Computer Architecture (ECS 154A) Study Guide

Davis Computer Science Club — Tutoring Committee

Winter Quarter 2015

Contents

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

1.3.5 Example

Create a function that takes three inputs — x2, x1, x0 — and produces a high or on signal when only two input signals are on.

x2	x1	x0	$f(x2, x1, x0)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

1.4 Canonical Disjunctive Normal Form

1.4.1 Minterm

The canonical disjunctive normal form is also known as a minterm, typically represented by a lower case 'm'. A minterm is a logical product term such that it uses each input variable once and it uses only the complement operator and conjunction operation. In other words, if there are n input variables, the minterm must keep to the following constraints:

1. be a logical product that evaluates to logically true
2. use each of the n variables once
3. only use NOT operators and AND operators

For three inputs, the minterms are as follows:

x_2	x_1	x_0		Minterm (Product Term)
0	0	0	m_0	$\bar{x_2}\bar{x_1}\bar{x_0}$
0	0	1	m_1	$\bar{x_2}\bar{x_1}x_0$
0	1	0	m_2	$\bar{x_2}x_1\bar{x_0}$
0	1	1	m_3	$\bar{x_2}x_1x_0$
1	0	0	m_4	$x_2\bar{x_1}\bar{x_0}$
1	0	1	m_5	$x_2\bar{x_1}x_0$
1	1	0	m_6	$x_2x_1\bar{x_0}$
1	1	1	m_7	$x_2x_1x_0$

Note: Product terms all evaluate to be logically true.

For a given function, f, it is possible to express the function as a "sum of products". This is a special form of the canonical normal form in that it only includes terms such that it makes the function logically true.

1.4.2 Example

Create a function that takes three inputs — x2, x1, x0 — and produces a high or on signal when only two input signals are on. Express this function as a sum of products.

	x2	x1	x0	$f(x2, x1, x0)$
	0	0	0	0
	0	0	1	0
	0	1	0	0
m_3	0	1	1	1
	1	0	0	0
m_5	1	0	1	1
m_6	1	1	0	1
	1	1	1	0

$$\begin{aligned} f(x_2, x_1, x_0) &= m_3 + m_5 + m_6 \\ &= (\bar{x_2}x_1x_0) + (x_2\bar{x_1}x_0) + (x_2x_1\bar{x_0}) \end{aligned}$$

1.5 Canonical Conjunctive Normal Form

1.5.1 Maxterm

The canonical conjunctive normal form is also known as a maxterm, typically represented by a lower case 'M'. A maxterm is a logical sum term such that it uses each input variable once and it uses only the complement operator and disjunction operation. In other words, if there are n input variables, the minterm must keep to the following constraints:

1. be a logical sum that evaluates to logically false
2. use each of the n variables once
3. only use NOT operators and OR operators

For three inputs, the maxterms are as follows:

x_2	x_1	x_0		Maxterm (Sum Term)
0	0	0	M_0	$x_2 + x_1 + x_0$
0	0	1	M_1	$x_2 + x_1 + \bar{x_0}$
0	1	0	M_2	$x_2 + \bar{x_1} + x_0$
0	1	1	M_3	$x_2 + \bar{x_1} + \bar{x_0}$
1	0	0	M_4	$\bar{x_2} + x_1 + x_0$
1	0	1	M_5	$\bar{x_2} + x_1 + \bar{x_0}$
1	1	0	M_6	$\bar{x_2} + \bar{x_1} + x_0$
1	1	1	M_7	$\bar{x_2} + \bar{x_1} + \bar{x_0}$

Note: Sum terms all evaluate to be logically false.

For a given function, f, it is possible to express the function as a "product of sums". This is a special form of the canonical normal form in that it only includes values such that it makes the function logically false.

1.5.2 Example

Create a function that takes three inputs — x2, x1, x0 — and produces a high or on signal when only two input signals are on. Express this function as a product of sums.

	x2	x1	x0	$f(x2, x1, x0)$
M_0	0	0	0	0
M_1	0	0	1	0
M_2	0	1	0	0
	0	1	1	1
M_4	1	0	0	0
	1	0	1	1
	1	1	0	1
M_7	1	1	1	0

$$
\begin{aligned}
f(x_2, x_1, x_0) &= M_0 * M_1 * M_2 * M_4 * M_7 \\
&= (x_2 + x_1 + x_0) * (x_2 + x_1 + \bar{x_0}) * (x_2 + \bar{x_1} + x_0) * (\bar{x_2} + x_1 + x_0) * (\bar{x_2} + \bar{x_1} + \bar{x_0})
\end{aligned}
$$

1.6 Karnaugh Maps

Karnaugh maps are a visual methodology to simplifying boolean expressions. When setting up the Karnaugh map, input values cannot differ more than one hamming distance. The resulting expressions are either a sum of products or a product of sums.

