CS 221 Computer Architecture

Week 12: Sequential Networks

Fall 2001

Course Schedule

W1	Sep 12- Sep 15	Introduction	
W2	Sep 19- Sep 22	Information Representation (1)	(Chapter 3)
W3	Sep 26- Sep 29	Information Representation (2)	(Chapter 3/4)
W4	Oct 3- Oct 6	Computer Architecture Pep/6	(Chapter 4)
W5	Oct 10- Oct 13	Assembly Language	(Chapter 5)
W6	Oct 17- Oct 20	High-Level Languages (1)	(Chapter 6)
W7	Oct 24- Oct 27	High-Level Languages (2)	(Chapter 6)
W8	31 Oct	Midterm	
	Nov 2 - Nov 3	Instruction Sets	(Tanenbaum)
W9	Nov 7 - Nov 10	Devices & Interrupts	(Chapter 8)
W10	Nov 14- Nov 17	Storage Management	(Chapter 9)
W11	Nov 21- Nov 24	Combinational Logic	(Chapter 10)
W12	Nov 28- Dec 1	Sequential Logic & Review	(Chapter 11)

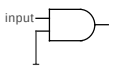
Plan for week 12

- **This weeks material**
  - Combinational Devices
  - Sequential networks
    - Latches and Clocked Flip-flops
    - Sequential Analysis & Design
- **Readings**
  - Chapter 11

I. Combinational Devices

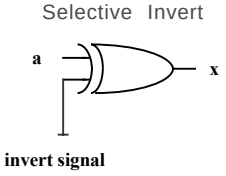
Combinational Devices

- **I will now describe some of the combinational devices typically used in computers.**
- **Several of these have an enable line.**
  - Acts as an on/off switch of an appliance
  - If enable is 0, output is 0 regardless of input
  - If enable is 1 output is according to function
- **Enables are implemented using a simple AND gate**
  - One ANDs each input with an on/off signal
  - Any existing AND gate can be controlled this way by adding an extra control input.



Selective Inverter

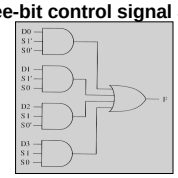
- **A selective inverter inverts the input if the control line is 1.**
  - When control line is 0, a passes unchanged.

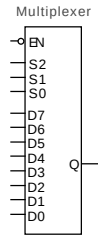


Multiplexer

- **A multiplexer routes one of several data inputs D to a single output Q.**
- **It uses a three-bit control signal S to do so.**

S2	S1	S0	Q
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7





Example use: STORER instruction

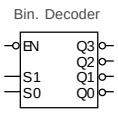
–Stores contents of register into memory
–Two-input multiplexer would make the selection
–Select line S would come from register specifier
–Input would come from a specific register bit line
–and output goes to bus.

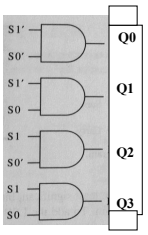
Binary Encoder

- **A multiplexer sets one of several output lines Q to 1, selected by an (in this case) two-bit binary input.**

S1	S0	D0	D1	D2	D3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

(b) Truth table.



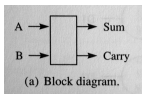


Example use: Opcode Parsing

First five bits in IR could be connected to a 5 x 32 bin decoder
Each output line would trigger circuitry for that instruction

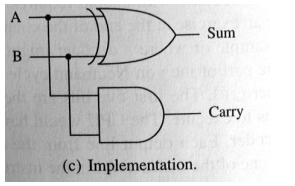
Half Adder

- **Adds bit A with bit B, outputting sum and carry.**



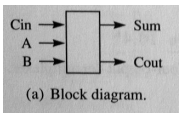
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Truth table.

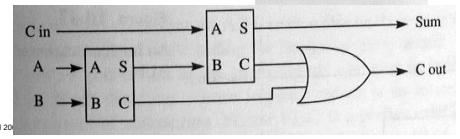


Full Adder= 2 Half Adders

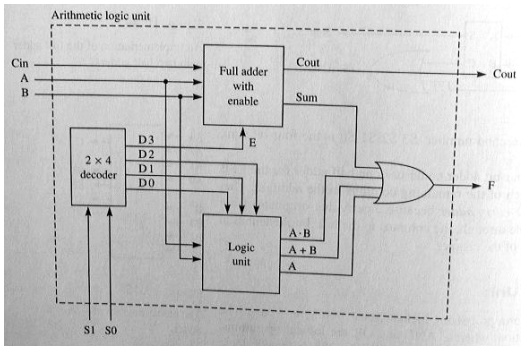
- **Adds bit A with bit B, outputting sum and carry, taking into account a carry from a less sign. bit.**



