Review of Instruction Formats, Program Build Process

Tom Kelliher, CS 240

Oct. 5, 2005

1 Administrivia

Announcements

First exam on Oct. 12, covering: 2.1–2.10, 2.13, 2.15.

Collect assignment; next assignment.

Assignment

Read 3.1–3.2.

From Last Time

Finished up function call mechanism.

Outline

1. Summary of operand addressing.
2. Program build process.

Coming Up

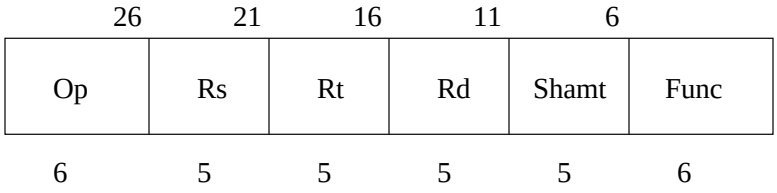
Number representation.

2 Summary of Operand Addressing Styles

Register, immediate, base and offset, PC-relative, "direct."

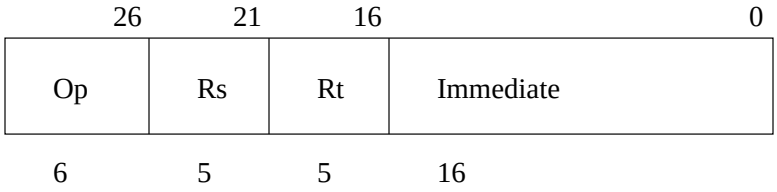
Base and offset is a form of indirect addressing.

1. Register mode. Format:



   Example: `add $t0, $t0, $t1`

2. Immediate mode. Format:



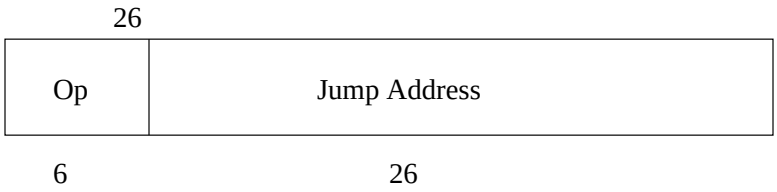
   Examples:

```
addi $t0, $t0, 1
lw $t1, 4($t0)
```

3. Suppose we need to branch further than ±32K words?

   (a) Example: procedure call in huge program.

(b) Solution: J format:



Example: `j reallyFarLabel`

(c) Conditional branch example:

```
                <code>
while:          beq $t0, $t1, reallyFarLabel
                <lots of code>
                b while
```

becomes:

```
                <code>
while:          bne $t0, $t1, near
                j reallyFarLabel
near:           <lots of code>
                j while
```

(d) Why can't this take us anywhere in memory?

Solved with `jr`!

3 Program Build Process

These are just comments. Make sure you read 2.10 on your own!

1. Assembler output: object file —

   (a) Data segment holds constants, generally strings and arrays. Scalar constants will be embedded in the code.

   Dynamic data is maintained in the heap segment.

(b) Relocation absolute addresses: J format and `jr` style instructions.

Program assembled assuming it will be load starting at address 0. This is never the case.

(c) Symbol table's external references — must be resolved by the linker.

2. Linker output: executable file —

(a) Think of source program split into multiple files — one function per file. These are compiled into object files.

Advantages?

(b) External references are *resolved* — calls between object files. Also, library code must be added to the code segment and absolute addresses for library function calls must be filled in within the original program.

Example: `printf("The answer is: %d", ans);`

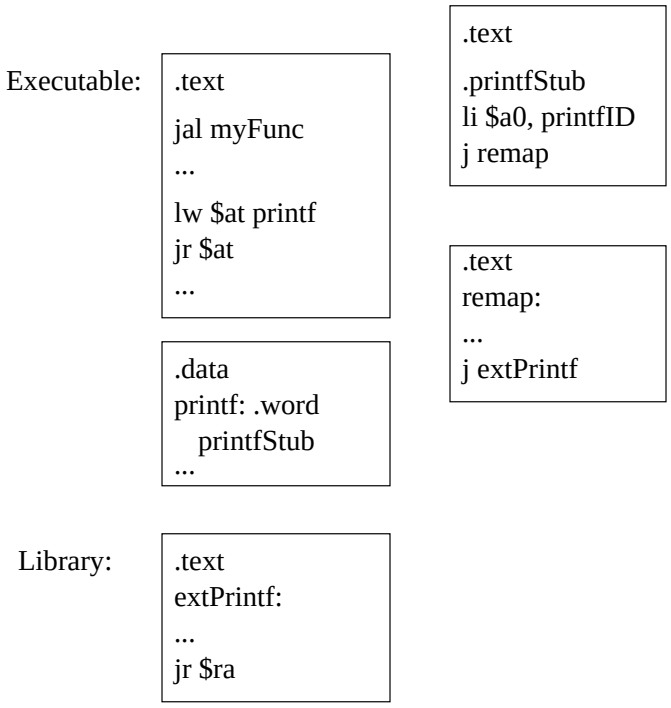
(c) Static linking. Impact upon disk and memory footprints.

3. Loader: load executable into memory and set execution environment.

4. Dynamic linked libraries.

(a) Initial version: smaller disk footprint. Linked at load time, so larger memory footprint, but library modules can be shared between programs.

(b) Lazy linkage: smallest footprints. Don't link libraries until they are actually needed. Requires indirection:



This is the mechanism Microsoft uses. DLL "Hell."