SURF!

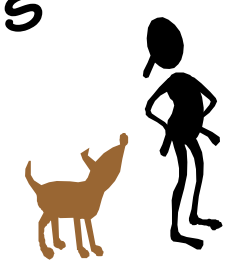
Summer Undergraduate Research Fellowship

- $3000 for the summer
- Looks great on your resume
- Interest meeting today 5:30-7:30 Greenlaw 431
  - Previous SURF recipients will be on hand to talk about their experience and to answer questions
- Applications due: March 1
- http://www.unc.edu/depts/our/summer_research.html

- Talk to me or other faculty for ideas

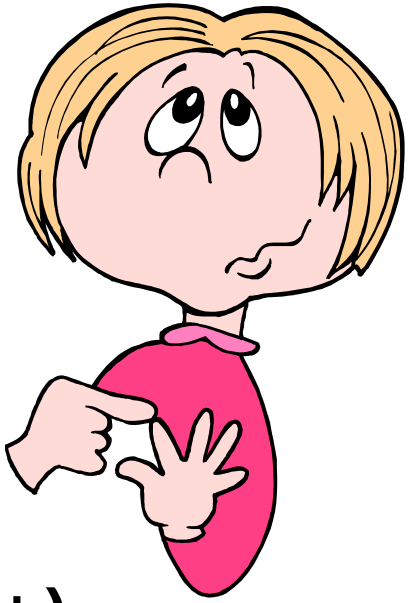
Adventures in Assembly Land

- What is an Assembler
- ASM Directives
- ASM Syntax
- Intro to SPIM
- Simple examples



A Simple Programming Task

- Add the numbers 0 to 4 ...
- 10 = 0 + 1 + 2 + 3 + 4
- In "C":



```
int i, sum;

main() {
    sum = 0;
    for (i=0; i<5; i++)
        sum = sum + i;
}
```

- Now let's code it in ASSEMBLY

What IS an Assembler?

- A program for writing programs
- Machine Language:
  - 1's and 0's toggled loaded into memory. (Did anybody ever really do that?)
- Assembly Language:

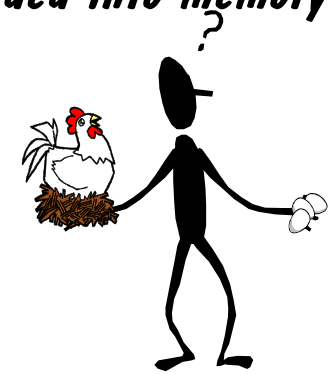
Front panel of a classic PDP8e. The toggle switches were used to enter machine language.

```
    .globl main
main:
    subu $sp, $sp, 24
    sw   $ra, 16($sp)
    li   $a0, 18
    li   $a1, 12
    li   $a2, 6
    jal  tak
    move $a0, $v0
```

→ **UASM PGM** →

```
01101101
11000110
00101111
10110001
.....
```

STREAM of bits to be loaded into memory



Symbolic SOURCE text file

ASSEMBLER Translator program

Binary Machine Language

Assembler:

1. A Symbolic LANGUAGE for representing strings of bits
2. A PROGRAM for translating Assembly Source to binary.

Assembly Source Language

An Assembly SOURCE FILE contains, in symbolic text, values of successive bytes to be loaded into memory... e.g.

```
.data 0x10000000
.byte 1, 2, 3, 4
.byte 5, 6, 7, 8
.word 1, 2, 3, 4
.asciiz "Comp 411"
.align 2
.word 0xfeedbeef
```

- Specifies address where following values are loaded
- First four byte values
- Second four byte values
- Four WORD values (each is 4 bytes)
- A string of 9 ASCII bytes (8 + NULL)
- Align to next multiple of 4 (2^2)
- Hex encoded WORD Value

Resulting memory dump:

```
[0x10000000]    0x04030201  0x08070605  0x00000001  0x00000002
[0x10000010]    0x00000003  0x00000004  0x706d6f43  0x31313420
[0x10000020]    0x00000000  0xfeedbeef  0x00000000  0x00000000
```

