CS 61C Fall 2011
Kenny Do
Final cheat sheet

- Increment memory addresses by multiples of 4, since lw and sw are byte-aligned
- When going from C to Mips, always use addu, addiu, and subu
- When saving stuff into the stack, addi to $sp
  - Stack frame includes return instruction address, parameters, space for local variables
- Calling conventions
  - Save and restore $s0-9 and $sp
  - Save $ra if callee does nested function call
  - Save $a0-3 and $t0-9 in caller if necessary
- Average Memory Access Time = L1 Hit Time + L1 Miss Rate * L1 Miss Penalty
  - L1 Miss Penalty = L2 Hit Time + L2 Miss Rate * L2 Miss Penalty
- Increasing associativity by 2 decreases size of Index by 1 bit and increases the size of Tag by 1 bit
- Big Endian vs Little Endian determines BYTE order, not the bit order within bytes
  - Big endian stores the most significant byte first
- 2 GHz = 500 picosec frequency
- Vars declared outside of main() are in static
- *bigArray[4] uses 4*4 bytes, but bigTriple[3] uses 3 * sizeof(bigTriple)
- Seek time is time it takes to move disk head from one track to another
- Updating other caches on write and invalidating on cache write maintain cache coherence during writes
- $2^{\#\text{ offset bits}} =$ block size
- Put starting arrow in FSM diagrams

1 Data Level Parallelism

- Flynn Taxonomy
  - {Single, Multiple} Instruction {Single, Multiple} Data Stream
- 8 XMM registers are 128 bits wide

2 Thread Level Parallelism

- Example SSE instrinsics on _m128d data type
  - _mm_{load, store, loadu, storeu, load1, add, mul}_pd
- All multicore CPUs are Shared Memory Multiprocessors
  - Single address space shared by all cores
  - Coordination/communication through shared variables in memory
    - Shared data coordinated via synchronization primitives (locks)
- MOESI protocol for each block in cache:
  - Modified = up-to-date data, changed (dirty), no other cache has a copy, OK to write, memory out-of-date
  - Owner = up-to-date data, other caches may have a copy (they must be in Shared state)
    - Only cache that supplies data on read instead of going to memory
  - Exclusive = up-to-date data, no other cache has a copy, OK to write, memory up-to-date
    - Avoids writing to memory if block replaced
    - Supplies data on read instead of going to memory
  - Shared = up-to-date data, other caches may have a copy
  - Invalid = not in cache
- Memory in multi-threaded
  - All threads can access globally shared memory, but each thread also has private data
- Main bottleneck of SMP is the memory system
- Data race is when two serial memory accesses from different threads to same location and at least one is a write
- Locks create critical section where only one thread operates

- Synchronization in MIPS
  - Load linked ll
  - Store conditional sc
    - Returns 1 in rt if location has not changed since ll
    - Always clobbers value of rt (to 1 or 0)
- Each thread has a state/context (PC, register file, sp)
- Each thread shares memory address space
- Each process has its own address space and contains multiple threads
- Multithreading on single processor occurs by time-division multiplexing (with rapid context switching)
- Test-and-set in MIPS
  - Try: addiu $t0, $zero, 1
    - ll $t1, 0($s1) # load semaphore
    - bne $t1, $zero, Try #unlocked?
    - sc $t0, 0($s1) # try to own & lock semaphore
    - beq $t0, $zero, Try # successful?
  - Locked: critical section
    - sw $zero, 0($s1) # unlock semaphore

3 OpenMP

- In parallel for pragma, all variables declared outside the for loop are shared by default, except for loop index which is private per thread
- omp_{get, set}_num_threads(), omp_get_thread_num()
- #pragma omp parallel private(privateVarName)
- #pragma omp critical, #pragma omp master

4 Synchronous Digital Systems

- Remember adder propogation delays
- Register "D" is data, "Q" is output
  - Input data must be stable from start of "setup" time to end of "hold" time (around the rising edge of the clock)
  - "clk-to-q" delay is between rising edge of clk and correct output of q
- Max delay = setup time + clk-to-q delay + cl delay

5 Boolean algebra

- $+$ means OR, $\bullet$ means AND, $\overline{x}$ means NOT
- Laws of boolean algebra
  - Complementarity
    - $x \cdot \overline{x} = 0$
    - $x + \overline{x} = 1$
  - Laws of 0's and 1's
    - $x \cdot 0 = 0$
    - $x + 1 = 1$
  - Identities
    - $x \cdot 1 = x$
    - $x + 0 = x$
  - Idempotent law
    - $x \cdot x = x$
    - $x + x = x$
  - Commutativity, associativity, and distrubution also apply
  - Uniting theorem
    - $(x + y)\, x = x$
  - Uniting theorem 2
    - $(\overline{x} + y)\, x = xy$
  - DeMorgan's Law
    - $\overline{x \cdot y} = \overline{x} + \overline{y}$
    - $\overline{x + y} = \overline{x} \cdot \overline{y}$
- Truth table <-> boolean sum-of-products <-> gate diagram -> truth table
- overflow $= c_n$ XOR c_{n-1} for left-most adder
  - c_n is carry out, c_{n-1} is carry in
- Mux
  - Truth table for mux with 4-bits of signals controls 16 inputs, so it has 2^{20} rows in truth table

