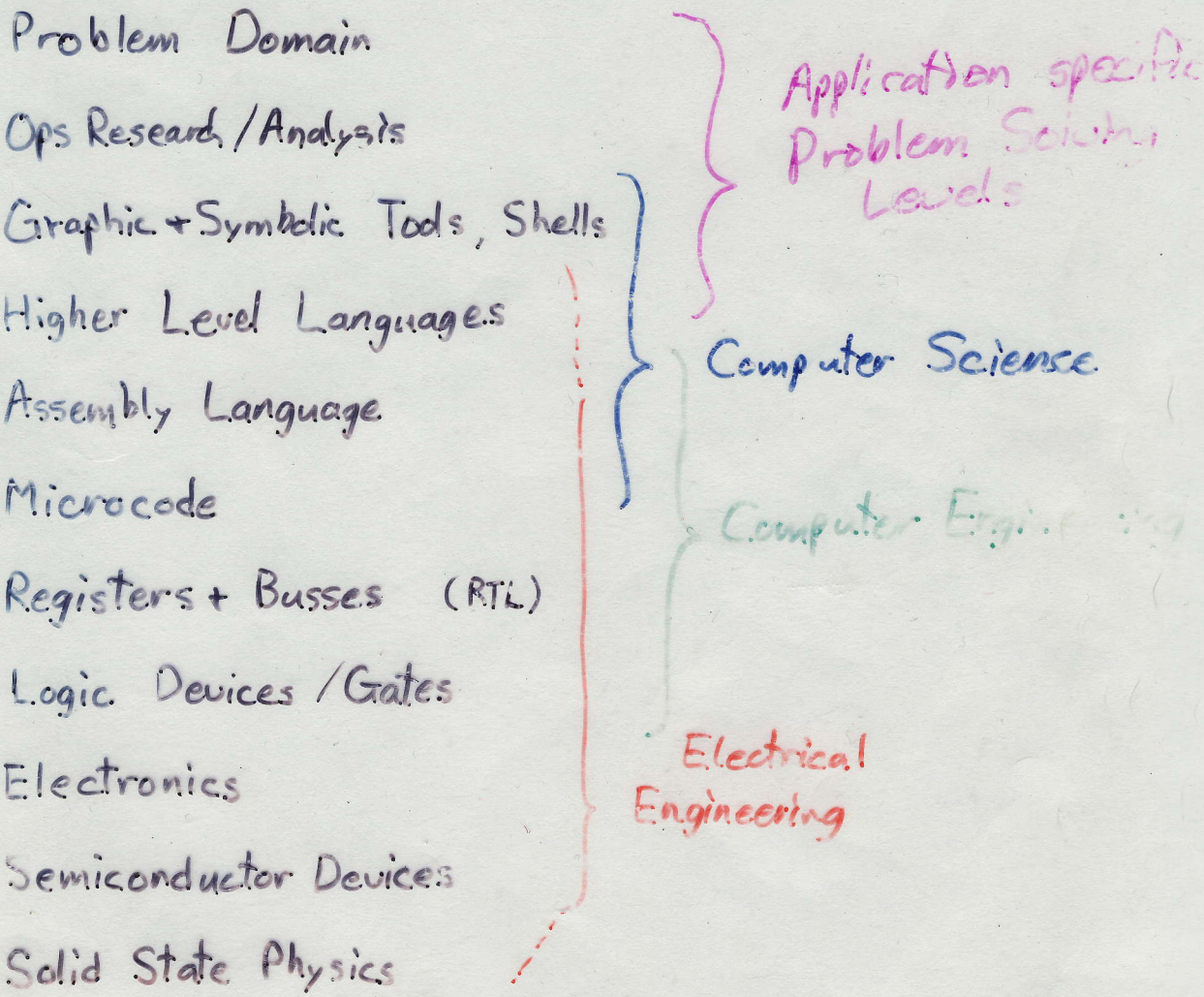
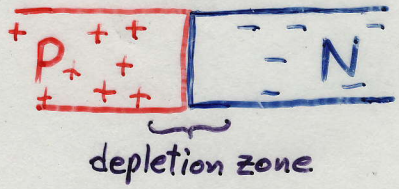
COMPUTER TECHNOLOGY HEIRARCHY

Computer technology has advanced by successively adding new abstractions, or more layers, to the technology heirarchy between problem and the existing tools for implementing a solution. Partitioning technology in levels like this allows each level to be worked without over involvement in details at lower levels. Each level furnishes a "model" to those above it

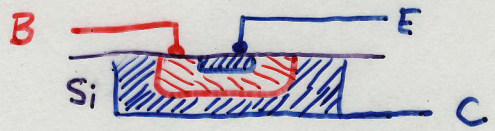


Examples - Technology Hierarchy

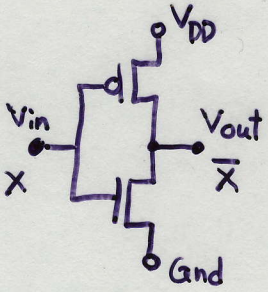
Solid State Physics :
The semiconductor junction

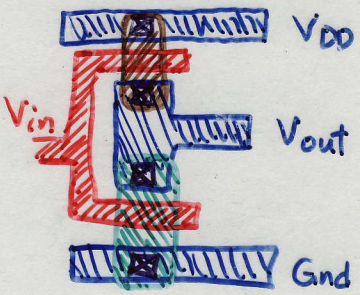


Semiconductor Devices :
The transistor

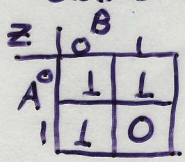


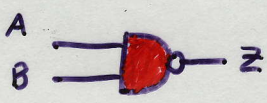
Electronics :
The CMOS inverter

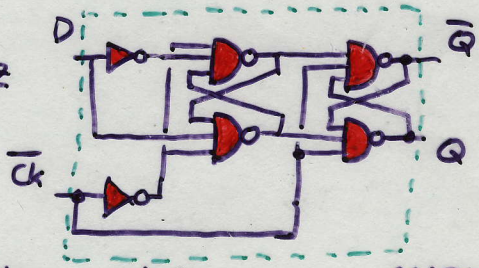


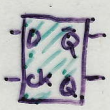


Logic :
Gates and latches

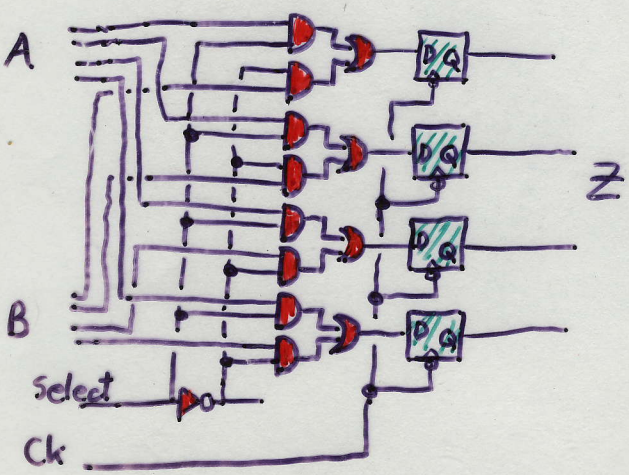


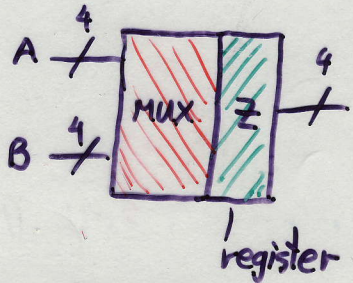






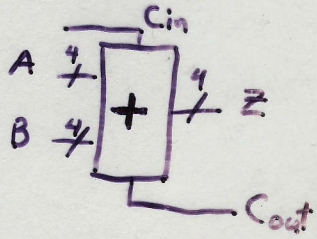
Registers : Assemblage of gates + latches (MSI level : Medium Scale Integration)