A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Arithmetic and Logic Unit



Building a Computer

- **Although this is beyond the scope of this class, you can see how combinational devices**
  - Implemented using OR and AND gates
  - Can be used to build a fully functional computer
- **However, how could this function without memory???**

II.
Sequential Networks

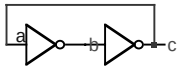
Latches & Flip-Flops

Sequential Networks

- **Combinational devices are useful and used in computers today to calculate things.**
  - Output depends on input only
  - However, since they do not have memory, the computer would immediately forget everything!
  - All your calculations are directly lost…
  - We therefore need networks that preserve a state
- **Such logic networks are called Sequential**
  - This is because state is preserved by continuous repetition through time.
  - They use the same gates as combinational devices
  - But apply feedback loops to keep the net active
  - Output depends on state of network…

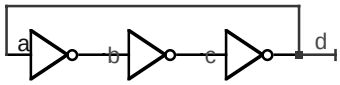
Latches and Clocked Flip-Flops

- **In combinational nets, the output goes to the input of a previously unconnected gate**
- **In sequential nets, the output feedback back into the net via an earlier input.**
  - This produces timed behavior due to the cyclic processing of the same material
  - Timing is based upon the delays introduced by the processing in each gate.



Stability of Network

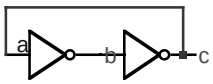
- **What would happen in the below inverter network if output d would be 1?**



1. **d is connected to a with a feedback loop so a=1**
2. **After the inverter, b=0 (this is one gate delay later)**
3. **one more gate delay later, c = 1**
4. **Finally, d = 0. But that's not possible!!! Yes is it is.**
   - – It simply means the network will oscillate between 1 and 0 with a period of 3 gate delays
   - ***This is therefore called an unstable network.***

Stability of Network (2)

- **What would happen in the below inverter network if output c would be 1?**

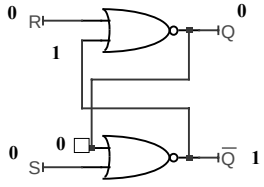


1. **c is connected to a with a feedback loop so a=1**
2. **After the inverter, b=0 (this is one gate delay later)**
3. **one more gate delay later, c = 1**
   - — So the network will maintain a stable state of 1
   - — Memory functions like this!
   - ***This is therefore called a stable network.***

Note that when turned on, its state is random!

The SR Latch

- **When turned on, the state of a stable network is randomly established**
  - To be useful, we need to be able to set it.
  - Such a device is called an SR Latch



It has a stable state when

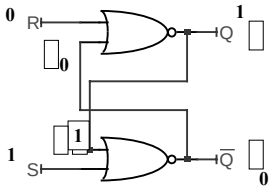
SR = 00

$Q\overline{Q}$ = 01

The SR Latch

- **Let's change S to 1**
- **T(g) is gate delay time**

Time	S	R	Q	Q	Stability
Initial	0	0	0	1	Stable
0	1	0	0	1	Unstable
T(g)	1	0	0	0	Unstable
2T(g)	1	0	1	0	Stable

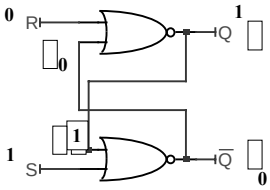


Change S back to 0

Stability maintained
Nothing happens because the effect of S depends on Q
The switch is already on!!

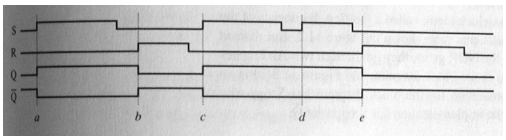
The SR Latch

- **The SR Latch acts like a switch.**
  - Bumping is turns it on: only works when not on.
  - Bumping R turns it off: only works when not off.



Timing Diagrams

- **Timing Diagrams will help you examine the state of a sequential network more easily than a table**
  - Although not necessarily with gate-delay accuracy

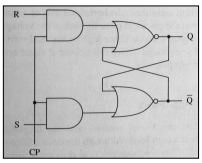


Gross Timing Diagram for turning the latch on (S) and off (R)

Clocked Flip-Flop

- **In a computer, each device is like a latch, with one of two states over time.**
  - All these devices need to be controlled and synchronized:
  - E.g., S and R should not be 1 simultaneously
  - This is done using a clock, which generates a sequence of pulses: on, off, on, off (the mhz thing)
- **Every sequential device has a clock (CP) input**
  - In addition to other inputs
  - It will respond only to input when the clock is high.
- **An SR Latch with a CP is called a Flip-Flop**
  - SR acts as an enable gate.
  - When CP is low, the enable gates are closed, and the switch cannot be turn on or off.