1.6.1 Example (Three Variables)

Find a minimum sum of products expression for $f(x_2, x_1, x_0) = m_1 + m_4 + m_5 + m_7$**.**

x_2 \ x_1x_0	00	01	11	10
0	0	1	0	0
1	1	1	1	0

$$f(x_2, x_1, x_0) = (\bar{x_1}x_0) + (x_2\bar{x_1}) + (x_2x_0)$$

1.6.2 Example (Four Variables)

Find a minimum product of sums expression for $f(x_3, x_2, x_1, x_0) = m_0 + m_2 + m_6 + m_8 + m_9 + m_{10} + m_{14} + D_{15}$**.**

x_3x_2 \ x_1x_0	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	0	0	X	1
10	1	1	0	1

$$f(x_3, x_2, x_1, x_0) = (x_3 + \bar{x_0}) * (\bar{x_2} + x_1) * (\bar{x_1} + \bar{x_0})$$

1.6.3 Example (Five Variables)

1.7 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. Caches also contain a valid bit which signifies whether there is loaded data into the cache — imagine starting up the computer for the first time, the cache is going to be empty. The issue is that even when a cache is empty (bits are all 0), it still holds some signal. As we continue, when we mention the size of a cache line, we refer to the size of the data blocks in the cache line only (excluding flags and tag).

Flags	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.

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

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

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

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

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

Format Partition Sizes and Bit Range

	Tag	Line Number	Offset
Size	12 Bits	8 Bits	4 Bits
Bit Range	[12,23]	[4,11]	[0,3]

The bit range represents what bits that partition occupies. Ex: Offset occupies bits 0, 1, 2, and 3. The four least-significant bits.

6.3 Fully Associative

A given address is partitioned into two components: Tag and offset. The reason why a line number is unused is because the CPU will perform a linear search through all cache lines looking for either a cache hit, an unused cache line, or a cache line to replace. Because of this, using full association has the lowest miss rates. The tag and offset performs the same as the tag and offset in direct mapping.

6.3.1 Format

Tag	Offset

6.3.2 Example

A CPU is using 11-bit addresses and is byte-addressable. The cache size is 128 bytes and the size of each cache line is 8 bytes. Find the following: Bit width of offset, bit width of tag, number of cache lines, and sizes and bit ranges of the partitions.

$$\begin{aligned}
\text{Bit_Width(Offset)} &= \text{Number of bits needed to address 8 blocks} \\
&= \log_2 8 \\
&= 3 \text{ bits}
\end{aligned}$$

$$\begin{aligned}
\text{Bit_Width(Tag)} &= \text{Size(Address)} - \text{Bit_Width(Offset)} \\
&= 11 \text{ bits} - 3 \text{ bits} \\
&= 8 \text{ bits}
\end{aligned}$$

$$\begin{aligned}
\text{Number of Cache Lines} &= \text{Size(Cache)/Size(Cache Line)} \\
&= 128 \text{ bytes}/8 \text{ bytes} \\
&= 16 \text{ Cache Lines}
\end{aligned}$$

Format Partition Sizes and Bit Range

	Tag	Offset
Size	8 Bits	3 Bits
Bit Range	[3,10]	[0,2]

6.4 Set Associative

Set associative caching is a hybrid between direct-mapped and fully associative. A given address is partitioned into three components: Tag, set number, and offset. What makes set associative different from direct-mapped caching is that a given address will access a specific set of cache lines instead of a single cache line. Within that set of cache lines, it will perform a linear search like the fully associative cache. The tag and offset perform the same as the direct-mapped cache and fully associative cache. **Important terminology:** N-way associative cache means that there are N cache lines per set. This is also known as the associativity.

