Course Review of CSE 230: Computer Organization, Fall 2004

Review Materials: Chapter 1; Chapter 2; Chapter 3; Chapter 5; **plus** *Lecture Notes and Homework Assignments.*

Central Points of Chapter 1:

1. Successful software development requires knowledge of computer organization.
2. Instructions and data can be naturally represented in binary numbers
3. Computers directly understand execute-only machine language instructions.
4. Five components of a computer: (1) control, (2) datapath, (3) memory, (4) input, (5) output. (1+2=CPU; 4+5=I/O)
5. Software and hardware hierarchy; Principle of abstraction – omits details that are not relevant to the task at hand; hierarchy of abstractions.
6. Decomposibility of computer system:
   - Application software – user programs
   - System software – (1) Compilers and Operating Systems make portable programs possible; (2) Greatly simplifies comparisons of different architectures.
   - Hardware – von Neumann Computer: stored programs; Maurice Wilkes: micro-programming
7. Instruction set architecture: a crucial abstraction of computer system.
8. Implementation of an architecture.

Central Points of Chapter 2:

1. Stored program computer:
    - represent instructions in binary
    - store instrcutions and data in memory
2. Machine language and instruction set architecture:
    - Formats: R, I, J
    - Fields: op, rs, st, rd, shamt, funct, immediate, and jump address.
3. Assembly language:
    - Symbolic machine language, almost one-to-one with machine language
    - Fixed format: one instruction per line– operation and operands, fixed number of operands
    - Registers
    - words, bytes
4. Primary memory: linear array with addresses, word and byte address.
5. Addressing modes: register, base, immediate, PC-relative.
6. MIPS instructions:
    - data movement: lw, sw – the only way to access memory
    - ALU: and, or, add, slt, sub, addi, lui, slti,
    - control: beq, bne, j, jr, jal
7. Instruction set design principles:
    - Simplicity favors regularity
    - Smaller is faster
    - Good design requires compromise
    - Make the common case fast
8. Software system: compile, assemble, link, load

Central Points of Chapter 3:

1. Binary representation of data: $B = b_{n-1}b_{n-2}...b_1b_0$
   Value(B)=$b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + ... + b_1 \times 2^1 + b_0 \times 2^0$

2. Representing signed integers:

   - Sign-and-Magnitude: "0" twice; $-(2^{n-1} - 1) \leq B \leq (2^{n-1} - 1)$
   - One's Complement: "0" twice; $-(2^{n-1} - 1) \leq B \leq (2^{n-1} - 1)$
   - Two's Complement: "0" once; $-2^{n-1} \leq B \leq (2^{n-1} - 1)$

3. Overflow: Conditions for overflow (for two's complement)

Operation	A	B	Result	Overflow?
A+B	+	+	−	Yes
			+	No
A+B	−	−	−	No
			+	Yes
A-B	+	+	−	No
			+	No
A-B	−	−	−	No
			+	No

   Conditions for non-overflow: (+)+(-); (-)+(+); (+)-(+); (-)-(-).

4. Subtraction in two's complement: negate (i.e., apply two's complement) the subtrahend and then add.

5. If no overflow occurs, the addition of two's complement integers can be done by adding the integers as positive binary numbers and ignore the carry from the highest order bit (MSB).

6. Full and half adder.

7. Ripple carry ALU: and, or, add, sub, slt

8. Carry-lookahead adder: $G_i = a_i b_i$ and $P_i = a_i + b_i$

9. Multiplication:

   - positive integers: shift and add (if multiplier bit is 1); three versions in increasing order of performance.

- negative integers: convert and multiply

10. Division:
    - positive integers: subtract and 1(or 0) quotient and shift (and restore). three versions in increasing order of performance.
    - negative integers: convert and divide

Central Points of Chapter 5:

1. Clocks and combinational and sequential circuits
    (a) Logic functions $\Longleftrightarrow$ Truth tables $\Longleftrightarrow$ Logic equations $\Longleftrightarrow$ Circuits
        - Boolean algebraic laws
        - Basic gates: AND, OR, NOR, NAND, XOR

    (b) Combinational (Unclocked) circuits:
        - MUX (selector)
        - DEMUX (decoder)
        - PLA (two-level: sum-of-products and product-of-sums)
        - ROM (direct implementation of truth-table multi-functions)

    (c) Sequential (Clocked) circuits
        - clock signal: cycle time = 1/ clock rate (frequency)
        - timing diagram
        - latches: S-R and clocked latches
        - Flip-flop: S-R, D
        - Registers, register file,
        - SRAM and DRAM

    (d) Finite state machines
        - outputs = function of inputs and current state
        - next state = function of inputs and current state
        - state transition diagram
        - state transition table
        - derive output expressions in terms of inputs and current state
        - derive next-state expressions in terms of inputs and current state

2. PC: increment with sequential instruction or update with branch destination

3. Register file

4. Single cycle implementation with combinational control

    - identify datapath for each instruction type three combine datapaths to maximize the sharing of datapath among all instruction types
    - trace the datapath to locate necessary control points/signals
    - based on instruction execution steps, timings, cause-effect relationships, and datapaths design circuits to generate control signals.

5. Multiple-cycle implementation of the processor

    - the same datapath and set of control signals as in the case of single-cycle implementation
    - identify five steps of instruction execution and the associated subset of control signals for each instruction at each step
    - finite state machine describing timings and sequences of steps of instruction executions
    - the use of register bits to store/represent the state information
    - microgramming implementation of the control unit