Notice the byte ordering. This MIPS is "little-endian" (The least significant byte of a word or half-word has the lowest address)

Assembler Syntax

- Assembler DIRECTIVES (Keywords prefixed with a '.')
  - Control the placement and interpretation of bytes in memory

`.data <addr>`	Subsequent items are considered data
`.text <addr>`	Subsequent items are considered instructions
`.align N`	Skip to next address multiple of 2^N

  - Allocate Storage

`.byte` $b_1, b_2, \ldots, b_n$	Store a sequence of bytes (8-bits)
`.half` $h_1, h_2, \ldots, h_n$	Store a sequence of half-words (16-bits)
`.word` $w_1, w_2, \ldots, w_n$	Store a sequence of words (32-bits)
`.ascii "string"`	Stores a sequence of ASCII encoded bytes
`.asciiz "string"`	Stores a zero-terminated string
`.space n`	Allocates n successive bytes

  - Define scope

`.globl sym`	Declares symbol to be visible to other files
`.extern sym size`	Sets size of symbol defined in another file

More Assembler Syntax

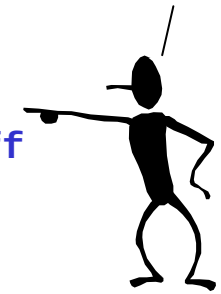
- Assembler COMMENTS

  All text following a '#' (sharp) to the end of the line is ignored

- Assembler LABELS
  - Labels are symbols that represent memory addresses. Labels take on the values of the address where they are declared. Labels declarations appear at the beginning of a line, and are terminated by a colon. Labels can be established for data items as well as instructions… e. g.

```
        .data
item:   .word 1                 # a data word
        .text
start:  .word 0x00821820        # add   $3, $4, $2
        .word 0x00031a00        # sll   $3, $3, 8
        .word 0x306300ff        # andi  $3, $3, 0xff
```

While having an assembler helps, coding like this is still painful. (Don't actually do this!)



Our Example: Variable Allocation

- Two integer variables (by default 32 bits in MIPS)

```
.data
.globl sum
.globl i
sum:    .space 4
i:      .space 4
```

- ".data" assembler directive places the following words into the data segment
- ".globl" directives make the "sum" and "I" variables visible to all other assembly modules
- ".space" directives allocate 4 bytes for each variable

Even More Assembler Syntax

- Assembler PREDEFINED SYMBOLS
  - Register names and aliases

    `$0-$31, $zero, $v0-$v1, $a0-$a3, $t0-$t9, $s0-$s7,`
    `$at, $k0-$k1, $gp, $sp, $fp, $ra`

- Assembler MNEMONICS
  - Symbolic representations of individual instructions

    `add, addu, addi, addiu, sub, subu,`
    `and, andi, or, ori, xor, xori, nor, lui,`
    `sll, sllv, sra, srav, srl, srlv,`
    `div, divu, mult, multu, mfhi, mflo, mthi, mtlo,`
    `slt, sltu, slti, sltiu, beq, bgez, bgezal, bgtz, blez,`
    `bltzal, bltz, bne, j, jal, jalr, jr,`
    `lb, lbu, lh, lhu, lw, lwl, lwr, sb, sh, sw, swl, swr, rfe`

  - pseudo-instructions (some mnemonics are not real instructions)

    `abs, mul, mulo, mulou, neg, negu, not, rem, remu, rol, ror,`
    `li, seq, sge, sgeu, sgt, sgtu, sle, sleu, sne, b, beqz, bge,`
    `bgeu, bgt, bgtu, ble, bleu, blt, bltu, bnez, la, ld, ulh,`
    `ulhu, ulw, sd, ush, usw, move,syscall, break, nop`

Actual "Code"

- Next we write ASSEMBLY code using the instruction mnemonics

A common convention, which originated with the 'C' programming language, is for the entry point (starting location) of a program to named "main".

