Brian Chen
chen187
1002297034
University of Toronto, Scarborough Winter 2017 CSCB58

# CSCB58 – Computer Organization Finals Notes

## Week 1 – Introductory and Electricity

**Waveforms –** Similar to truth tables, but associate time as well
**Hardware** is different from **Software** design because we need to describe circuits, connections, logic gates, etc.
**Logic –** Logic-0 is 0v, Logic-1 is 5v, we differentiate between True and False this way.
**Voltage –** Electric particles want to flow from high electric potential to low electric potential. Potential is voltage
**Current** – Rate of the flow (voltage)
**Ground –** Current flows towards the zero voltage point of a circuit
**Resistance** – The relationship between current and voltage – R = V/I

- **Insulators –** Things that have high resistance, don't conduct electricity at all
- **Conductors –** Low resistance, conducts electricity and are used for wires
- **Resistance is measured in Ohms, Ω**
- **Semiconductors** - Somewhere between conductors and insulators

**Semiconductors** do not conduct electricity naturally – we have to dope them. We introduce impurities to increase the number of free charge carriers – **Doping** means introducing new elements

- **n-type** adding elements which have 5 electrons in valence layer
- **p-type** adding elements which have 3 electrons in valence layer

**N-type** semiconductors are electrons that are not bound
**P-type** semiconductors are holes to represent absence of electrons
**Depletion layer** – What happens when p-n combine, the electrons from n mix with holes of p, cancelling each other out. This section is the depletion layer

- The original current caused by the hole/electron recombination is called the **Diffusion Current**
- Once this layer is wide enough, doping atoms that remain create an **electric field** in that region
- The electric field causes holes to go back to p, unbound electrons to go back to n called **Drift Current**

**Forward Bias –** When positive voltage is inserted to p junction, electrons go to n junction

- Narrows the depletion layer and increases electron diffusion current

**Reverse Bias –** When positive voltage is inserted to the n junction,

- Widens the depletion layer, electrons go through with more difficulty

**Transistors –** The basis of forward bias (short circuit) and reverse bias (open circuit)

A – B
  C
If C is on, then A and B are connected. If C is off, then A and B are not. This is an example of a transistor

## Week 2 – Circuit Creation

**Creating circuits:**

1. Create truth tables
2. Express as Boolean expressions
3. Convert to gates

NPN transistors → Ones without the dot, connected when high, disconnected when low
PNP transistors → NPN transistors with the dot, connected when low, disconnected when high
**Minterms →** List which outputs are high, and AND them together (e.g. for 1101 we have A*~B*C*D), OR all of these
**Maxterms →** List which outputs are low, and OR them together (e.g. for 1101 we have A+~B+C+D), AND all of these
**Sum of Minterms (SOM):** We OR the minterms together (ABCD)+(AB~CD) etc. this represents our expression
**Product of Maxterms (POM):** We AND the maxterms together (A+B+C+D)(A+C+D) etc.
We want to reduce to NAND gates, as they are the cheapest available.
We call this the **simplest expression → lowest Gate Cost (G) or lowest gate cost with NOTs (GN)**
We use **Karnaugh Maps (KMaps)** to find the simplest expression for any given circuit.

- ➔ Write truth table
- ➔ Simplify using SOM or POM
- ➔ Translate into KMap
- ➔ Translate into circuits

## Week 3 – Logical Devices

**Multiplexers (MUX)** – Selector signals. Constructed using (S*X) or (~S*Y) (so that only one of them is true) 2-1mux
**Adders** – Small circuit devices that add two digits together (binary)

- ➔ Half adders – two input, one bit width adder that computes given X, Y, C and S, which are C = Carry and S = Sum bit respectively. C = (X*Y)  S=(X xor Y)
- ➔ Full-adders are the same as half adders except they take in a carry bit from the previous, so given inputs X, Y, Z, it computes C, S. Z is sometimes called $C_{in}$ and C is sometimes $C_{out}$. C = (X*Y) + (X*Z) + (Y*Z) Z=(X xor Y xor Z)
- ➔ Ripple carry binary adder – uses multiple chained full adders to compute numbers.

**Subtraction –** Take 1s complement of the item being subtracted (e.g. X – Y, take !Y), then add 1 to it to get 2s complement. Add X to !Y+1 and then see if the carry bit is 1 or 0. If carry bit is 1, then sign bit is low (0), if carry bit is 0, then sign bit is high (1). E.g. 53-27 = 26 = 00011010 **whereas** 27-53 = -26 = 11100110 -> -00011010 -> -26

Brian Chen
chen187
1002297034

**Comparators** – Takes in two inputs A, B, and returns if A==B, A>B, or A<B

- ➔ A == B = A*B + ~A*~B
- ➔ A > B = A*~B
- ➔ A < B = ~A*B

**Decoders –** Think about how we used our HEX Display decoder, HEX0 is high for values of 0000, 0010, etc. (think almost like a sum of minterms)

# Week 4 – Sequential Circuits

**Sequential Circuits –** Having feedback in the circuit, result of having the output be connected to its own input again
**Latches –** Circuits that are combined for feedback to have steady behavior (simple sequential circuits are not stable) – we store bits
**SR latch –** Set/Reset Latch – we have forbidden states due to unstable behavior. A contradiction occurs where Q = ~Q, so we only take which signal changes first, which may vary. We **forbid** these states