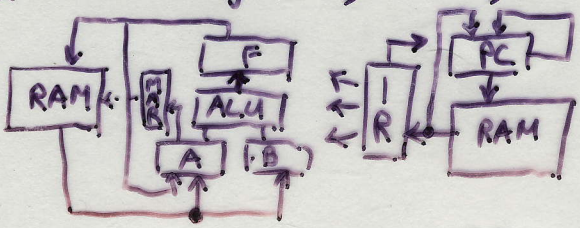
(7498)

Also consider :



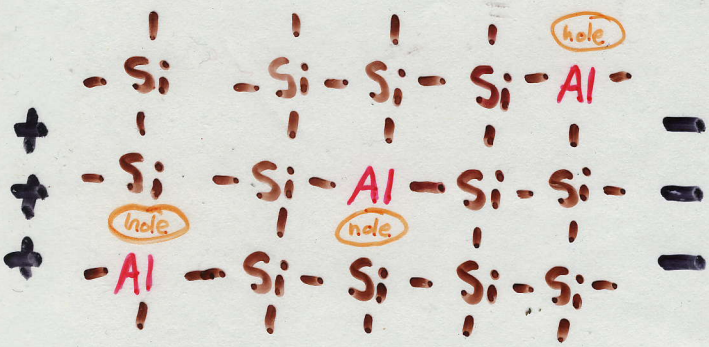
(7483)

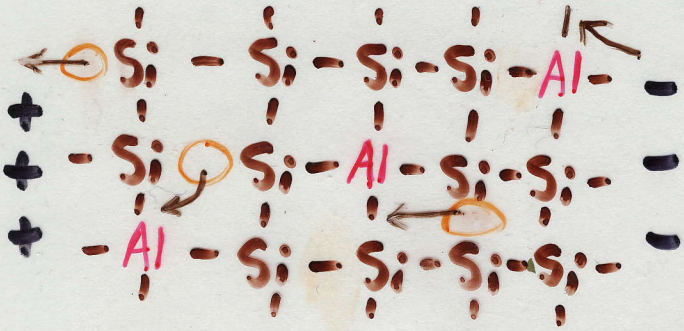
The CPU : Registers, Busses, Mux'es, ALU etc. to form a Processor

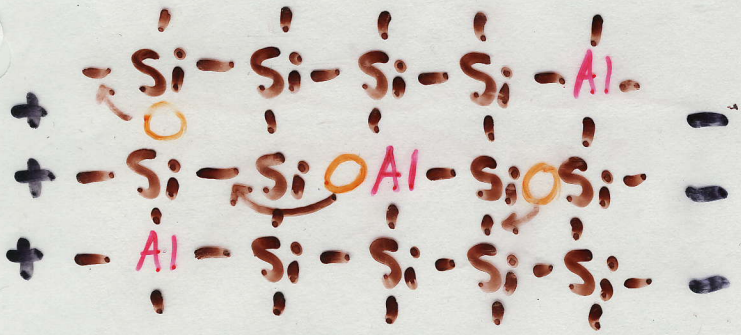


* This is where we are headed!

