SARAH'S 2GA3 MIDTERM CHEAT SHEET

Digital Logic Basics

Voltage V (volts)	The potential energy difference between two points in a circuit.
Current I (amperes)	Represents the flow of electrons along a wire. The number of charges that move through the wire per second.
Resistance R (ohms)	The measure of the opposition to current flow in an electrical circuit.

Ohm's Law: $R = \frac{V}{I}$

Series Circuits:

$V_T = V_1 + V_2 + V_3$

$I_T = I_1 = I_2 = I_3$

$R_T = R_1 + R_2 + R_3$

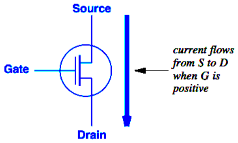
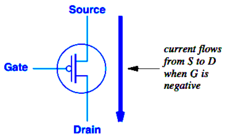
Parallel Circuits:

$V_T = V_1 = V_2 = V_3$

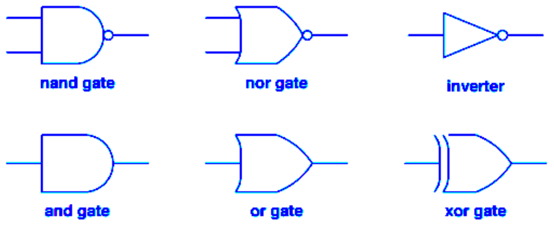
$I_T = I_1 + I_2 + I_3$

$\frac{1}{R_T} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}$

Transistors (MOSFET): The most widely used Transistor in modern electronics. It is made from semiconductive materials i.e. silicon, germanium. The control circuit is connected from the source S the drain D. The gate G controls whether S and D are connected.

G	**NMOS**	**PMOS**
	 Source Gate Drain current flows from S to D when G is positive	 Source Gate Drain current flows from S to D when G is negative
0	On/Connected	Off/Disconnected
1	Off/Disconnected	On/Connected

Logic Gates: Built from transistors.

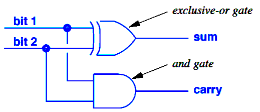


Note: and, or and xor actually have two inputs, and the inverter is also known as the ***not*** *gate.*

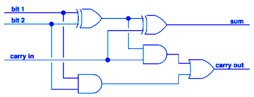
Understanding Logic Gates Built from Transistors: **1.** Separate the control circuit(s) (each one connected to G) and the main circuit from S to D, **2.** Cross out closed circuits, and determine the output.

Logic Circuits: Built from logic gates.

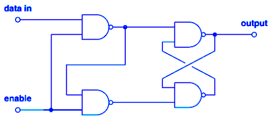
Half Adder:



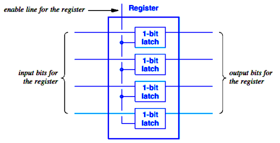
Full Adder: accepts a carry input



Latch: saves one binary digit. Enable line 1 = write, 0 = read,

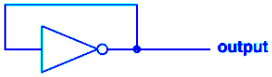


Register: a group of latches that can read and write a single n-bit number (n is the width).



enable line for the register
Register
1-bit latch
1-bit latch
1-bit latch
1-bit latch
input bits for the register
output bits for the register

Propagation Delay: The rate at which a transistor switches, Typical $\Delta T \approx 100ps$. To maintain synchronization, we rely on an oscillator (e.g. a quartz crystal) to produce a periodically repeating signal and initiate the next computation. We measure the **Clock Frequency** $f = \frac{1}{T}$. Most clocks operate between 100 MHz to several GHz, and complex circuits have multiple clocks.

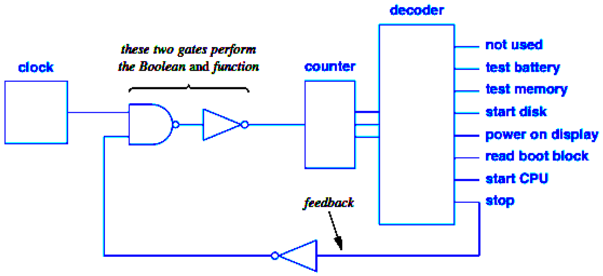


Flip Flop: Switches its output between 0 and 1 each time the input changes from $0 \rightarrow 1$ (rising edge) or $1 \rightarrow 0$ (falling edge).

Binary Counter: Each time the signal changes from $0 \rightarrow 1$, the counter outputs the count (as a binary number).

Decoder: maps a binary number to a set of 2^n outputs (for n-bit input). Often used for executing steps in sequence. (e.g. below)

Fixed Logic Circuit: continuous as long as $feedback$ is true



Data and Program Representation

Bit: 0 or 1 **Byte:** typically 8 bits, a k-bit byte can represent 2^k different values. **Word:** A collection of n bytes (n based on the architecture)

Memory: We can access data $memory[index] = word$. For storing the number 0x 04 03 02 01: **Little Endian:** least-significant byte first e.g. 01 02 03 04 **Big Endian:** most-significant byte first e.g. 04 03 02 01

Integer Representation

1. Binary Weighted Positional: default **2. Sign-Magnitude:** represent negative numbers! **3. One's Complement:** make arithmetic easier! **4. Two's Complement:** only one zero!