6.4.1 Format

Tag	Set Number	Offset

6.4.2 Example

A CPU is using 64-bit addresses and is byte-addressable. It also uses a 3-way set associative cache with a cache size of 98,304 bytes and 32 sets. Find the following: Number of lines per set, size of each set, size of cache line, bit width of offset, bit width of set number, bit width of tag, and partition sizes and bit ranges.

$$\text{Number of Lines per Set} = 3 \text{ (Given)}$$

$$\begin{aligned} \text{Size(Set)} &= \text{Size(Cache)/Number of Sets} \\ &= 98304/32 \\ &= 3072 \text{ bytes} \end{aligned}$$

$$\begin{aligned} \text{Size(Cache Line)} &= \text{Size(Set)/ Number of lines per set} \\ &= 3072 \text{ bytes}/3 \\ &= 1024 \text{ bytes} \end{aligned}$$

$$\begin{aligned} \text{Bit\_Width(Offset)} &= \text{Number of bits needed to represent 1024 bytes} \\ &= \log_2 \text{Size(Cache Line)} \\ &= \log_2 1024 \\ &= 10 \text{ bits} \end{aligned}$$

$$\begin{aligned}
\text{Bit_Width(Set Number)} &= \text{Number of bits needed to represent 32 sets} \\
&= \log_2 \text{Number of sets} \\
&= \log_2 32 \\
&= 5 \text{ bits}
\end{aligned}$$

$$\begin{aligned}
\text{Bit_Width(Tag)} &= \text{Size(Address)} - \text{Bit_Width(Set Number)} - \text{Bit_Width(Offset)} \\
&= 64 \text{ bits} - 5 \text{ bits} - 10 \text{ bits} \\
&= 49 \text{ bits}
\end{aligned}$$

Format Partition Sizes and Bit Range

	Tag	Set Number	Offset
Size	12 Bits	5 Bits	10 Bits
Bit Range	[15,63]	[10,14]	[0,9]

6.4.3 Associativity Remarks

It can actually be noted that direct-mapped caching and fully associative caching are a form of set associative cache. Suppose that our cache has N lines and we use an 1-way set associative cache. In this scenario, it would imply that each set consists of one cache line. This would mean that the set number is the exact same as a line number. Therefore, 1-way set associative caches are also direct mapped caches.

Suppose that our cache has N lines and we use an N-way set associative cache. This implies that there is a single set with N lines. Recall the format of the set associative — tag, set number, and offset. If there is only one set, how many bits are required to represent one set. $\log_2 1 = 0$. Therefore, there is no need for any bits to represent that one set. In this case, the format becomes – tag and offset. Also, recall that in set associative, it first uses the set number to access a specific set and then performs a linear search among the cache lines within that set. Since there is only one set, it performs the linear search over that entire set by default. Therefore, an N-way set associative cache is also a fully associative cache.

6.5 Cache Properties

6.6 Advance Caches

7 Virtual Memory

7.1 Virtual Address Format

Virtual Page Number	Offset

7.2 Page Table Entry Format

Flag Bits	Protection Bits	Physical Frame Number

7.3 Example

Given a 1 Megabyte physical memory, a 30 bit virtual address, and a page size of 8 KB, find the following: number of page frames, bit width of offset, number of entries in the page table and the width of each entry.

$$\begin{aligned}
\text{Number of Page Frames} &= \text{Size(Physical Memory)/Size(Page)} \\
&= 2^{30} \text{ bytes}/2^{13} \text{ bytes} \\
&= 2^{17} \text{ page frames}
\end{aligned}$$

$$\begin{aligned}
\text{Bit_Width(Offset)} &= \text{Number of bits needed to reference 8 KB} \\
&= \log_2 2^{13} \\
&= 13 \text{ bits}
\end{aligned}$$

Virtual Address Partition Sizes

	Virtual Page Number	Offset
Size	49 Bits	13 Bits

$$\begin{aligned}
\text{Number of Page Table Entries} &= \text{Number of Virtual Pages} \\
&= 2^{\text{Bit_Width(Virtual Page Number)}} \\
&= 2^{49}
\end{aligned}$$

$$\begin{aligned}
\text{Bit_Width(Page Table Entry)} &= \text{Number of needed to reference } 2^{17} \text{ page frames} \\
&= \log_2 2^{17} \\
&= 17 \text{ bits}
\end{aligned}$$

8 Multi-Cycle CPU Design

9 Pipeline CPU Design