6 CPU Design

- Stages of the datapath
  1. Instruction fetch
     (a) Also where we increment PC
  2. Instruction decode
     (a) Read opcode and read data from necessary registers
  3. ALU (Execute)
     (a) Includes calculating the address of memory for lw and sw
  4. Memory access
     (a) Only lw and sw do anything at this stage
     (b) Expected to be fast because of cache, otherwise multicycle stall
  5. Register write
     (a) Idle for stores, branches, and jumps
- Load uses all 5 stages
- Registers between each stage to hold intermediate data and control signals
- Always include incrementing the PC in Reg Transfer Language, and always start by fetching the instruction
  - Ex: $\{op, rs, rt, Imm16\} \leftarrow MEM[PC]$
- Critical path is longest path through logic and determines length of clock period
- Control signals
  - nPCsel = 0 (next PC is PC + 4), 1 (branch), X (jump)
  - Jump = 1 (is a jump), 0
  - ExtOp = zero, sign
  - ALUsrc = 0 (regB), 1 (immed)
  - ALUctr = ADD, SUB, OR
  - MemWr = 1 (write memory), 0
  - MemToReg = 0 (ALU output goes to reg), 1 (Mem output goes)
  - RegDst = 0 (rt), 1 (rd)
  - RegWr = 1 (write register), 0
- Pipelining rate limited by slowest pipeline stage

- Hazards
  - Structural hazards
    - Required resource is busy (ie load or store)
    - Stalling instruction fetch for that cycle causes pipeline bubble
    - Fixed by writing to reg in first half of clock, then reading from reg during second half
  - Data hazard
    - Need to wait for previous instruction to complete its data read/write
    - Forward result from one stage to another (although load will still cause stall)
    - Load delay slot is the instruction after a load
      - If it uses result of load, then stall for 1 cycle
  - Control hazard
    - Fetching next instruction depends on branch outcome
    - Options are move branch comparator to stage 2, predict branch outcome and flush pipeline if wrong, or always execute the branch-delay slot (delayed branch)
- Multiple issue = start multiple instructions in multiple pipelines per clock cycle
  - Static multiple issue
    - Compiler reorders, groups instructions (and pads with nops if necessary) into packets
    - No dependencies within packets
    - Static dual isue has 1 ALU/branch instr and 1 load/store instr
  - Dynamic multiple issue allows CPU to execute instructions out of order to avoid stalls
    - Speculation: start by guessing what to do with instruction, then roll-back if guess was wrong

7 Virtual Memory

- Each process has its own page table, and OS changes page tables by changing Page Table Base Register
- Allows for sharing memory with protection and separate address spaces
- Page Table located in physical memory
  - Physical Page Number aka "Page Frame"

  - Cols are valid?, access rights, physical page address
  - Row index == virtual address' page number
  - Physical address is PPN, offset
- Translation Lookaside Buffer is cache of page table
  - VPN is split into TLB tag and index
  - Cols are TLB tag, PPN, dirty? (== need to write to disk when replaced), ref (to calculate LRU replacement)
  - Row index == index from VPN

8 I/O

- Memory Mapped Input/Output dedicates portion of address space to communication with devices
  - Polling: processor reads from control register in loop until it is ready, then writes/reads data register, which resets ready bit of control register
- Exceptions arise within CPU
  - PC of offending instr is saved in Exception Program Counter, cause is saved in Cause register, and jumps to exception handler code at 0x8000 0180
    - After exception, pipeline flushed, handler executed, then instr executed from scratch or program terminated
    - Precise exceptions (actually used)
      - Earliest exception-causing instr is handled first
    - Imprecise exceptions
      - Pipeline stopped, software handler works out cause and what to do
- Interrupts from external IO controller
  - Asynchronous, but does not prevent any instruction from completion
- Switched network has higher bandwidth than shared network
- Mean Time to Failure
- RAID 3 = Sequential bytes on different drives, dedicated parity drive
- RAID 5 = rotated parity, faster small writes

9 Amdahl's Law

- $$f(n) = \frac{1}{(1-P) + \frac{P}{N}}$$

- P is percentage of parallelizable code
- N is number of cores used
- f(n) is amount of speedup code gains
- Assumptions
  - No contention for shared resources (ideal!)
  - No per-thread overhead (ideal!)
  - No pipelining