Method	Form	Max	Min	Casting
Bin. Pos.	$[m]$	2^k-1	0	$m \rightarrow 0 \ldots 0\ m$
Sign-Magn.	$[s \mid m]$	2^{k-1}-1	$-(2^{k-1}$-1$)$	$s\ m \rightarrow s\ 0\ 0\ 0 \ldots 0\ m$
1's Comp.	$[s \mid m]$ $\begin{cases} +s, & m \\ -s, & !m \end{cases}$	2^{k-1}-1	$-(2^{k-1}$-1$)$	$s\ m \rightarrow s\ s\ s \ldots s\ m$
2's Comp.	$[s \mid m]$ $\begin{cases} +s, & m \\ -s, & !(m-1) \end{cases}$	2^{k-1}-1	$-(2^{k-1})$	$s\ m \rightarrow s\ 0\ 0\ 0 \ldots 0\ m$

IEE 754 Float Representation

$$[s \mid e+b \mid m] \begin{cases} s = 0, \ 1.m \times 2^e \\ s = 1, \ -(1.m \times 2^e) \end{cases}$$

Max/Min: $\pm 1.m_{max} * 2^{e_{max}}$ **Min Increment:** $\pm 1.0 * 2^{e_{min}}$

Special Value	e	m
0	0	0
$\pm\infty$	all 1s	0
NaN	all 1s	$\neq 0$

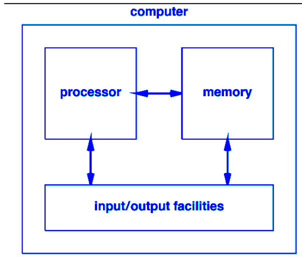
# of bits	s	e	m
Half float 16-bit	1	5	10
Float 32-bit	1	8	23
Double 64-bit	1	11	52

Decimal to Binary Digit: **1.** take the fractional part of the decimal **2.** multiply by 2 **3.** take the leading whole number as the next digit i.e. 0 or 1 **4.** repeat!

Decimal to IEEE Float Representation: **1.** Determine the number of bits for s,m and e **2.** Convert to Binary Digit $\times 2^0$ **3.** Normalize $\rightarrow 1.m^e$ **4.** Add b if $e > 0$ **5.** Store as $[s \mid e+b \mid m]$

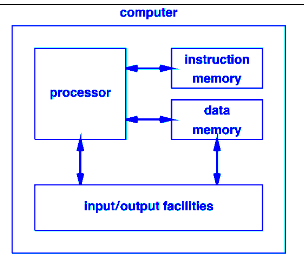
Processors and Instructions

Von Neumann Architecture



Advantages: Flexibility (most popular in everyday computers!) **Disadvantages:** Memory Bottleneck, Von Neumann Bottleneck (Only one interface between the processor and memory)

Harvard Architecture



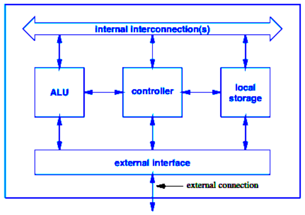
Advantages: Less susceptible to memory bottleneck, Security advantage (you can't overwrite instructions) **Disadvantages:** Rigid split (not flexible)

Processor Types:

Fixed Logic	Function fixed in hardware
Selectable Logic	Choose one of several fixed funcitons
Parametrizable Logic	Parameters govern computation
Programable Logic	List of instructions provided at runtime (relys on programming)

Processor Categories:

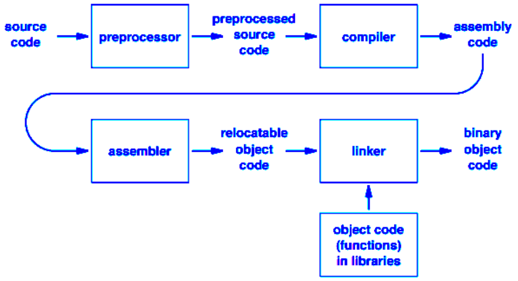
Co-processors	Dedicated function	Fixed/Selectable Logic	Floating point engine
Micro-controllers	Direct hardware control	Programmable Logic	Elevator doors, opening pipes
Embedded Systems	Real-time OS, garunteed execution time, dedicated hardware	Programmable Logic	Coffe machine
General Purpose	Interchangeable and compatible for multiple systems	Programmable Logic	Computer



