|  |  |  |  |
| --- | --- | --- | --- |
| Logic Gates & Timing Diagram   * Clock cycle begins on the rising edge * One cycle last from rising edge to rising edge * Gates create delay | Combinational logic:   * Doesn’t depend on itself (no loops) * Updates nearly instantly   Examples:   * 1-Bit Full Adder: Carry-in, Carry-out * Multiplexers: S input, ⎡log 2 (S)⎤, control signals | Sequential logic:   * State depends on itself (has loops) * Exhibits propagation delay * Changes on rising edge of inputs (clock)   Examples:   * SR Latch, D-latch, flip-flops | We need a way to represent instruction like ADD X0,X0,X1 in binary  ● Opcodes are unique per instruction  ● Since we have 32 registers, we need log 2 (32)= 5 bits for Rm/Rn/Rd  ● To use numbers in our instructions, we use “immediates” ● We only have so many bits for immediates ● Here, we have 11 bits available, so we can represent up to 2 11 = 2048 values |
| Pipelined Model   * Split each stage into its own cycle   + Use the stage with the longest delay as clock cycle time * Allows for multiple stages to execute at the same time, but for different instruction   Non-pipelined:   * CC = sum of stages * Latency = sum of stages   Pipelined:   * CC = longest stage * Latency = longest stage \* num of stages * Pipeline registers have register * overhead | We currently have a five-stage circuit, where the delay of each stage is  250ps, 150ps, 25ps, 200ps, and 75ps, respectively.  1. How long is the clock cycle if the circuit is non-pipelined? **We add up all the stage delays: CC = 250ps + 150ps + 25ps + 200ps + 75ps = 700ps**  2. How long is the clock cycle if the circuit is pipelined? **Use the stage with the longest delay as clock cycle time: CC = 250ps**  3. With the pipeline above, what’s the latency of one pass of data? **The data must go through all the stages: 250ps \* 5 = 1,250ps**  4. What would the latency be if we considered pipeline registers with  20ps after each stage? **We add the pipeline overhead to the pipeline latency: (250ps + 20ps) \* 5 = 1,350ps** | Cache   * The memory-subsystem is much too slow for efficient data-access in modern processors. * To rectify this, modern processors ship with SRAM data caches, typically split into 2 or 3 levels. * These caches help avoid memory accesses on frequently-used data (most programs only access small pools of data repeatedly anyway).   Cache Hit/Miss   * If the data is already in the cache (set index and tag are the same and valid bit = 1) then you have a hit. * Otherwise, you have a miss and you bring B bytes into the cache * Miss Rate = Number of cache misses / Total number of references * Lower miss rate is better. Try to limit the number of cache misses | Cache Organization   * S = number of sets * E = number of lines per set * B = number of bytes per line * The capacity of a cache: C = S\*E\*B * S = 2^s or s = log2 (S) * B = 2^b or b = log2 (B) * Tag is the rest of the bits * m = t + s + b   Cache Types   * It is possible to reduce cache misses by making the cache associative. * In a direct mapped cache (E = 1), each set contains just a single line, * making conflict misses prevalent. * In an N-way associative cache (E = N), each set contains N lines. Modern * processor caches are usually 4-way to 16-way associative. In the event of a conflict, an algorithm is used to determine which line is evicted (typically Least-Recently-Used). * In a fully associative cache, all lines are on a single set. This is rarely implemented in real hardware due to cost. |
| Starting from an empty cache, is it possible to get a miss rate of 0 (assuming there is at least one reference)?  **No, your first reference will always be a miss.**  **This is because it is not possible to find data with the same set index and tag in the cache if it is empty.** | In an 8-way associative cache with 24 double-word blocks, where 16 bits  are used for memory addresses. The cache is byte-addressed.  1. How many bits are used for the set index? **We need s=2 bits.**  2. How many bits are used for the byte offset? **We need b=3 bits.**  3. How many bits are used for the tag? **We need t=11 bits.** | ● To start off, we need to figure out the relevant information and fill  in the cache table:  ● E = 8 ; “8-way associative cache”  ● m = 16 ; “16 bits are used for memory addresses”  ● B = 8 ; “24 double-word blocks”  ● S = 24/8 = 3 ; “24 double-word blocks”, “8-way associative cache”  We can fill in the rest:  ○ s = ⌈log2(S)⌉ = 2  ○ b = log2(B) = 3  ○ t = m – (s + b) = 11  ○ C = B \* E \* S = 192 |

A black and white rectangular frame with text

Description automatically generatedA screenshot of a computer program

Description automatically generatedA screenshot of a computer error

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generatedA diagram of a clock cycle

Description automatically generatedA diagram of a diagram

Description automatically generated with medium confidenceA diagram of a graph

Description automatically generatedA diagram of a diagram

Description automatically generatedA group of black arrows

Description automatically generatedA diagram of a computer code

Description automatically generatedA computer screen shot of a computer code

Description automatically generated with medium confidence