February 22

- Continuing fixing the holes where the rain gets in ...

From the real world!

Another point on Endians

Another point on Representations

Vocabulary (prefixes, MIPS)

Programming details (registers, pseudo instructions, align, functions)

What does that assembly language do? (problem 3.2)

Big Picture Questions (syscall? OS? Who/What/Where?)

Performance in Real Life

From Microprocessor Report Volume 15, number 1, January 2001

Athlon Edges Out Pentium 4: *AMD Slightly Faster, But Intel Chip Shows Great Potential* by Peter N. Glaskowsky

Our first hands-on tests of Intel's 1.5GHz Pentium 4 and AMD's 1.2GHz Athlon have borne out the predictions we made four months ago based on Intel's preview of the P4's microarchitecture. The P4 processes substantially fewer instructions per clock (IPC) than its predecessor on highly conditional code, such as that found in typical PC productivity software, and is rarely more efficient than the Pentium III, even on multimedia algorithms. The P4's higher clock speed gives it impressive peak throughput on selected tests, but overall, on a variety of tests, it comes in slightly behind its primary competitor.

Performance in Real Life

From Microprocessor Report Volume 15, number 1, January 2001

Benchmark Suite	Athlon 1.2GHz	Pentium 4 1.5GHz	Athlon/P4 ratio (% delta)
ZDM Business Winstone 2001	44.8	42.4	5.7
ZDM Content Creation Winstone 2001	60.8	54.3	12.0
BAPCo/Mad Onion SYSmark 2000 composite	221	209	5.7
Internet Content Creation	238	226	5.3
Office Productivity	209	197	6.1

Endians?

Consider the following code

```
foo:  .ascii “Gary”      # foo takes 4 bytes, 32 bits
      la    $t1, foo     # t1 = &foo
```

What is the value of the WORD at foo? In other words what is the value of register t2 after:

```
      lw    $t2, 0($t1) # what is in t2?
```

Little Endian ✍ t2 == 0x79726147

Big Endian ✍ t2 == 0x47617279

On BOTH machines:

```
      lb    $t3, 0($t1) # t3 == ‘G’
```

Hex

- Numbers in hex are commonly preceded by 0x
  - 0x1 == 1, 0x10 == 16, etc.
- Hex is cool because each digit corresponds to 4 bits

0x0==0000b, 0x1==0001b, 0x2==0010b, 0x3==0011b
0x4==0100b, 0x5==0101b, 0x6==0110b, 0x7==0111b
0x8==1000b, 0x9==1001b, 0xA==1010b, 0xB==1011b
0xC==1100b, 0xD==1101b, 0xE==1110b, 0xF==1111b

So hex to binary is EASY!

0x2A == 00101010b

0x8001 == 1000000000000001b

binary to hex is easy too!

1000011110100000b == 0x87A0

Vocabulary

MIPS is either:

Millions (decimal) of Instructions Per Second

or

the computer architecture we're studying.

Unfortunate but true. It should always be obvious by the context.

On future tests I will strive to make the distinction by using the word "architecture" in conjunction with MIPS when naming the machine.

Programming Details

READ THE BOOK!

Choosing Registers

- Arguments in $a0-4
- Results in $v0-1
- In a “leaf” function
  - use $t0-7 for everything and you won’t have to save and restore anything
  - if you need more then save $s0-7, use them, and then restore at end
- In a “non-leaf” function
  - use $t0-7 for temps that don’t need to be saved across calls
  - use $s0-7 for variables that you need across calls
- Always save $s0-7, $ra, $sp if you modify them.
- Use memory pointed to by $sp to save and restore registers, allocate arrays, etc.

pseudo instructions

- They aren't *REAL* instructions
- You can think of them as
  - shorthand
  - macros
  - inline functions
  - syntactic sugar
- They are supposed to make your life easier and your programs easier to read
  - move $t1,$t0 ✍ add $t1, $t0, $zero
  - la $t0,foo ✍ lui $t0, UPPER16(foo)
ori $t0, LOWER16(foo)
  - li $t0, 23 ✍ addi $t0, $zero, 23
  - li $t0, 0x2300AB ✍ lui $t0, 0x23
ori $t0, 0x00AB

What does align do?

- align fills in enough 0's to get to an address with the indicated number of trailing zeros

The point is you shouldn't have to know the numerical value of an address to get the result you want...

Suppose I want a table that starts at an address that is a multiple of 1024

```
    .align 10 # next address will be a multiple of 2^10
foo:
    .word 100
```

What does the code in 3.2 do?

```
int maxdups = 0;
int thedup = 0;
for(i=0; i<size; i++) {
   int t = array[i];
   int dups = 0;
   for(j=0; j<size; j++)
       if(t == array[j])
             dups++;
   if(dups > maxdups) {
       maxdups = dups;
       thedup = t;
   }
}
```

Big Picture Issues

- What does syscall do? (see A49)
- How does the OS load my program, make multiple programs run, protect one program from another, etc.