- ➔ We have two latches, NOR SR Latch and NAND SR Latches
- ➔

**Clocks –**Are regular signals where high values indicate the output of a latch may be sampled
**Clocked SR Latch –** Add a clock signal, C, to SR latches that S signal originally becomes (NAND(S, C)) and R becomes (NAND(R, C))
**D Latch –** Prevents both R and S from going high so that we can forbid the states. Simply make S, R, stem from D where R becomes !D and S is just D.into the clocked SR latch.
**Flip-Flop** – A latched circuit whose output is triggered with rising and falling edge of a clock pulse
**We use positive-edge triggered flip-flops**

# Week 5 – Sequential Circuit Design

**Shift Registers –** A series of D flip-flops that can shift values in the register 1 bit at a time
**Load Registers –** A series of D flip flops that can load a large bitwise value all at once

- ➔ D flip flop with enable – controls load register when it can load its values

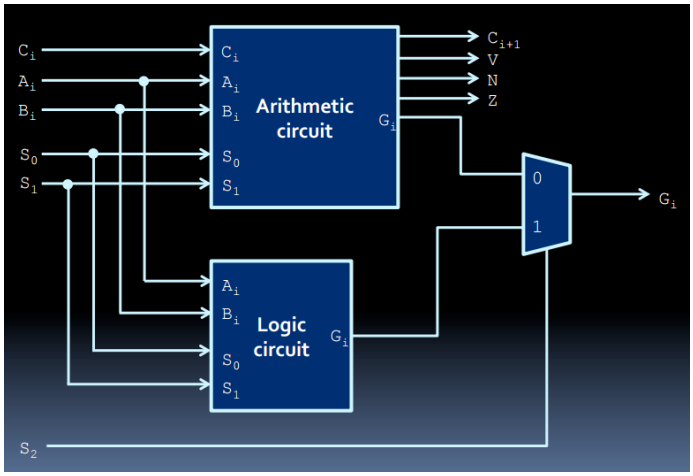
**Finite State Machines –** FSMs depict an abstract model that captures operation of a sequential circuit with flip flops as states. Each flip flop can store 2 states, so we need $\log_2 n$ flip flops to store n states.
**Moore Machine –** The output for the FSM depends on current state
**Mealy Machine –** The output for the FSM depends on state and the input (so in state X can produce different outputs depending on how it got to state X)
**Timing and State assignments** –

1. Make it so that each neighboring flip flop differs by one flip flop value, if possible
2. Add intermediate transition states

# Week 6 – Processor Components

**Microprocessors** combine everything, registers to store values, adders and shifters to process data, FSMs to control processing
**Processors are –** Storage thing, Arithmetic thing controlled by a controller thing. We know storage thing = registers, controller thing = FSMs, now the arithmetic things are ALUs
**ALUs – Arithmetic Logic Units-**
Components:

- ➔ Inputs (A, B)
- ➔ Operation and Mode selects (S)
- ➔ Carry Input ($C_{in}$)
- ➔ Data Output (G)
- ➔ Carry Output ($C_{out}$)
- ➔ Overflow Indicator (V)
- ➔ Negative Indicator (N)
- ➔ Zero Indicator (Z)




We have $S_2$ as the selector for L or A (logic or arithmetic), and we can have the carry bit flow in to give specific values too to arithmetic. We then have G as the output, and VHCZ as our specifications for our arithmetic.
**Booth's Algorithm –** Product Register will be 2x width of largest item
Let A = $0100_2$ and B = $0111_2$, we have A x B =
-B = $1001_2$ (2s complements)
Set up a table

| Iteration | Step | | | | | | | | | L | Description |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | **0** | **1** | **0** | **0** | 0 | Init |
| 1 | 1.00 | | | | | | | | | | No action |
| 1 | 2 | 0 | 0 | 0 | 0 | 0 | **0** | **1** | **0** | 0- | Shift |
| 2 | 1.00 | | | | | | | | | | No action |
| 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | **0** | Shift |
| 2 | 1.10 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | **1** | **0** | Subtract |
| 3 | 2 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | **0** | **1** | Shift |
| 4 | 1.01 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | **0** | **1** | Addition |

Brian Chen
chen187
1002297034

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 2 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | **0** | **0** | Shift |

Basically – LHS = 0s to start, RHS = A, we shift until we shifted len(max(A, B)) times. If rightmost bit of A and L are (0, 1) then add B to LHS, otherwise if (1, 0), then subtract B (or + 2s complement B) to LHS, else if (0, 0), just shift it.

# Week 8 – Processor Storage and Control

**Registers –** Small number of fast memory units that allow values to be read/written
**Memory –** Larger grid of memory cells that are used to store main info to be processed by CPU
Typical setup- MIPS (32 reigsters, each register having 5 bit addressing for 32 bits).
**Memory** is made up of a decoder and rows of memory units
Each row of memory units are made up of *n* storage cells (RAM cells with select and a SR latch inside of it)
Since some busses will be used for input **and** output, we have to use a **Tri-State Buffer** gate. There's a WE (Write Enable)
Reading: Access Time (stable before can be read), Output Hold Time (held after address change)
Writing: Address setup time, Address setup time to write end, Data setup to write end, data hold from write end
**Registers** are meant to be used locally for execution, **Memory** is meant to be used to house large data values

**Control Words –** Information needed to process data and execute program(s)
**Instruction Register –** Takes 32 bit instruction fetched from memory and reads first 6 bits – **OpCode** and performs that operation
**MIPS Instruction Types –**

- ➔ R-Type **OpCode(6) rs(5) rt(5) rd(5) shamt(5) funct(6)**
- ➔ I-type **OpCode(6) rs(5) rt(5) immediate(16)**
- ➔ J-type **OpCode(6) address(26)**

For **R-Type** instructions, OpCode = 000000 and the last 6 digits of funct() are what operation it is