Midterm Topics

The midterm (which is closed everything) will cover course material up through procedure call. (The performance topic, originally intended to be on the midterm, will **not** be on the midterm.)

Questions will assume you've attended lectures and sections, done the reading in the text, done the reading of other handouts, looked at sample code available from the course calendar, done the assignments, looked at sample assignment solutions.

Here's a run down of topics.

ISA Topics

- What is an ISA?
  - Distinction from an implementation
- Fetch-Increment-Execute
- Load-store architecture
- Registers (what and why?) and PC
- R-type / I-type / J-type instructions
  - operations available
    - signed vs. unsigned arithmetic and loads
  - encoding
    - How are they encoded?
    - What limits does this place on what they can do?
    - How are those limits addressed by the ISA?
- Memory
  - unit of addressing (byte) vs. transfer size
  - base-displacement addressing (why?)
  - limits on size of physical memory
  - alignment
- (Conditional) branches and jumps
  - PC-relative branches (why?)
- ISA support for procedures (jal)

Software Topics

- Process image:
  - text / static data / heap / stack
- Role of compiler (using C as the example programming language)
  - Type checking, scope, and other compile-time only ideas
  - Translate instructions to “equivalent” assembler
  - Create variables: automatic/local vs. static/global variables (lifetime vs. scope)
  - Role of C's .h files / role of #include
- Role of the assembler
  - Encode simple instructions into binary
  - pseudo-ops

- provide symbol definition and use information to linker
- data types?

- Role of the linker
  - Compose process image
    - "concatenate" .o files
    - Resolve external symbol values
    - Patch instructions (Which? Why?)
- Role of the loader
  - Find free memory and read .exe file contents into it
  - Set up $gp, $sp, $fp
  - Set PC to entry point
- Starting from a C program, give assembler that achieves same result
  - Expressions, loops, if...then...else, etc. (break, boolean expression ? ... : ...)
  - arrays and pointers
- Libraries: static and dynamic

Procedure Call Conventions

- When details are asked for, they will be about the Cebollita procedure call convention
- The stack and activation records (aka stack frames)
- What does any procedure call convention have to do?
  - Caller saved registers
  - Passing arguments to callee
  - Callee saved registers
  - Allocation / deallocation of local variables
  - How any return value is passed back
  - How the callee returns to the instruction following the call (i.e., the one following the jal in the caller)
- $ra / $fp / $sp
  - What they are, how they're used, why we need them
- How are global variables addressed? How are local variables addressed?
- Alternatives to Cebollita
  - general differences in the MIPS calling convention
    - what are they? What is the motivation for these differences?
  - Other possible procedure call mechanisms I might invent
    - How do they differ from Cebollita? When would they be better/worse than what Cebollita does?

Miscellaneous

- Arithmetic implications of shifting
  - logical vs. arithmetic shifts
  - "sign extension"
- Binary representation of characters and strings
- Hex