Clocked Flip-Flop (2)



- **This allows one to set up S and R for the desired transition while CP is low**
  - wait for the next clock pulse
  - and set the latch.
- **A clock makes time digital like voltage**
  - No mistakes due to timing glitches when setting devices.

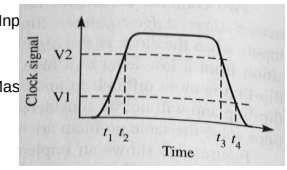
Master-Slave SR Flip-Flop

- **A clocked flip-flop is level-sensitive because the latch responds to CP only when clock is high**
  - However, it can still be unstable when output is fed back into SR via a combinational network
  - The device needs to be sensitive only for a very brief time, so feedback cannot affect its state.
- **To do this, there are two techniques**
  - Edge-triggered flip-flops are only sensitive when clock makes transition from low to high
  - Master-slave SR flip-flops the output is copied to a slave and used for feedback.
  - At no time can feedback run through master and slave.

Master-Slave SR Flip-Flop (2)

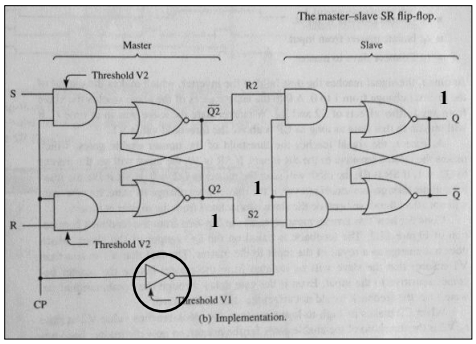
- **The idea is to first set the input flip-flop**
  - when the clock is high,
  - then copy this to an output flip-flop
  - when the clock goes low.

  - t2: Connect Master to Inp
    - Set Master value
  - t3: Disconnect Input
  - t4: Connect Slave to Mas
    - Set Output value
  - t1: Disconnect Slave

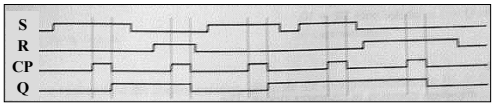


Master-Slave SR Flip-Flop (3)

1



Master-Slave SR Flip-Flop (3)



Like SR but set after clock = low!

Characteristic Table

- Truth Table
- give initial state
- specifies T+1

S(t)	R(t)	Q(t)	Q(T+1)	Result
0	0	0	0	No Change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	-	ND
1	1	1	-	

The Basic Flip-Flops

- **There are four common flip-flops:**

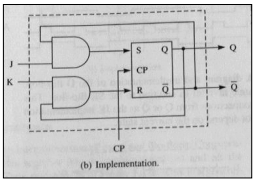
1. SR Flip-Flop
2. JK Flip-Flop
3. D Flip-Flop
4. T Flip-Flop

- **Can all be constructed from the SR Flip-Flop**

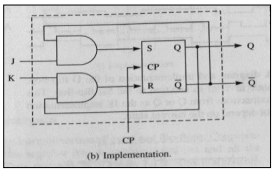
JK Flip-Flop

- **Uses undefined transition SR = 11 as toggle**
  - If Q =1 then Q=0 and vice versa

J(t)	K(t)	Q(t)	Q(T+1)	Result
0	0	0	0	No Change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	



JK Flip-Flop

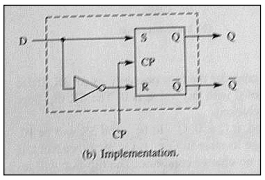


- **When JK = 10, R must be 0**
  - If Q is 0 Qbar will be 1 SR = 10 as well -> SET
  - If Q is 1 Qbar will be 0 SR = 00 no change -> SET
- **Similarly when JK = 01, final state Q = 0 like should**
- **If JK = 11 the values of Q and Qbar are used as input: R=Q and S=Qbar**
  - This will always yield a toggle

D Flip-Flop

- **Data flip-flop has only one input besides clock**
  - Q copies D with one clock cycle delay
  - So it stores the data until the next clock pulse

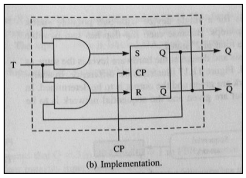
D(t)	Q(t)	Q(T+1)	Result
0	0	0	
0	1	0	Delay
1	0	1	Delay
1	1	1	



