List of symbols defined and their value by module
  - List of symbols used in module (including external)
  - List of "instructions"



Addressing

I: Immediate
R: Relative
A: Absolute
E: External

Lab #1: Write a Linker

input

```
1 xy 2
2 z xy
5 R 1004  I 5678  E 2000  R 8002  E 7001
0
1 z
6 R 8001  E 1000  E 1000  E 3000  R 1002  A 1010
0
1 z
2 R 5001  E 4000
1 z 2
2 xy z
3 A 8000  E 1001  E 2000
```

Fancy Output (not req)

```
Symbol Table
xy=2
z=15
Memory Map
+0
0:       R 1004          1004+0 =  1004
1:       I 5678                    5678
2: xy:   E 2000 ->z                2015
3:       R 8002          8002+0 =  8002
4:       E 7001 ->xy               7002
+5
0:       R 8001          8001+5 =  8006
1:       E 1000 ->z                1015
2:       E 1000 ->z                1015
3:       E 3000 ->z                3015
4:       R 1002          1002+5 =  1007
5:       A 1010                    1010
+11
0:       R 5001          5001+11=  5012
1:       E 4000 ->z                4015
+13
0:       A 8000                    8000
1:       E 1001 ->z                1015
2 z:     E 2000 ->xy               2002
```

Required output

```
Symbol Table
xy=2
z=15
Memory Map
000: 1004
001: 5678
002: 2015
003: 8002
004: 7002
005: 8006
006: 1015
007: 1015
008: 3015
009: 1007
010: 1010
011: 5012
012: 4015
013: 8000
014: 1015
015: 2002
```