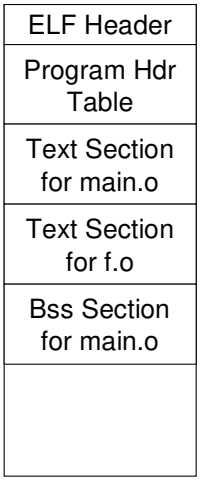
Linker

CS 217

Linker

- Combines multiple object files into a single executable file (`a.out`)
  - Also called a "link editor" or "linker/loader"
  - Object files (*.o) are ELF files
  - Executable files (a.out) are also ELF files



Unix `ld` Command

- Implicit (called by compiler)

```
gcc foo.c bar.c
```

- Explicit (supports separate compilation)

```
gcc -c foo.c
gcc -c bar.c
gcc foo.o bar.o
```

alternatively

- ld /usr/lib/crt0.o foo.o bar.o –lc –lm
  (do "**`gcc -v foo.o bar.o`**" to see full ugliness)

Two Main Functions

- Relocation:
  - assembler assumes each .o file begins at address 0, but when multiple object files are linked together some addresses must be relocated
- Resolution:
  - all external references must be resolved; this also involves searching libraries to find any undefined symbols

Example

main.c

```
static int a;
extern int b;
int c[10];
main() {
   f(a, b);
   g(a, b);
}
```

f.c

```
int b;
f(int a, int b) {
   return a + b;
}
g(int a, int b) {
   return a - b;
}
```

Example (cont)

Then run…

```
% gcc -c main.c
% nm -p main.o

00000050 b _a
         U _b
00000028 C _c
         U _f
         U _g
00000000 T _main
```

```
% gcc -c f.c
% nm -p f.o

00000004 C _b
00000000 T _f
0000000c T _g
```

```
B = bss symbol
D = data object symbol
T = text symbol
U = undefined
```

Relocation

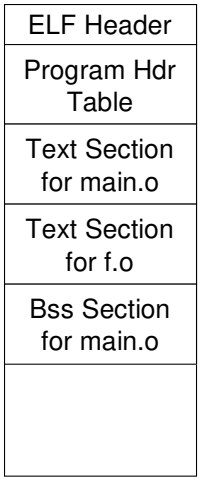
- Concatenate `.o` files to output `a.out`

```
% gcc main.o f.o
% nm -p a.out
00004000 d __DYNAMIC
000040c8 b _a
00004098 B _b
000040a0 B _c
00004098 D _edata
000040d0 B _end
00004090 D _environ
000024c4 T _etext
000022e0 T _f
000022ec T _g
00002290 T _main
```

```
B = bss symbol
D = data object symbol
T = text symbol
U = undefined
```

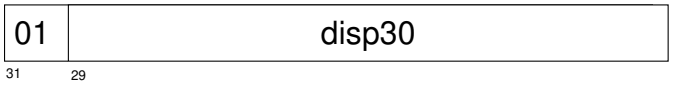
Relocation

- How are addresses in instructions handled?
  - Labels in call, branch, sethi, or, etc.



Relocation Entries

- Example 1: procedure call




```
ld   a,%o0
ld   b,%o1
call add
nop
st   %o0,c
```

What if “add” is in another file?

Relocation Entries

- Example 2: global variable

```
      .global
sum:  .word 0

      .local
cnt:  .word 0

      sethi %hi(cnt),%l0
      or dst,%lo(cnt),%l0
      ld [%l0], %o0
      inc %o0
      sethi %hi(sum),%l0
      or dst,%lo(sum),%l0
      st %o0, [%l0]
```

```
int sum = 0;

int cnt = 0;

sum = cnt + 1;
```

Relocation Entries

- Example 2: global variable

sum.o

```
        .global
sum:    .word 0
```

```
int sum = 0;
```

```
        .local
cnt:    .word 0

        sethi %hi(cnt),%l0
        or dst,%lo(cnt),%l0
        ld [%l0], %o0
        inc %o0
        sethi %hi(sum),%l0
        or dst,%lo(sum),%l0
        st %o0, [%l0]
```

```
int cnt = 0;

sum = cnt + 1;
```

What if "sum" is in another file?