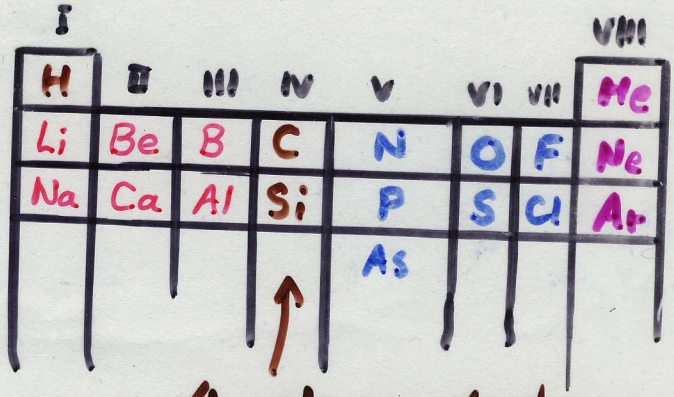
Current in PMOS devices







As electrons change positions, it appears that the "holes" flow from + to −

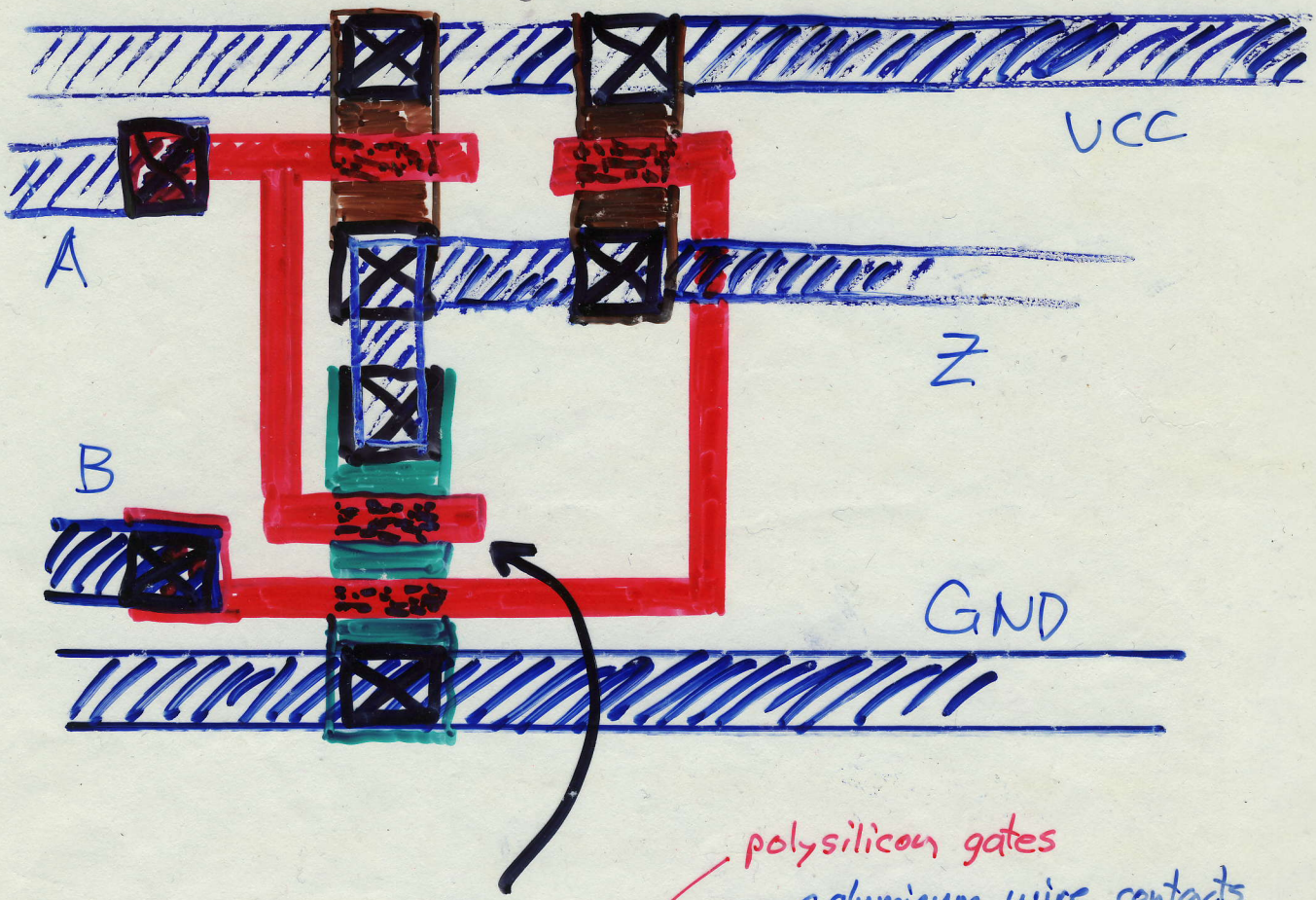


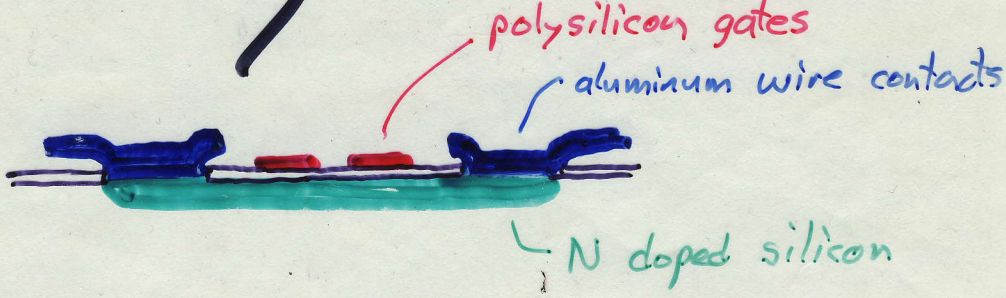
4 valence electrons — These semiconductors try to establish 4 bonds with adjacent atoms

Al, with 3 valence electrons, will fit into the Si lattice, but brings 1 too few electrons to the party.

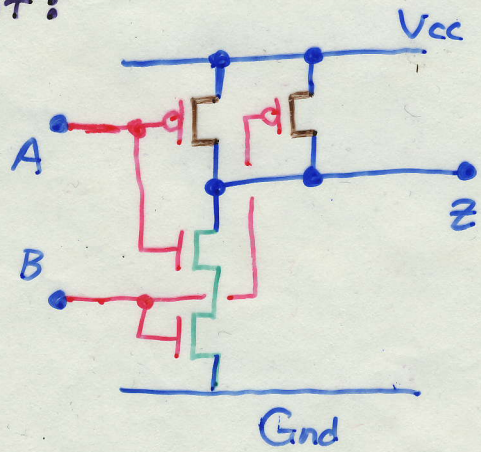
While P-doped Si is electrically neutral, The material craves more electrons — It has a surplus of "holes".

NAND gate in CMOS





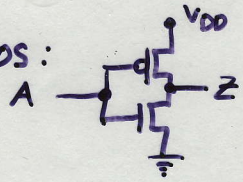
Circuit:

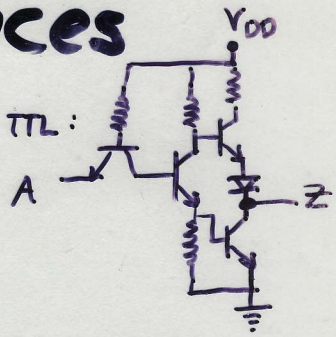


Logic Functions and Devices

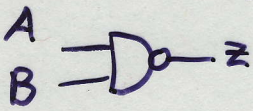
Inverter : 

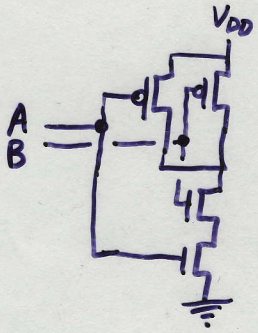
$Z = \bar{A}$

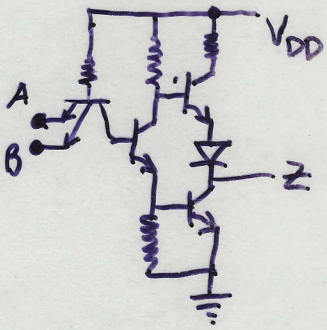
CMOS: 

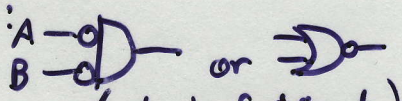
TTL: 

Nand : similar to inverter, but multiple inputs





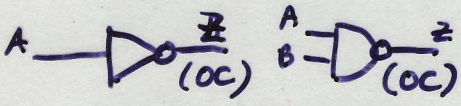


Nor : 

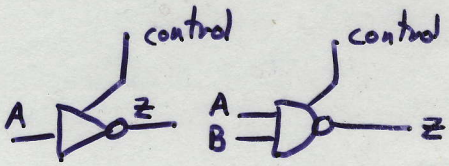
(dual of Nand)

And, Or : as Nand, Nor but with inverted output

Buffers

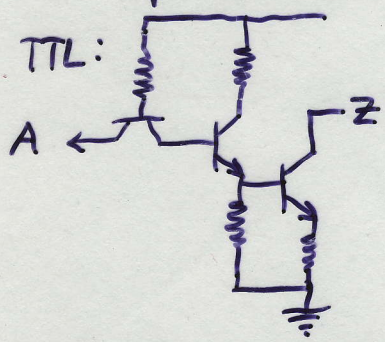


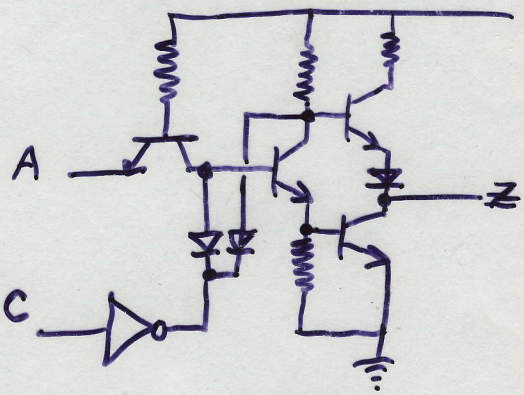
Open collector



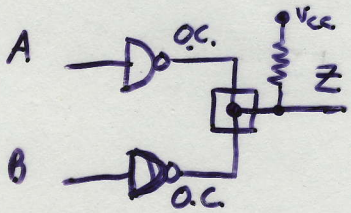
tri-state (simplified)

TTL:





Allows "wired-OR"