T Flip-Flop

- **Toggle flip-flop has only one input besides clock**
  - Like a JK but with only one input, connected to JK
  - So when T = 1 JK=11 and the flip-flop toggles
  - If T = 0 it does not change state

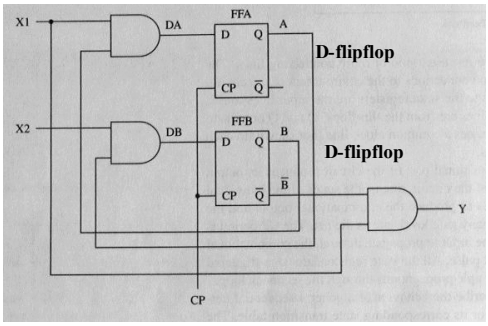
T(t)	Q(t)	Q(T+1)	Result
0	0	0	No Change
0	1	1	
1	0	1	Toggle
1	1	0	



Sequential Analysis

- **In sequential analysis, like combinational analysis, the aim is to determine the output of a given net.**
- **For this, two tools are used:**
  - State transition tables
  - State transition diagrams

Example



Two possible states

- **Because there are two flipflops, 4 possible states:**
  - Q of FFA is A, Q of FFB is B
  - AB= 00 01 10 11
- **Two inputs X1X2, 4 combinations:**
  - X1X2 = 00 01 10 11
- **Given the current state AB and current input X1X2**
  - What is current output?
  - What is output at next clock pulse?

  In D flipflops, Q(t+1) = D(t)

  A(t+1) = X1•B(t)

  B(t+1) = X2•A(t)

  Y(t) = X1•Bbar(t)

State Transition Table

Present state A	Present state B	Next state, present output (AB, Y) Input X1 X2 = 00	01	10	11
0	0	00, 0	00, 0	00, 1	00, 1
0	1	00, 0	00, 0	10, 0	10, 0
1	0	00, 0	01, 0	00, 1	01, 1
1	1	00, 0	01, 0	10, 0	11, 0

AB = 10 and X1X2=01

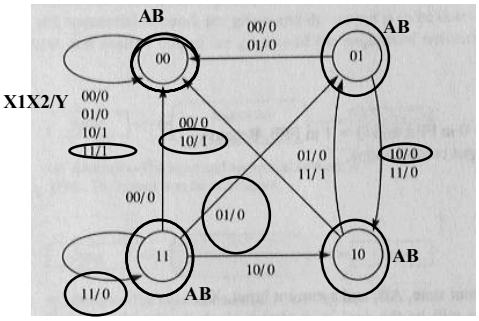
X1=0 and B(t) is 0 => A(t+1) = X1•B(t) = 0

X2=1 and A(t) is 1 => B(t+1) = X2•A(t) = 1

Next state is AB=01

Current output Y=X1•Bbar(t) = 0

State Transition Diagram



Excitation Tables

- **Characteristic tables are good for analysis**
  - Give next state given state and input
- **Excitation tables are good for design:**
  - Give input required to produce state and next state

Excitation Table for SR flip-flop

Q(t)	Q(t+1)	S(t)	R(t)
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

Excitation Table for JK flip-flop

Q(t)	Q(t+1)	J(t)	K(t)
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Excitation Table for D flip-flop

Q(t)	Q(t+1)	D(t)
0	0	0
0	1	1
1	0	0
1	1	1

Excitation Table for T flip-flop

Q(t)	Q(t+1)	T(t)
0	0	0
0	1	1
1	0	1
1	1	0

Sequential Design

- **Here the aim is to obtain a network given a state transition diagram or table.**
  - This can be a complicated procedure and is beyond the scope of this course.
- **The idea is:**
  - From the state diagram, make a transition table
    - For each combination of present state and input
    - List next state and present output
  - Next, transfer the entries for each flip-flop input to a Karnaugh map
    - Design the combinational part of the sequential circuit by minimizing the Karnaugh map
  - Draw the minimized combinational circuit that provides the input for the flip-flops

Final Exam

- **JHA Clergy St.**
- **Dec 19th**
- **0900**

- **Only scrap paper allowed!!**
- **PEP/6 instruction set provided**

Review

1) Introduction and Overview
- kinds of computers: analogue, digital, stored program, von Neumann machines
- history of computers: Babbage, Ada Lovelace, von Neumann, Turing,

2) Representation of Data (Chap. 3 of Warford)
- internal representation: character codes, ASCII, binary
- number systems: integer representation, unsigned binary, signed-magnitude, one's complement, two's complement
- operations in binary: carry bit, overflow bit, values of NZVC, range of two's complement
- hexadecimal representation