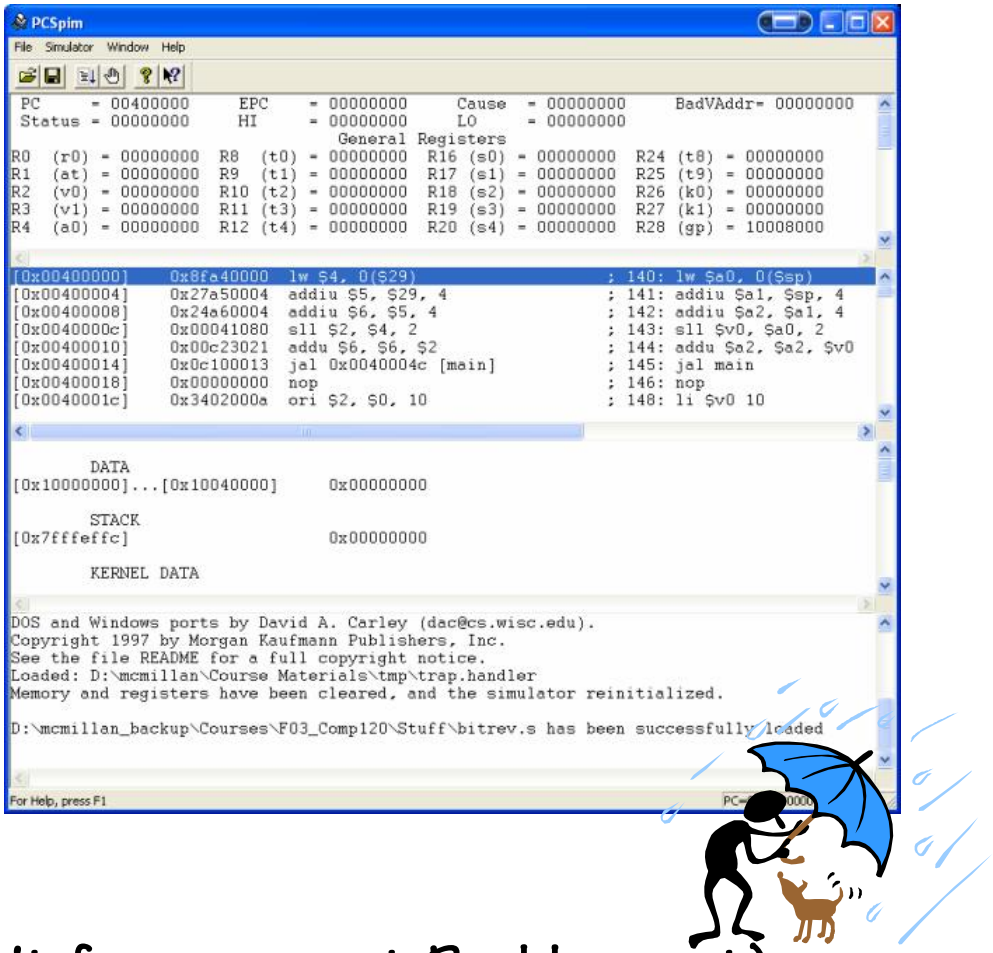
```
.globl main
.text
main:
     add   $8,$0,$0      # sum = 0
     add   $9,$0,$0      # for (i = 0; ...
loop:
     addu  $8,$8,$9      # sum = sum + i;
     addi  $9,$9,1       # for (...; ...; i++
     slti  $10,$9,5      # for (...; i<5;
     bne   $10,$0,loop
end: beq   $0,$0,end
```

Bookkeeping:

1) Register $8 is allocated as the "sum" variable
2) Register $9 is allocated as the "i" variable