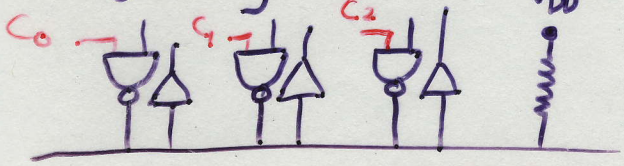


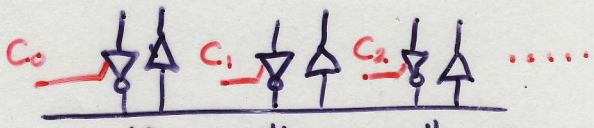
A \ B	0	1
0	1	0
1	0	0

A \ C	0	1
0	1	hiz
1	0	hiz

Allows Bussing :

Bussing :



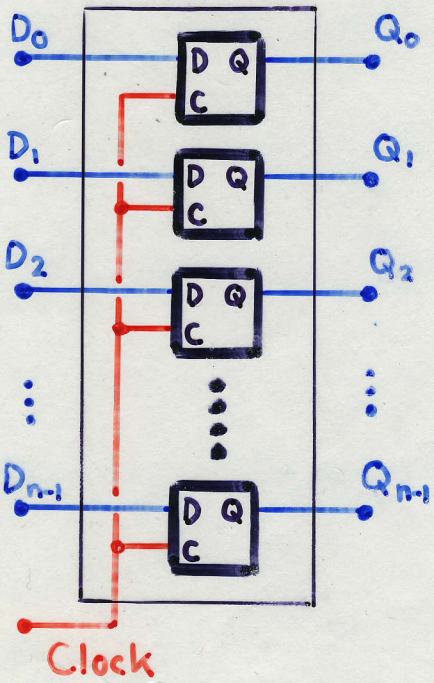


Has active pull-ups for faster switching

See Appendix A for more.

RTL (Register Transfer Level) Components

1. The register



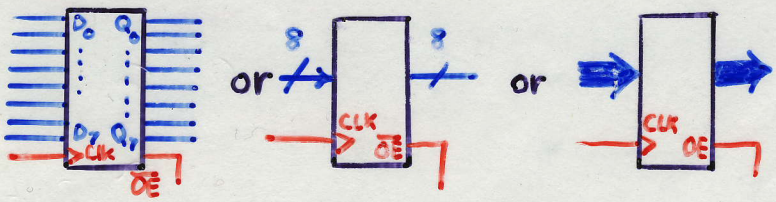
a) Is an array of flipflops with common CLK

b) Contents referred to as a value (e.g. multiple bits expressed in binary, hexadecimal, etc.)

c) Specified by: • n = number of bits
- Clock type (edge, level, etc.)
- Inverted clock
- Output control (eg. 3 state)
- Inverted Outputs
- Possibly MUX on input

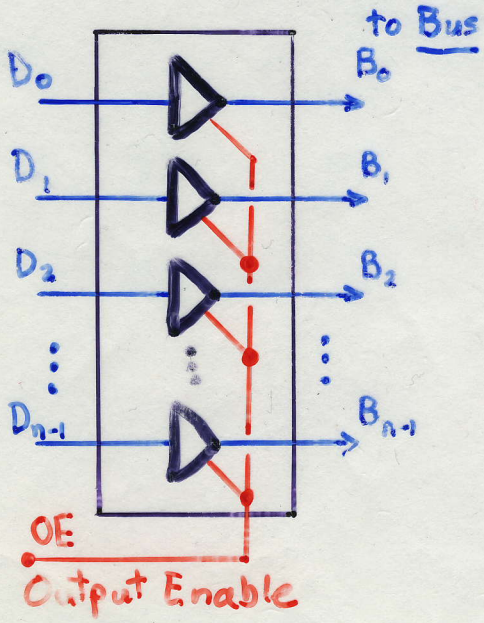
d) Examples:
7475 : 4 bit, "transparent" with complementary outputs
7498 : 4 bit "transparent" with 2 in MUX
74(?)373 : 8 bit "transparent" with OE (3 state)

Symbol used :



The register is our basic data storage device.

2. The Buffer



a) Is an array of binary buffer elements

b) Usually tri-state (or open collector)

c) Used to selectively switch a multi-bit value from some source onto a bus (Multiplexing function)

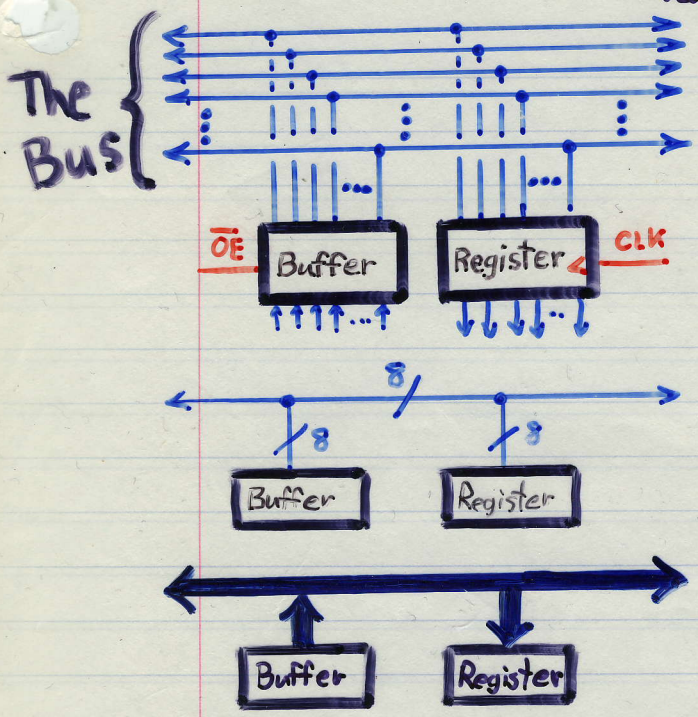
d) Specified by • number of bits n
- Inversion of data and/or enable
- Type of output - e.g. tri-state

e) Examples: 74125 4 bit, separate enables
74241 8 bit, 2 enables

RTL Components (continued) 4

3. The Bus

: An organized collection of signals (wires) used to convey data (values) from one place to another. Usually, there is more than one possible source, and/or more than one possible destination.



(The Bus is equivalent to a Mux, with output routed to various destinations.)

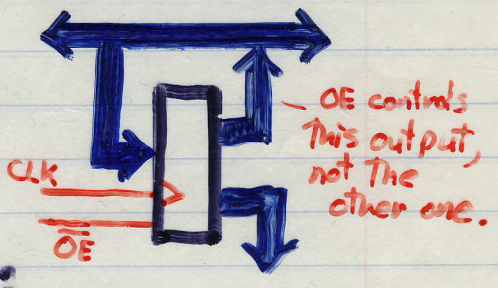
There may be many buffers on the bus, and many registers. Devices other than registers may take data from the bus. (eg. ALU's)

Bus rules:

1) Only one buffer (device on the bus) may be enabled at a time.

