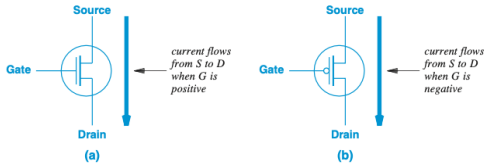
Logic Basics

Transistors

MOSFETs have 4 components: Source, Gate, Drain, and Base

PNP/NMOS: Is on when gate is positive. No circle.
NPN/PMOS: Is on when gate is negative. Has circle.

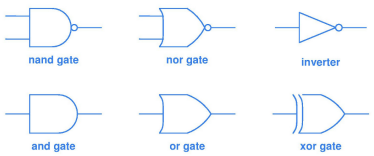
Generally, transistors are used to pull the output to either a positive voltage, or a zero voltage (1 or 0, on or off). If output is not pulled to one of these, the output is floating and is indeterminate in voltage.



Logic Circuits

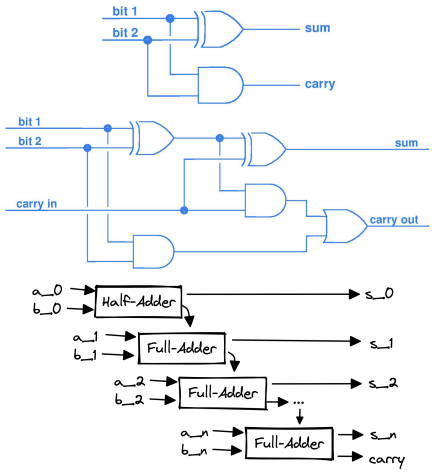
Symbols

p	q	and	nand	or	nor	xor	invert p
1	1	1	0	1	0	0	0
1	0	0	1	1	0	1	0
0	1	0	1	1	0	1	1
0	0	0	1	0	1	0	1



Adders

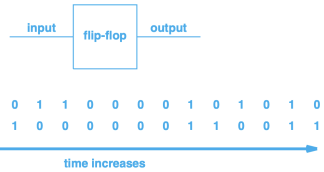
Left: Half-adder. Right: Full-adder.



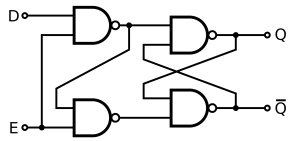
Unlike half-adders, full-adders can recieve carry-in bits. To add more bits, chain multiple full-adders together. If final carry at end is 1, there's an overflow error.

Latches & Flip-flops

Flip-flop Every time the input switches from 0 to 1, the output switches to the opposite.

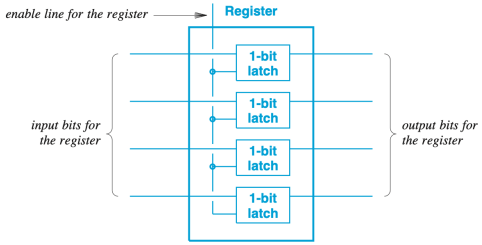


Gated D-latch



E/C	D	Q	$\overline{Q}$	Comment
0	X	Q_{prev}	$\overline{Q}_{prev}$	No change
1	0	0	1	Reset
1	1	1	0	Set

- Q - The stored bit
- D - Data/bit to write to Q
- E - The enabler (Must be 1 to enable writing, otherwise nothing changes)
- Operate on the principle of propagation delay.
- Stacking many of them can be used to create a register:



Counters

For each transition from **low to high**, the counter increments the binary output by 1. (Counting the transition from high to low does the same thing, but the lecture used rising-edge counters).

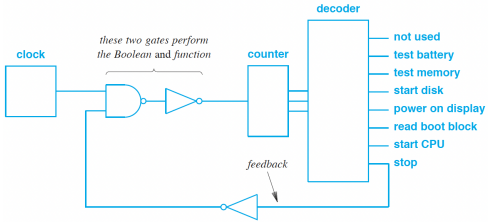
Propagation Delay

The rate at which a transistor switches. Typical 100 ps (picoseconds)

Decoders

Take in an n-bit number, and turn on one of 2^n outputs.

