

EECS 370

Discussion 2

Week of 1/23 – 1/29

- I) Project 1
 - a. Get started on the project NOW!
 - b. Two submissions with feedback per day
- II) LC-2K9
 - a. .fill example
 - i. lw 0 1 step
 - ii. add 1 2 2
 - iii. ...
 - iv. step .fill 3
 - b. beq
 - i. If label, branch target is absolute
 - ii. If offset, branch target is relative
- III) Function calls
 - a. Call stack (MIPS)
 - i. Incoming parameters
 - ii. \$FP
 - iii. Return address
 - iv. Callee saved registers
 - v. Local variables
 - vi. Spilled registers
 - vii. Caller saved registers
 - viii. Outgoing parameters
 - ix. \$SP
 - b. Section between \$FP and \$SP is a stack frame
- IV) Caller/callee saved
 - a. “You” refers to the current stack frame, or current function scope
 - b. Caller saved registers
 - i. IF you want to save the value inside over a function call, you must save it yourself
 - ii. No need to save if the value will not be used afterwards
 - iii. Also called “temporary” registers, because their values are not automatically preserved
 - iv. When you first start using them, no need to save the value that used to be there (because YOUR caller is responsible!)
 - c. Callee saved registers
 - i. When you first start using them, you must save the value that used to be there, because YOU are responsible

- ii. No need to save values over a function call (they will always be preserved, because any subroutines you call will be responsible for saving them)
 - iii. Also called “permanent” registers, because their values are always preserved
 - d. The important stuff: when do you save values?
 - i. Caller saved register
 - 1. If two things are true: you are about to enter a subroutine, AND you need the value in the register later
 - ii. Callee saved register
 - 1. If you need to use the register AT ALL, you must save its value before doing anything with it
 - e. Advantages/disadvantages
 - i. Caller saved
 - 1. No need to save if isn't “live” across a function call
 - 2. Leaf routines: 0 overhead!
 - ii. Callee saved
 - 1. Only one save/load needed within the current function, even if there are loops!
 - 2. No need to save at all if you don't use it!

V) Class problems

- a. Data layout
 - i. How much memory is required for the following data, assuming the data starts at address 500?
 - 1. char a;
 - 2. struct {
 - int b;
 - char c;
 - 3. } d;
 - 4. int *e;
 - 5. char f;
 - 6. short g[10];
- b. MIPS memory space
 - i. Where do all these variables go?
 - 1. char c;
 - 2. void bar(char *b)
 - 3. {
 - static int x;
 - b = malloc(20);
 - printf(“Hello World”);
 - ii. The printf line is especially tricky! A static string is created and a pointer to it is passed to printf, not the actual string itself!