2) Do enables before clocking data into registers from the bus. (Get timing right)

A common device on busses is a register that can be written or read:

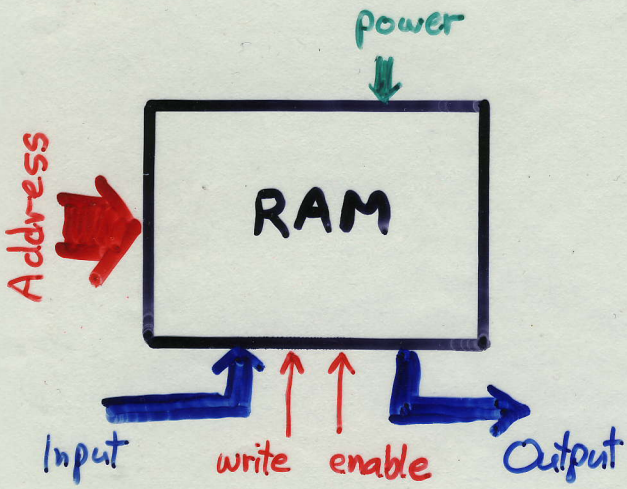


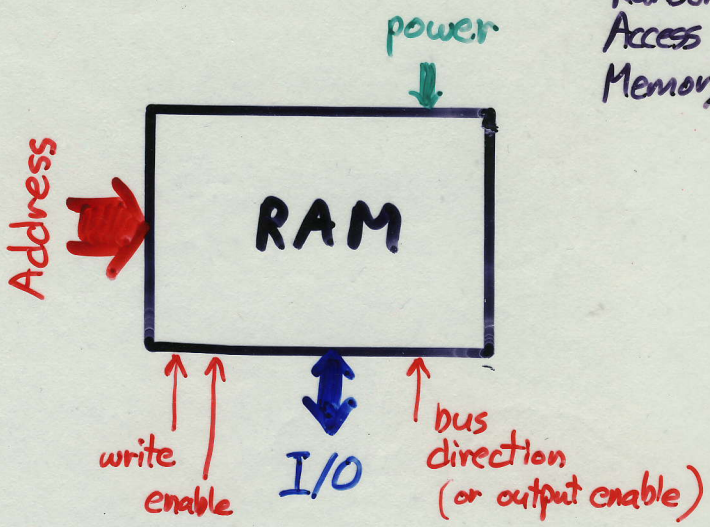
Other RTL Components:

4. Memory (RAM) (ROM): like an addressable collection of registers
5. ALU (Arithmetic, Logic Unit) performs operation on data
6. MUX's (Multiplexers) + DeMux's (Decoders)
7. Counters, Shift registers, etc.

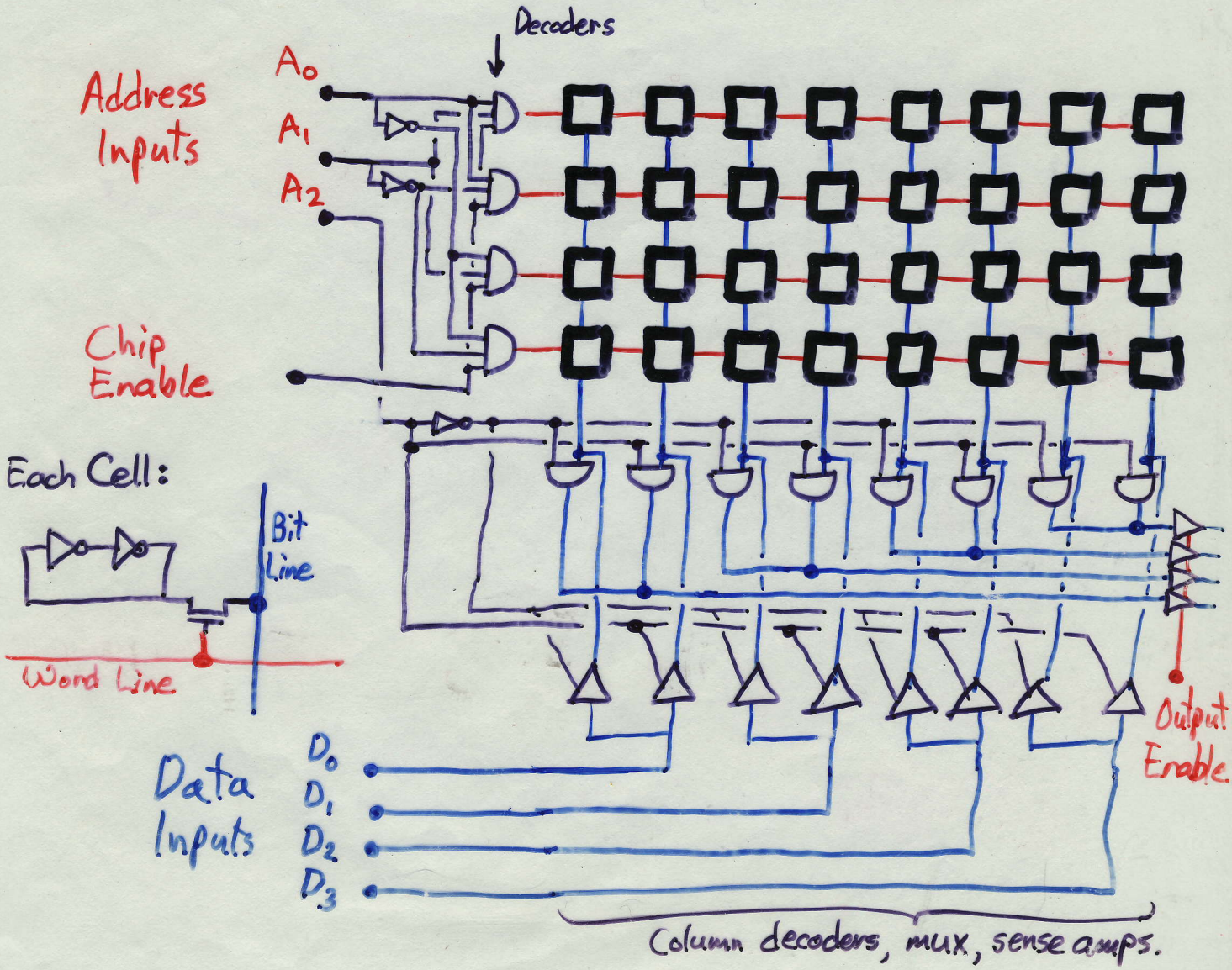
Put enough of these together correctly, with control circuitry and you get a CPU.

