Review sheet for midterm exam

The information on this sheet is intended to help you identify the key points that you should be comfortable knowing. It is not a substitute for the last month of lectures and homework assignments, and so it is possible that there will be questions on the exam that are not answered by the material on this review sheet.

The mid-term will be given in class, Monday, May 1. No books, no notes, no electronic devices. You will be provided with a copy of the green card from the book.

Computer Instructions

Reduced Instruction Set Computer. Reduced compared to Complex – simple memory access model, regular instruction format, single instruction width, etc.

Registers. Register names and usage convention for each register. Which registers should not be used at all (at, k0, k1), which registers can be changed without saving and restoring them (a0-a3, t0-t9, v0-v1), which register is a constant ($zero), and which registers must be saved and restored if a procedure wants to use them (s0-s7, gp, sp, fp, ra). Instructions: move.

Load and Store Instructions. Memory address, what it means that memory is byte-addressable, byte, half-word, word, double word, and alignment in memory. Load and store word, load and store byte, difference between signed and unsigned loads, difference between aligned and unaligned loads/stores. Basic addressing mode: offset + base register value. Big-endian, little-endian storage convention. Immediate mode values (constants embedded in the instructions) and the limitations on their magnitude (limited by the size of the 16-bit field). Instructions: la, li, lb, lbu, lw, sb, sw.

Branch instructions. Be able to read and interpret the various branch instructions. Limitations on the distance that a branch can go. PC-relative branch addressing. Understand what the slt comparison instructions do and why they are useful for the branch instructions. Know that the branch pseudo-ops expand to include the comparison for you if needed and so most of the time you don’t need to actually write an explicit comparison. Instructions: b, beq, bne, bgt, bge, blt, ble, beqz, bnez, bgtz, bgez, bltz, blez.

Jump instructions. Understand that the jump and link instruction is the key to calling procedures, that the jump register instruction is the means for getting back, and how they interact through the $ra register. Plain jumps can go further than any branch due to the larger offset that they contain. Instructions: j, jal, jalr, jr.

Arithmetic and Logical instructions. Recognize that a trailing "i" indicates an "immediate value (a constant)", a trailing "u" indicates "unsigned (no overflow exception generated)". Be able to read a table of existing register values and an instruction to operate on some of those registers, and write down the result of the instruction.

Understand what happens when you do a "shift logical" to the bits in a register (bits that are shifted out are dropped, bits that are shifted in are zero, the rest of the bits are in a new position). Understand AND and OR operations, and what it means to use a mask with an AND instruction. Instructions: add, sub, div, mul, and, or, sll, sllv, srl, srlv.

Writing assembly code. The .data directive tells the assembler to put data in the heap, .text tells it to put instructions in the program code section. Directives .word, .byte, .asciiz store data in memory. A label in the code or data sections is a symbolic reference to a location in memory. Know how to write comments that add meaning for the next programmer and are not just a translation of the instructions into words. Assembly code is the set of instructions that you write, machine code is the binary values that those instructions get translated into.

Procedures

Understand how program memory is laid out (program, heap, stack) and how the heap and the stack grow towards each other. Identify the steps in calling another procedure (set up parameters, transfer control, acquire needed storage, do the task, make result available, release storage, return) and understand how each step is accomplished. Registers for parameters, stack space for parameters, adjusting stack pointer to control stack usage, stack space for saving and restoring register values that must be preserved, $v0 to return a value. Given a small segment at the start of a procedure, draw the contents of the stack frame.

Leaf procedure, non-leaf procedure, calling tree diagram. Given a short sequence of code, draw the calling tree and label the arguments of the called procedures.

Numbers and Formats

Character data. It is common to store characters using 8-bit values that fit in a byte. Strings of such characters can be terminated by a null-byte (a zero value) or they can be counted with a separate variable holding the number of characters in the string. Other encoding formats such as Unicode use more bits per character and thus can encode larger numbers of characters. Understand generally how Unicode encoding accomplishes the larger range. Counted and null-terminated strings.

Binary, hex, and decimal number bases. Total number of values in a field that is n bits wide is 2^n. Maximum value is 2^n-1. Starting from a binary, hex, or decimal value less than 16_{10}, convert it to the equivalent value in binary (0 to 1111_2), to hex (0 to F_{16}) or to decimal (0 to 15_{10}). Convert any length binary number to the equivalent hex value, and hex to binary. Know that the following value is the largest that will fit in eight bits:

$1111\ 1111_2 = FF_{16} = 255_{10}$.

Understand that shifting a number one bit position to the right divides by 2, shifting one position to the left multiplies by 2. See the Number Base Charts in lecture 3 for information about why the values are related the way they are. Hex numbers are often written with a leading "0x" to indicate that they are base 16.

Signed Integer Numbers. Understand 2's complement notation. The sign bit is the high order bit (most significant bit, bit 31 in a 32-bit quantity), it is 0 for positive numbers, 1 for negative numbers. Convert from negative to positive representation or vice versa using "complement and add 1."

Floating Point. Understand the various fields of a floating point number and how they are used.

Pipelining

The five stages of the basic MIPS pipeline are IF, ID, EX, MEM, WB. Instruction Fetch, Instruction Decode, Execute, Memory, Write Back. The instruction set was designed to be pipelined: The instructions are all 4 bytes, which simplifies IF. Instructions have common format (op code location, register locations), that simplifies ID. Only load and store instructions access memory, which simplifies MEM. Most memory operations are aligned, that simplifies MEM. Branch instructions are a problem for pipelines because they change the order of instruction fetches, which may have already started by the time the branch decision has been made. A strategy for coping with this is the "branch delay slot", in which an instruction is always executed after the branch but before the instruction at the new destination address. Another strategy is "static branch prediction" in which the programmer or the compiler makes a guess about which will happen more frequently. "Dynamic branch prediction" keeps records with a simple state table and builds a prediction based on program behavior. The cost of a mistake is the need to flush the instructions that are already in the pipeline, which slows it down.