Structure of Processor (CPU): Arithemtic Logic Unit (ALU): One dedicated unit containing hardware for addition, subtraction and logic operations. Only performs one operation at a time and relies on a controller to specify which operation it will perform. **Local Storage:** Registers **Controller:** Decides what happens and where the data flows. It steps through the program and coordinates the actions of all other hardware units to perform the specified operations. **External Interface:** handles all communication between the processor and the rest of the computer systems i.e. memory and IO **Internal Connections:** one or more hardware mechanisms that are used to transfer values between the other hardware units e.g. BUS and control lines

Fetch-Execute Cycle: The OS constantly runs:

```
ip = start of program
Repeat forever
 instuction = fetch (memory [ip])
 execute instruction
 ip++
```



Program Translation: Source Code → Assembly Code → Relocatable Object Code → Executable

Instruction Set Architecture (ISA):

Complex Instruction Set Computer (CISC)	Reduced Instruction Set Computer (RISK)
Used by X86 complex operations, variable number of clock cycles per operation, fewer instruction calls, high power consumption	Used by ARM simple operations, all take the same clock cycles, many instruction calls, low power consumption

RISK Pipeline: RISK creates a pipeline that executes one instruction per cycle.

Hazards:

Hazard Type	Description	Solutions
Data Hazard	waiting for data from earlier instruction	Data forwarding between stages, Rearrange computation
Control Hazard	incorrect instruction is in the pipeline (branching)	Conditional branch prediction (if prediction is incorrect, flush pipeline)
Structural Hazard	resource conflict	Load data in parallel e.g. using seperate banks

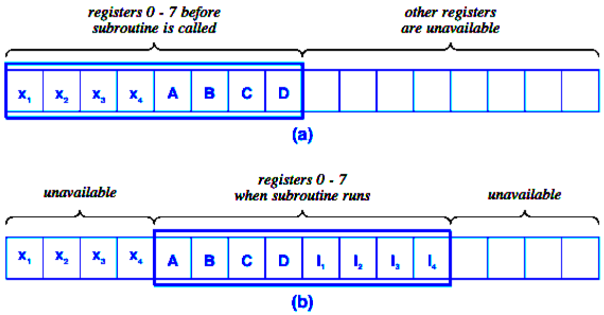
Pipeline Rules: At "fetch operands," only the operands must be available (not where we are storing results). Only when a jump is executed we flush the pipeline.

Example:

```
1: loop : add r1 r1 r2
2: add r5 r2 r3
3: add r4 r2 r3
4: add r5 r0 r3
5: sub r7 r1 r3
6: cmp r5 r4
7: bne loop
8: . . .
```

stp	stg1	stg2	stg3	stg4	stg5	done
0	1					
1	2	1				
2	3	2	1			
3	4	3	2	1		
4	5	4	3	2	1	
5	6	5	4	3	2	1
6	7	6	5	4	3	2
7	7	6	S	5	4	3
8	7	6	S	S	5	4
9	8	7	6	S	S	5
10	9	8	7	6	S	S
11	10	9	8	7	6	S
12	11	10	9	8	7	6
13	12	11	10	9	8	7
14	1	flush				
15	2	1				

Register Window: A processor optimization for argument passing. The register hardware only exposes a subset of registers that moves automatically each time a subroutine is invoked, and moves back when the subroutine returns.


registers 0 - 7 before subroutine is called
other registers are unavailable
x_1
x_2
x_3
x_4
A
B
C
D
(a)
unavailable
registers 0 - 7 when subroutine runs
unavailable
x_1
x_2
x_3
x_4
A
B
C
D
l_1
l_2
l_3
l_4
(b)

Instruction Representation and Execution

Instruction Encoding Types: Each instruction/operation is given a unique OP-CODE (identifier) in the ISA.

Instruction Types:

Fixed Number of Operands	Bit fields always have the same semantics
Variable Number of Operands	Use memory more efficiently

Operand Types:

Data Source	immediate value encoded in the instruction, a numbered register, a memory address
Data Destination	in a numbered register, in a spefic memory locaiton

Operand Encoding Types:

Implicit Encoding	Operand is assumed some fixed value
Explicit Encoding	Operand field contains all information necessary to interpret the operand

Encoding a Given Instruction: **1.** Review the ISA **2.** Separate the available n-bits into the operation code (OP-CODE) and its operands **3.** Enter the OP-CODE according to the ISA **4.** Enter the operands according to the ISA **5.** Add 0's to the unused portions (entering 1s would not make a difference since these slots are ignored, nevertheless, insert 0s by convention) **6.** Convert the entire bit-string to Hexadecimal

Operand Addressing: Different ways to address operands is order from fastest to slowest. **1.** Immediate value encoded in instruction **2.** Value in register, register in instruction **3.** Value in memory, memory in address in instruction **4.** Value in memory, memory address in register, register in instruction **5.** Value in memory, memory address in different memory location encoded in instruction