Memory Organization (RAM) = Random Access Memory



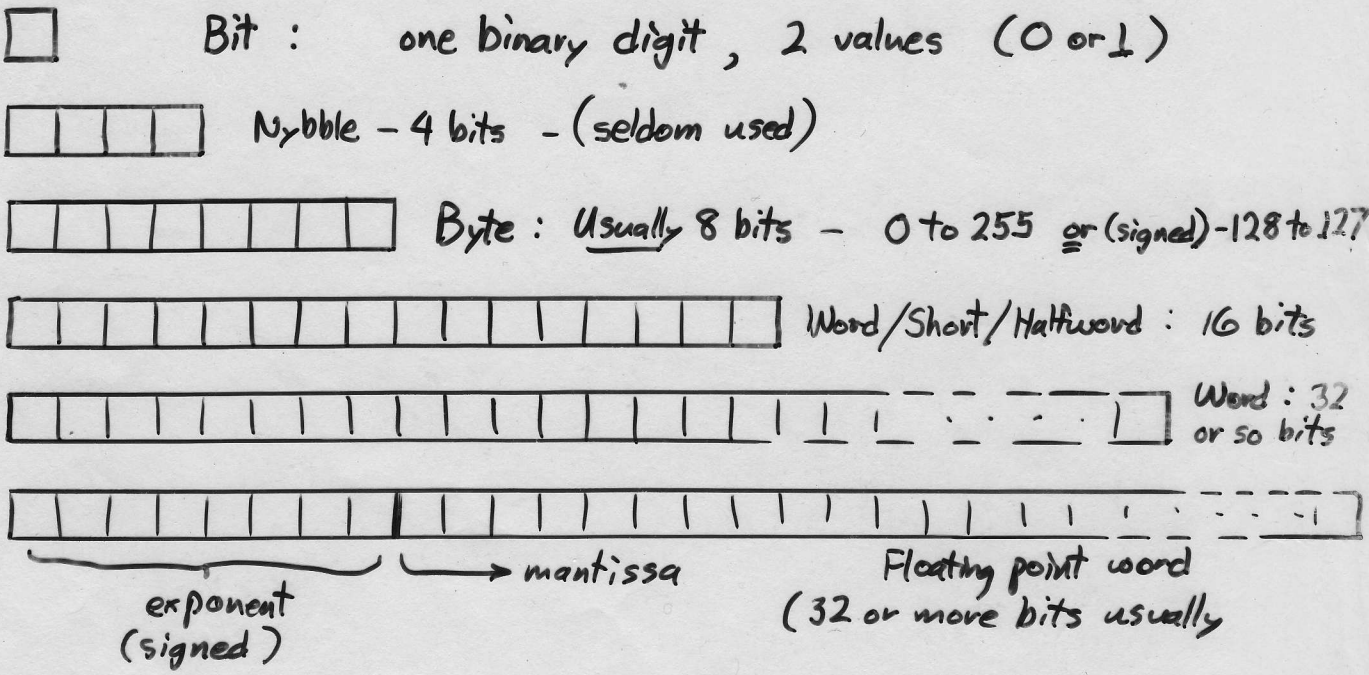


Inside The Box:



Encoding of Information

Size



Mapping binary arrays to positive integers

bit:	7	6	5	4	3	2	1	0
	0	1	0	0	1	1	0	1
	128	64	32	16	8	4	2	1

$= 1 \times 2^6 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 77$

$= 64 + 8 + 4 + 1 = 77$

Mapping signed binary arrays to negative integers

bit	S	6	5	4	3	2	1	0
	1	0	0	1	0	0	1	1

2 representations exist:

1's complement: Flip all bits and treat as positive except for sign

→ -1101100 = -108

note: 11111111 equals -0

2's complement: Flip all bits, Then add 1 and treat as positive except for sign

→ -1101100 → -1101101 = -109

(Most computers nowdays use 2's complement)

Floating Point: Many, Many different representations

Encoding of Information (cont.)

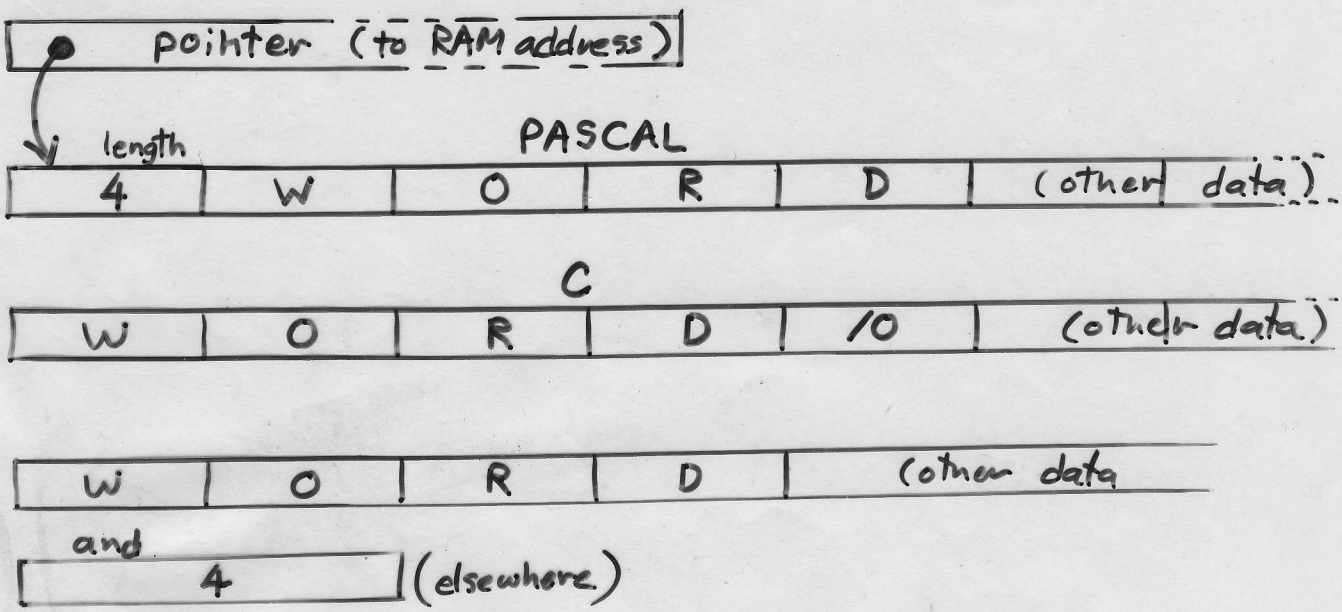
Character and String Data

(ASCII = American Standards Committee on Information Exchange) see Appendix D

7	6	5	4	3	2	1	0
x	1	0	0	0	0	0	1

→ 'A' ASCII character

Strings: (of arbitrary length)



Notation:

for the stored data:

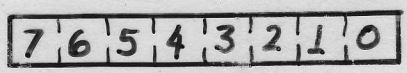
0	1	0	0	1	0	1	0	0	0	1	1	1	0	1	0

binary	0100101000111010	notation	b0100101000111010
octal	0 4 5 0 7 2	notation	o45072
hexadecimal	4 A 3 A	notation	0X4A3A
decimal	19002	notation	19002

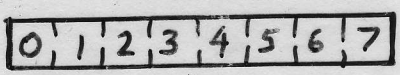
Note: Hexadecimal has come to be preferred for work at the machine/assembly level since digits align on byte boundaries for multi-byte words.

