

Other Operand Types, Program Build Process

Tom Kelliher, CS 240

Feb. 18, 2002

1 Administrivia

Announcements

If you didn't see my e-mail:

1. HW II is due Monday, March 4.
2. The first exam will be Friday, March 8.

Assignment

Read 3.10. I will have you compile a C function to MIPS assembler in class. You will:

1. Decide what registers you will use.
2. Produce a frame map.
3. Compile the function into MIPS.
4. Write and add code to push/pop the frame, save/restore registers, etc.

From Last Time

Finished up function call mechanism.

Outline

1. Character strings.
2. Summary of operand addressing.
3. Program build process.

Coming Up

Function writing lab.

2 Character Strings

1. C strings are char arrays terminated by a NULL character.
2. In MIPS assembler, `.asciiz` is a NULL-terminated string.

`.ascii` is *not* terminated.

Example functions we'll look at:

(a) `int strlen(char *s)`

(b) `char * strcat(char *dest, char *src)`

`strcat` is dangerous — buffer overflows.

3. `strlen`:

(a) C code:

```
int strlen(char *s)
{
    char *t = s;

    while (*t != '\0')
        ++t;
```

```

    return t - s;
}

```

(b) Basic MIPS code:

```

# s in $a0
# t in $t0
# Current character of s in $t1
# strlen returned via $v0

                move $t0, $a0
while:          lbu $t1, 0($t0)
                beqz $t1, endWhile
                addi $t0, $t0, 1
                b while
endWhile:      sub $v0, $t0, $a0

```

4. strcat:

(a) C code:

```

char * strcat(char *dest, char *src)
{
    char *t = dest;

    while (*t != '\0')    /* Find end of dest. */
        ++t;

    while (*src != '\0') /* Start appending src to dest. */
    {
        *t = *src;
        ++t;
        ++src;
    }

    *t = '\0';           /* Append NULL to dest. */
    return dest;
}

```

(b) Basic MIPS code:

```

# dest in $a0

```

```

# src in $a1
# t in $t0
# current character in $t1
# dest returned via $v0

                move $t0, $a0
while1:         lbu $t1, 0($t0)
                beqz $t1, endWhile1
                addi $t0, $t0, 1
                b while
endWhile1:
while2:         lbu $t1, 0($a1)
                beqz $t1, endWhile2
                sb $t1, 0($t0)
                addi $t0, $t0, 1
                addi $a1, $a1, 1
                b while2
endWhile2:     sb $0, 0($t0)
                move $v0, $a0

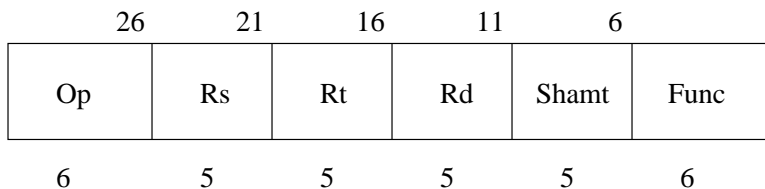
```

3 Summary of Operand Addressing Styles

Register, immediate, base and offset, PC-relative, “direct.”

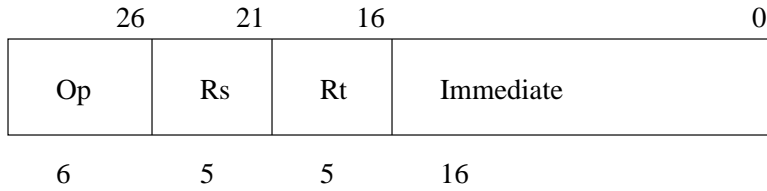
Base and offset is a form of indirect addressing.

1. Register mode. Format:



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

2. Immediate mode. Format:



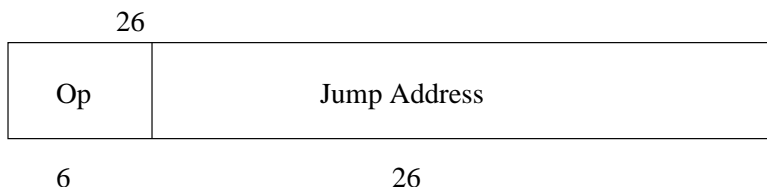
Examples:

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

3. Suppose we need to branch further than $\pm 32K$ 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!`

4 Program Build Process

These are just comments. Make sure you read 3.9 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) External references are *resolved* — 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);`

- (b) Static linking.

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

Dynamic linking.