SPIM Simulation

- Let's tryout our Example
- We'll use the SPIM (MIPS backwards) integrated ASSEMBLER, SIMULATOR, and DEBUGGER.
- Links to SPIM program and docs are available at "Links" section of the course website
(You'll need to download it for your next Problem set)



Getting Started SPIMing

- The following hints will get you started with SPIM
  - By default a small fragment of code is loaded called the "kernel". We will discuss the role of this code in detail, later in the course. For now, you can assume it's job is to branch to the "main" label of your code. It does so in approximately 4 instructions.
  - You can "single-step" the machine by pressing [F10]
  - The results of the execution of each instruction are reflected in the register and memory location values
  - Illegal operations result in an "exception" which causes your code to automatically jump back the kernel. For our purposes now, this will be due to a bug in your program.
  - Refer to the manual for more fancy usage, such as setting and executing to breakpoints

A quick demo

A Slightly More Challenging Program

- Add 5 numbers from a list ...
- sum = $n_0 + n_1 + n_2 + n_3 + n_4$
- In "C":

```
int i, sum;
int a[5] = {7,8,9,10,8};

main() {
    sum = 0;
    for (i=0; i<5; i++)
        sum = sum + a[i];
}
```

- Once more... let's encode it in assembly

Variable Allocation

- We cheated in our last example. Generally, variables will be allocated to memory locations, rather than registers (Though clever optimization can often avoid it).
- This adds an array with 5 integers

```
.data
.globl sum
.globl i
.globl a
sum:    .space 4
i:      .space 4
a:      .word 7,8,9,10,8
```

- ".word" allows us to initialize a list of sequential words in memory. The label represents the address of the first word in the list, or the name of the array

The New Code

- Note the small changes:

```
.globl main
.text
main:
     sw    $0,sum        # sum = 0;
     sw    $0,i          # for (i = 0;
     lw    $9,i          # allocate register for i
     lw    $8,sum        # allocate register for sum
loop:
     sll   $10,$9,2      # covert "i" to word offset
     lw    $10,a($10)    # load a[i]
     addu  $8,$8,$10     # sum = sum + a[i];
     sw    $8,sum        # update variable in memory
     addi  $9,$9,1       # for (...; ...; i++
     sw    $9,i          # update memory
     slti  $10,$9,5      # for (...; i<5;
     bne   $10,$0,loop
end: beq   $0,$0,end
```

A Coding Challenge

- What is the largest Fibonacci number less than 100?
  - Fibonacci numbers:

    $F_{i+1} = F_i + F_{i-1}$

    $F_0 = 0$

    $F_1 = 1$
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

- In "C"

```
int x, y;

main() {
    x = 0;
    y = 1;
    while (y < 100) {
        int t = x;
        x = y;
        y = t + y;
    }
}
```

MIPS Assembly Code

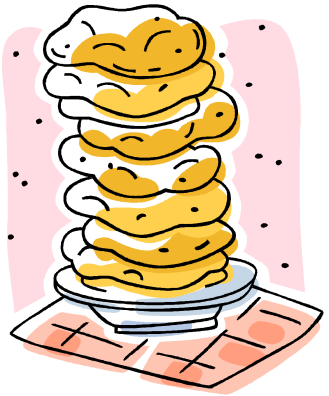
- In assembly

```
.data
.globl x
.globl y
x:    .space 4
y:    .space 4
```

```
.globl main
.text
main:
     sw    $0,x          # x = 0;
     addi  $9,$0,1       # y = 1;
     sw    $9,y
     lw    $8,x
while:                   # while (y < 100) {
     slti  $10,$9,100
     beq   $10,$0,endw
     add   $10,$0,$8     #     int t = x;
     add   $8,$0,$9      #     x = y;
     sw    $8,x
     add   $9,$10,$9     #     y = t + y;
     sw    $9,y
     beq   $0,$0,while   # }
endw:
     beq   $0,$0, endw   # answer is x
```

Next Time

- Parameterized Programs
- Procedures
- Stacks



- MIPS procedure linkage conventions