The Endian Wars

Bit order in a byte :



Little Endian



Big Endian

For bytes, virtually everyone uses The Little Endian scheme

Byte order (and memory address order) in a Word :

xx3 xx2 xx1 xx0 ← Address

Little Endian

MSB = most significant bit

LSB = least significant bit

xx0 xx1 xx2 xx3 ← Address

Big Endian

Little Endian : ~~PDP-11~~, ~~VAX~~, Intel + others ← more of these seemingly

Big Endian : Motorola + others

Suppose you print the bytes in order :

Little Endian 0x1234 (hexadecimal) : 3412

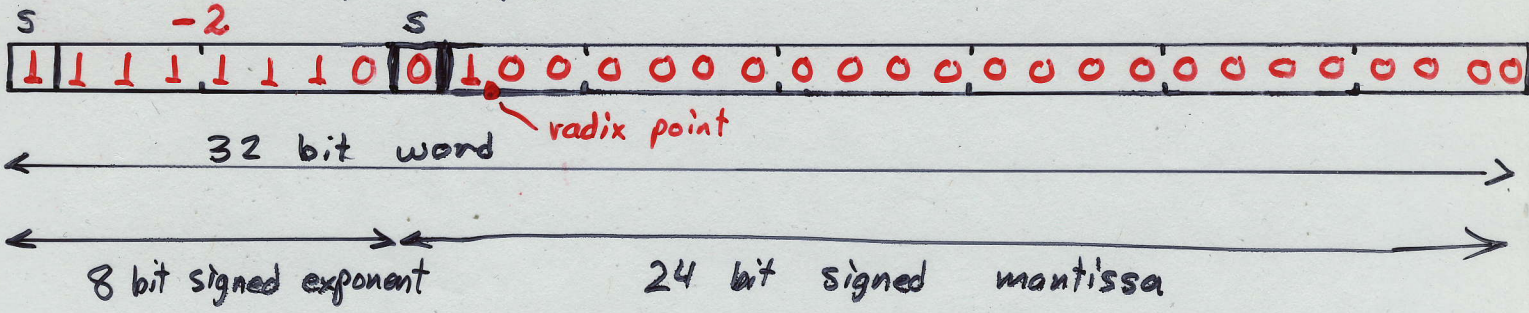
Big Endian 0x1234 : 1234

Theorists, number oriented applications seem to prefer "little endian" byte order.

Character oriented applications seem to prefer "big endian" byte order.

Floating Point

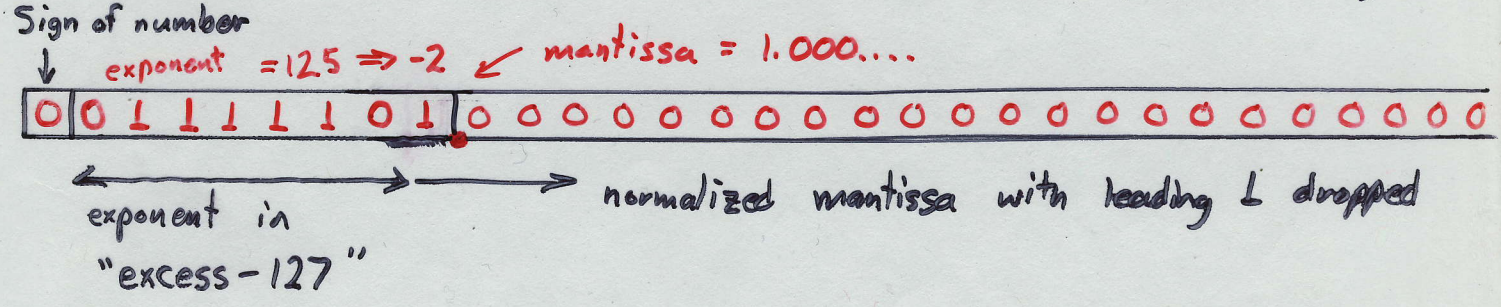
A "simple" approach: .25



Problems: 1) You cannot use a simple arithmetic compare to see which of two numbers is larger

2) For normalized numbers you always have a "1" in bit position 23. This is wasteful.

An alternative (IEEE 32 bit "single precision")



Now:

1) We can use a simple arithmetic (integer) subtract to compare two numbers

2) We have an extra bit of precision for the mantissa.

4) Numbers that can be represented: $\pm 1.M \times 2^{E-127}$

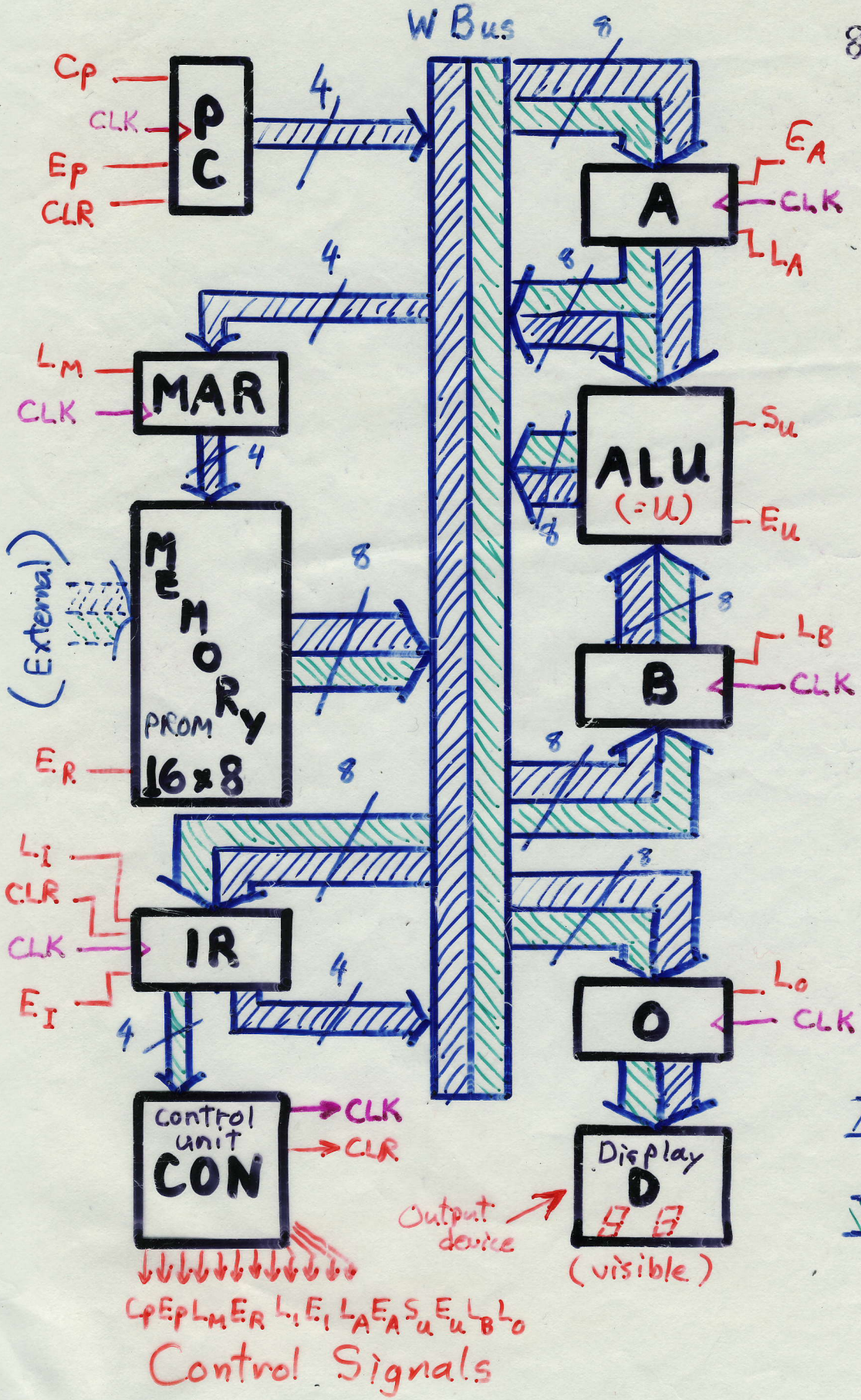
5) Special values for exponent:

0 = 00000000 ⇒ zero exactly

255 = 11111111 ⇒ infinity

See Fig 7.25 on p 305 for other, longer, formats.