Feedback Loops



- Counter increments each time the clock changes from 0 to 1
- Decoder moves to the next instruction each time
- Stop sends a signal back to the and gate which stops counter

Multiplexer / Demultiplexer

Multiplexer turns n signals into a single signal. Chooses which signal to let through. **Demultiplexer** turns a single signal into n signals. The single signal chooses which signal to output.

Software vs Hardware Design

Unlike software, which uses iteration, hardware uses replication. The advantages of replication are increased elegance, higher speed, and increased reliability. Hardware also uses gate minimization, abstraction and power optimization.

Fixed & Programmable Logic

Fixed logic circuits: Pre-determined function.
Programmable logic: FPGAs (reprogrammable, but still a significant cost to switching functions). **Stored program and re-programmable circuits:** Your computer right now.

Data Encoding

1 Byte = 8 bits. 1 **Byte** encodes a character, integer, or pointer. 1 **Word** is n bytes, determined by the architecture.

Converting between bases

Base 10 to Base N Divide decimal # by # of new base. Take remainder as rightmost digit. Divide quotient of previous divide by new base. Repeat until quotient is zero.
Base N to Base 10 Take each column position of each digit, zero indexed, as n. For each column, do $c \cdot b^n$, where c is the value of the column, and b is the base value in base 10.

Important bases table

Hex	Bin	Dec	Hex	Bin	Dec
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	A	1010	10
3	0011	3	B	1011	11
4	0100	4	C	1100	12
5	0101	5	D	1101	13
6	0110	6	E	1110	14
7	0111	7	F	1111	15

Fraction to Binary

1. Let x be the decimal part of our fraction.
2. Let b be our binary output.
3. Multiply x by 2.
4. If $x \geq 1$, subtract 1 from x and append 1 to b.
5. If $x < 1$, append 0 to b.
6. Goto step 3 until $x! = 0$.

Signed Integers

- **Two's complement:** Flip all bits, add 1. Range is $-(2^{n-1}), +(2^{n-1}-1)$
- **Sign-magnitude:** First bit is sign, rest is magnitude. Range is $-(2^{n-1}-1), +(2^{n-1}-1)$.
- **One's complement:** Flip all bits. Eg, +6 is 0110, so to get -6 go from $0110 \rightarrow 1001 \rightarrow 1010$. This has 2 zero, positive and negative zero. Range is $-(2^{n-1}-1), +(2^{n-1}-1)$.

Cast from Ints

You might want to cast a lets say 8 bit int, to a 16 bit int. How you do that?
sign-magnitude: Copy the MSB to the bigger ints MSB. Then take the remaining bits and fill them from the right to left.
one's complement and two's complement: Copy the Lowest order bits (the bits except the sign defining bit) and copy them to the other ints lowest order bits.
Take the MSB and make all the remaining bits in the new int the MSB.

IEEE-754 Floats

Can represent values from 2^{-126} to 2^{127} which is 10^{-38} to 10^{38} in decimal. If double precision is used then the range is 2^{-1022} to 2^{1023} which is 10^{-308} to 10^{308} in decimal. Separated into **sign, exponent, and mantissa**. Entire float is represented in binary, and the exponent is biased by $2^{b-1}-1$, where b is the number of exponent bits. Leading 1 is dropped from mantissa. To calculate from IEEE, do $1.(m_2) \times 2^{(e_2)}$. (Convert out of base 2 first). Range is approximately $2^{(-2^b)+2}, 2^{2^b-1}$, where b is the bits dedicated to the exponent, **minus 1**.