3) Computer Architecture (Chap. 4 of Warford)
- computer organization: machine components, main memory, secondary storage, tapes, disks, I/O devices, terminals
- the CPU: arithmetic logic unit, arithmetic logic operations, general purpose registers, control unit, special purpose registers, memory interface, MAR, MDR
- Pep/6 computer: hardware, CPU, registers, main memory, I/O devices
- Pep/6 instructions: instruction format, opcodes, addressing modes, unary operations, non-unary operations, direct addressing
- Input-Output: CHARI CHARO instructions, execution cycle
- Pep/6 operating system: RAM and ROM, application program memory, operating system memory

4) Assembly Language (Chapter 5 of Warford)

- assembly language: format, labels, opcodes, mnemonic names, operands, modes, comments
- assembly programs: format, object code, the Pep/6 assembler and simulator
- pseudo instructions: .word, .byte, .block, .ascii, .addrss, .equate
- unimplemented opcodes (used in DECI, DECO, HEXO)
- program development: design, code in HLL , translate, optimize
- translation to assembly: variables and types, assignment, arithmetic

Assembly Language (Chapter 6 of Warford)

- translation to assembly: branching structures, if statements, comparisons, loops, for statements
- arrays, indexed addressing
- stacks, stack special instructions, implementing procedure call,
- passing parameters: using the stack, stack relative addressing
- data types and structures

6) Instruction Sets (Lecture material)

- classification of instruction sets by instruction type, CISC vs. RISC
- classification by number of operands (0, 1, 2, 3 operand machines, load-store machines)

7) Device Management (Lecture material, Chapter 8 of Warford)

- events, I/O events, handling events
- device interface: device data register, device control register, detection of events
- communication paths: bus structure, controllers
- port mapped I/O, memory mapped I/O, polling
- exceptions: traps (synchronous exceptions) and interrupts (asynchronous exceptions)
- exception handling: interrupt vectors, RTI instruction

8) Memory Management (Lecture Material, plus Chapter 9 of Warford)

- storage allocation strategies: use of secondary storage
- swapping, uniprogramming, fixed-memory partitioning, variable memory partitioning, segmentation, paging, demand paging, with virtual memory
- characteristics of memory management methods: allocation, protection, hierarchy, relocation

9) Combinational Networks (Chapter 10 of Warford)

- specifying combinational networks: truth tables, Boolean expressions, logic diagrams
- Boolean algebra, values and operations: AND, OR, NOT
- Fundamental properties: commutativity, associativity, distribution, identity laws, complement laws, duality
- theorems of Boolean algebra: idempotence, absorption, de Morgan's, consistency
- logic gates: And, Or, Not, Nand, Nor, Xor gates.
- constructing logic diagrams for combinational networks
- converting between turth tables and Boolean expression: two level OR of AND circuits and AND of OR citcuits
- implementation using NAND gates
- minimization using Karnaugh maps in 3 and 4 variables
- combinational devices: enable and select lines, multiplexers, binary decoders, demultiplexer, one bit half adder, 1 bit full adder, four bit ripple adder

10) Sequential Networks (Chapter 11 of Warford)

- Latches and clocked flip flops
- Other basic flip-flops
- Sequential Analysis
- Characteristics Tables and Excitation Tables
- Examples of sequential networks

11) Additional material

- In addition to the above list, you should be able, to understand enough of Java programming language to be able to say what a simple program in Java does. If you understand the examples included in the lecture notes, then you do have enough knowledge of Java.

Exam

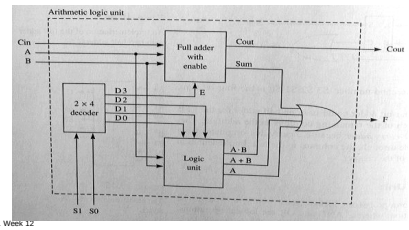
- **The exam will be structured in the following way:**
- **Part A) Multiple Choice**
  - 8 questions each worth 3 marks
  - Example:

What is the maximum number that can be represented by a 16 bit binary number?

a) 65536
b) 32768
c) 65535
d) 32767

Exam

- **Part B) Short Answers**
  - 4 questions each worth about 6 marks
  - *Example: What is an ALU? Give a brief description of its components*



Exam

- **Part C) Long Answers**
  - 2 questions each worth about 26 marks
  - *Example: Translate the following java program*
    Code will be given, you only need to translate functionality of system calls
    it is expected that you translate subroutines
    it is expected that your program works.
    Note: be sure array sizes are passed on!
  - *Example: Given this truth table, give minimal OR of AND boolean expressions for this function*

Questions?