SAP-1 Data Path (+)



8 bit machine

E_x = Enable output of register X

L_x = Load register X

CLR = Clear (set to ∅)

Cp = Count + (increment)

S_U = Select function for ALU

CLK = clock

The memory is Read Only (It is loaded by a process external to the processor. For example a front panel.

Least Sign. 4 bits

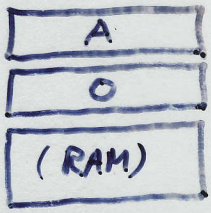
Most Sign. 4 bits

SAP-1 Instruction Set

R = register (memory address)

0 = Ø Ø Ø Ø	R	LDA	(R) → A
1 = Ø Ø Ø 1	R	ADD	(R) + A → A
2 = Ø Ø 1 Ø	R	SUB	A − (R) → A
E = 1 1 1 Ø		OUT	A → O
F = 1 1 1 1		HLT	Halt execution

Programming Model:



Sample Program:

label	mnemonic	operands
START	LDA	R9
	ADD	RA
	ADD	RB
	ADD	RC
	SUB	RD
	OUT	
	HLT	

Assembly Language

Ø	Ø 9
1	1 A
2	1 B
3	1 C
4	2 D
5	E X
6	F X

Machine Language

0	0	0	0	0	1	0	0	1
1	0	0	0	1	1	0	1	0
2	0	0	0	1	1	0	1	1
3	0	0	0	1	1	1	0	0
4	0	0	1	0	1	1	0	1
5	1	1	1	0	x	x	x	x
6	1	1	1	1	x	x	x	x
7	x	x	x	x	x	x	x	x
8	x	x	x	x	x	x	x	x
9	d	d	d	d	d	d	d	d
A	d	d	d	d	d	d	d	d
B	d	d	d	d	d	d	d	d
C	d	d	d	d	d	d	d	d
D	d	d	d	d	d	d	d	d
E	x	x	x	x	x	x	x	x
F	x	x	x	x	x	x	x	x

Memory Image

Alternate Assembly Language:

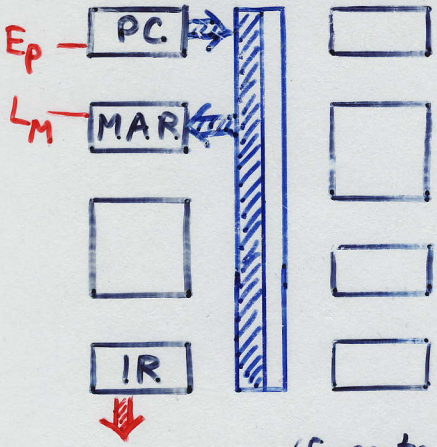
V	DC	
T	DC	
X	DC	
Y	DC	
Z	DC	
START	LDA	V
	ADD	T
	ADD	X
	ADD	Y
	SUB	Z
	OUT	
	HLT	

Somewhere the assembler would need to be instructed to allocate addresses for data starting at 9 and code starting at START to be compatible with the (manual) loader.

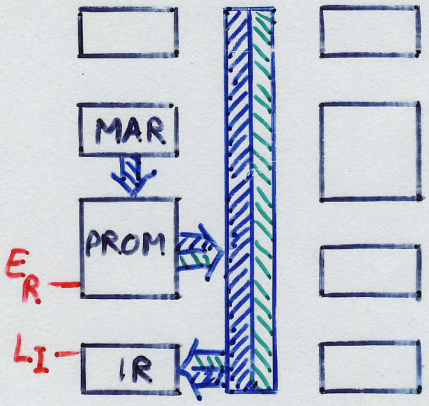
Usually, the loader decides exactly where code and data go, but that's a more advanced topic.

SAP-1 Instruction Cycle

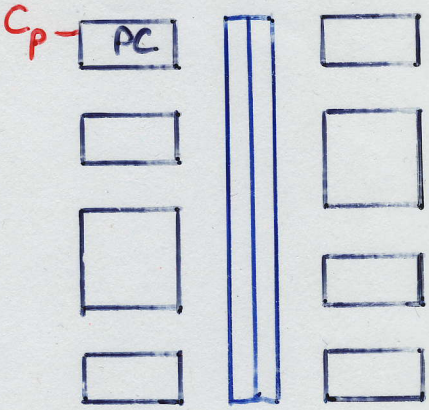
1 Fetch Instruction



2 Address Instruction

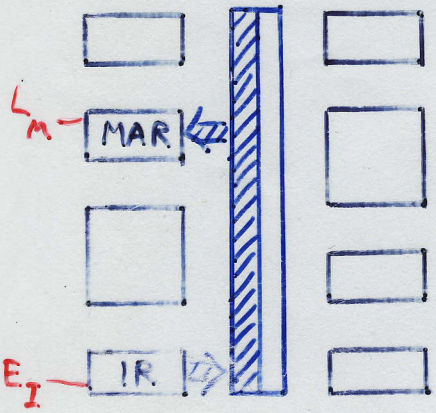


3 Increment PC

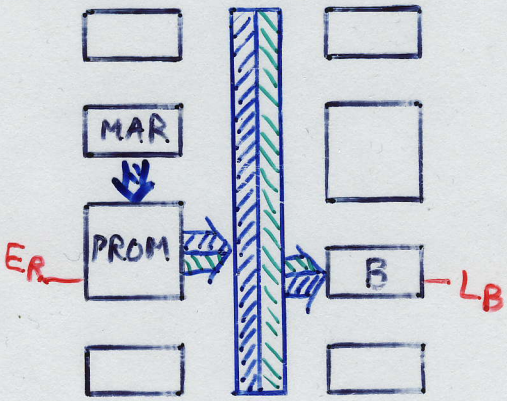


(Execute 1) (Execute 2) (Execute 3)

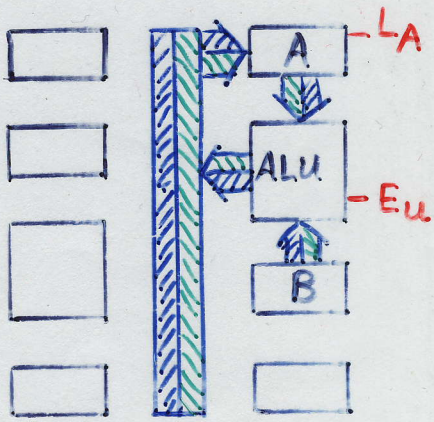
4 Fetch Data



5 Load B



6 Add



Control Signal Sequence :

Phase C_P E_P L_M E_R L_I E_I L_A E_A S_u E_u L_B L_O

	Addr	
1	0	1 1
2	1	1 1
3	2	1
4	3	1 1
5	4	1 1
6	5	1 Ø 1
⋮	⋮	

Notice : Register A must be master-slave, since it is referenced and loads during same clock !!

Simple Controller Implementation