Relocation Entries

- Assembler adds relocation entries to object file
- Each relocation entry:
  - Identifies instruction (call, sethi, etc.) that references relocatable address
  - Tells how to fix reference -- e.g.,

```
R_SPARC_WDISP30     call
R_SPARC_WDISP22     ba, be, bne,…
R_SPARC_HI22        sethi
R_SPARC_LO10        or
…
```

Example

main.c

```
static int a;
extern int b;
int c[10];
main() {
   f(a, b);
   g(a, b);
}
```

f.c

```
int b;
f(int a, int b) {
   return a + b;
}
g(int a, int b) {
   return a - b;
}
```

Example

```
elfdump -r main.o

Relocation: .rela.text
  type              offset   addend  section    with respect to
R_SPARC_HI22          0x4        0  .rela.text      a
R_SPARC_LO10          0x8        0  .rela.text      a
R_SPARC_HI22          0xc        0  .rela.text      b
R_SPARC_LO10         0x10        0  .rela.text      b
R_SPARC_WDISP30      0x1c        0  .rela.text      f
R_SPARC_HI22         0x24        0  .rela.text      a
R_SPARC_LO10         0x28        0  .rela.text      a
R_SPARC_HI22         0x2c        0  .rela.text      b
R_SPARC_LO10         0x30        0  .rela.text      b
R_SPARC_WDISP30      0x3c        0  .rela.text      g
```

Symbol Resolution

- Linking must resolve all symbols

```
% gcc main.o
ld: Undefined symbol
  _b
  _f
  _g
```

- Linking must define each symbol once

```
% cp f.o g.o
% gcc main.o f.o g.o
ld: g.o: _f multiply defined
g.o: _g multiply defined
```

Symbol Resolution (cont)

- Linkers can build an object file incrementally

```
% ld -o ab.o a.o b.o
% ld -o a.out ab.o c.o
```

- Linkers do not type check across modules

Symbol Resolution (cont)

- Linkers combine all modules from .o files, even if they are not referenced
- Linkers include modules from .a files (libraries) only if they are referenced
  - When find unresolved symbol, search modules
  - Only those modules that define one of the unresolved symbols are linked
  - Search for a particular symbol stops as soon as a match is found

```
gcc –o sort sort.o tree.o array.a
```

Symbol Table

- Executable files contain a symbol table
  - Useful for debugging

```
elfdump –s main.o

Symbol Table:  .symtab
  value       size     type bind oth ver shndx  name
0x00000000 0x00000000  NOTY LOCL  D    0 UNDEF
0x00000000 0x00000000  FILE LOCL  D    0 ABS    main.c
0x00000000 0x00000004  OBJT LOCL  D    0 .bss   a
0x00000000 0x00000000  NOTY LOCL  D    0 .text  gcc2_compil…
0x00000000 0x00000000  SECT LOCL  D    0 .bss
0x00000000 0x00000000  SECT LOCL  D    0 .text
0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF  b
0x00000004 0x00000028  OBJT GLOB  D    0 COMMON c
0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF  f
0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF  g
0x00000000 0x0000004c  FUNC GLOB  D    0 .text  main
```

Symbol Table

- Can remove by “stripping”

```
% gcc main.o f.o
% ls -l a.out
-rwxrwxr-x  llp   6376  a.out
% strip a.out
% ls -l a.out
-rwxrwxr-x  llp   3932  a.out
% nm a.out
nm: a.out: no symbols
```

Advanced Topics

- Dynamic Linking
  - complete linking step at execution time
  - supports dynamic libraries
- Shared Objects
  - text segment shared by multiple processes
  - requires less physical memory