Computer Architecture (ECS 154A) Study Guide

Davis Computer Science Club — Tutoring Committee

Winter Quarter 2015

1 Boolean Algebra

1.1 Operations

Operation	Symbol	Example
NOT	$\bar{}$	$\bar{A}$
	!	$!A$
	$\neg$	$\neg A$
	$\sim$	$\sim A$
AND	$\wedge$	$A \wedge B$
	$*$	$A * B$
		AB
OR	$\vee$	$A \vee B$
	$+$	$A + B$
XOR	$\oplus$	$A \oplus B$

1.2 Equivalence Laws

Name	OR Version	AND Version
Commutative	$A + B = B + A$	$A * B = B * A$
Associative	$(A + B) + C = A + (B + C)$	$(A * B) * C = A * (B * C)$
Distributive	$A + (B * C) = (A + B) * (A + C)$	$A * (B + C) = (A * B) + (A * C)$
Identity	$A + 0 = A$	$A * 1 = A$
Annulment	$A + 1 = 1$	$A * 0 = 0$
Idempotent	$A + A = A$	$A * A = A$
Complement	$A + \bar{A} = 1$	$A * \bar{A} = 0$
De Morgan's	$\overline{(A + B)} = \bar{A} * \bar{B}$	$\overline{(A * B)} = \bar{A} + \bar{B}$

1.3 Truth Table

Truth tables are mathematical tables composing of every combination of inputs and the resulting function. The number of combinations or rows in the truth table is 2^N, where N is the number of inputs.

1.3.1 NOT

A	$f(A) =!A$
0	1
1	0

1.3.2 AND

A	B	$f(A, B) = A * B$
0	0	0
0	1	0
1	0	0
1	1	1

1.3.3 OR

A	B	$f(A, B) = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

1.3.4 XOR

A	B	$f(A, B) = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

1.4 Canonical Normal Form

1.5 Karnaugh Maps

1.6 Quine-McClucksey Algorithm

2 Combinational Logic Circuits

2.1 Gates

2.2 Timing Diagrams

2.3 Multiplexers, Decoders, Shifters

2.4 Adders and Subtracters

2.5 Designing Combinational Logic Circuits

3 Finite State Automata

3.1 Moore Model

3.2 Mealy Model

4 Sequential Logic Circuit

4.1 Latches

4.2 Flip Flops

4.3 Registers and Counters

4.4 Designing Sequential Logic Circuits

5 Single Cycle CPU Design

6 Cache

Effectiveness is based on the concept of information reuse: temporal locality and spatial locality.

6.1 Memory Hierarchy

Goal: Make memory perform as if it was made of the most expensive and fastest type, but cost as if made of the cheapest type.

1. Fast, Hot, Expensive
2. Static RAM
3. Dynamic RAM
4. Disk

A cache is smaller than main memory and is composed of numerous cache lines. Cache lines consist of a dirty bit, a tag, and block(s) of data. If the dirty bit is on, then it signals the CPU to write the data from this cache line into main memory when this cache line is freed.

Dirty Bit	Tag	Block(s)

6.2 Direct Mapped Cache

A given address is partitioned into three components: Tag, line number, and offset. The line number directly accesses a specific cache line. It then compares the tag partitioned from the address to the tag stored in the cache line. If the tags match, then it is a cache hit. Otherwise, it becomes a cache miss. Assuming that it was a cache hit, it then proceeds to use the offset to select which block to read or write. The block of data in the cache line can be thought of as an array and the offset as an index into this "array".

6.2.1 Format

Tag	Line Number	Offset

6.2.2 Example

A CPU is using 24-bit addresses and is byte-addressable. Each line in cache holds 16 bytes of data and each block is 1 byte. Assuming that the tag is 12 bits wide, find the following: number of lines in cache, size of cache, size of tag, and sizes of each partition in the format.

$$\text{Size(Cache Line)} = 16 \text{ bytes}$$

$$\begin{aligned} \text{Offset} &= \text{Number of bits needed to address 16 blocks} \\ &= \log_2 16 \\ &= 4 \text{ bits} \end{aligned}$$

$$\begin{aligned} \text{Bit\_Width(Line Number)} &= \text{Size(Address)} - \text{Bit\_Width(Tag)} - \text{Bit\_Width(Offset)} \\ &= 24 \text{ bits} - 12 \text{ bits} - 4 \text{ bits} \\ &= 8 \text{ bits} \end{aligned}$$

$$\begin{aligned} \text{Number of Cache Lines} &= 2^{\text{Bit\_Width(Line Number)}} \\ &= 2^8 \\ &= 256 \end{aligned}$$

$$\begin{aligned} \text{Size of Cache} &= \text{Number of Cache Lines} * \text{Size of Cache Lines} \\ &= 2^8 * 2^4 \\ &= 2048 \text{ bytes} \\ &= 2 \text{ KB} \end{aligned}$$

Format Partition Sizes and Bit Range

	Tag	Line Number	Offset
Size	12 bits	8 bits	4 bits
Bit Range	Bits 12-23	Bits 4-11	Bits 0-3

7 Virtual Memory

8 Multi-Cycle CPU Design

9 Pipeline CPU Design