(a)

(b)

Special Values

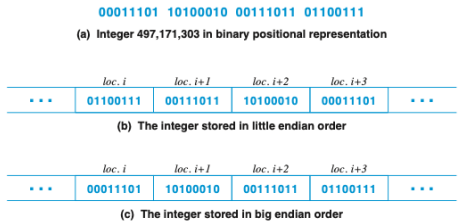
Exponent	Mantissa	Value
all 1s	all 0s	$\pm\infty$
all 1s	not all 0s	NaN
all 0s	not all 0s	denormalized
all 0s	all 0s	± 0

Binary-coded decimals

Instead of representing decimals using a float, use an arbitrarily long string of bytes, usually as binary ints. Efficiency can be improved using packed BCDs, by putting 2 ints in one byte (each digit occupying a nibble). 0x2D is used by the textbook to represent a negative, placed at the end of the BCD. For a packed BCD, drop the 2.

Endianess

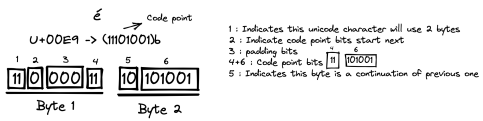
Endian: The way in which words(group of bytes) are stored in memory. **Big Endian:** Most significant byte first. **Little Endian:** Least significant byte first. Little endian is the most common method of storing data in memory.



ASCII and Unicode

ASCII: 128 characters, 7 bits. **Unicode:** 1,114,112 characters, 21 bits. Has variable length encoding for optimization i.e has as many bytes as needed.

Unicode Encoding



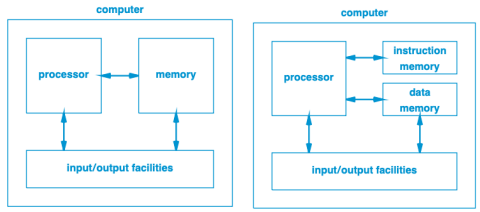
Types of architecture

Von Neumann architecture (left):

- Single memory block which contains both instructions and data.
- Offers complete flexibility: at any time, owner can change how much of the memory is devoted to programs and how much to data.
- More popular.

Harvard architecture (right):

- 2 separate memory. One is used for instruction, one is used for data.
- Inflexible, as you cannot use part of the instructional memory to store data and vise versa.
- Less popular. Sometimes used in small embedded systems.



Von Neumann Bottleneck

On computers running Von Neumann Architecture, time spent accessing memory can limit performance. To avoid the bottleneck, designs are chosen where operands are moved to registers instead of system memory.

Types of processors

A processor is a digital device that can perform a computation involving multiple steps.

Categories based on logic:

- **Fixed logic**: Function fixed in hardware, performs a single task
- **Selectable logic**: Choose one of several fixed functions.
- **Parametrizable logic**: Accepts a set of parameters that control the computation of fixed functions.
- **Programmable logic**: list of instructions provided at runtime (you can code them)

Categories based on Complexity:

- **Co-processors**: Dedicated function. Usually performs a single task at high speed. Used in -> Floating point accelerator. Fixed/Selectable logic.
- **Microcontrollers**: Direct hardware control. Used in -> Elevator doors. Programmable logic.
- **Embedded System Processors**: real-time OS, dedicated hardware. usually more powerful than microcontrollers. Used in -> smart phone Programmable logic.
- **General-purpose Processors**: compatible for multiple systems. Used in -> CPU in a PC. Programmable logic.

Parts of a processor

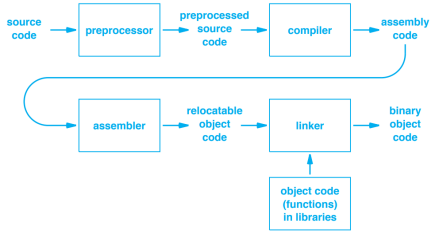
- **Controller**: Responsible for program execution. Steps through the program and coordinates the actions of all other units.
- **Arithmetic logic unit**: Performs all computational tasks. Performs one operation at a time according to controller.
- **Local storage (registers)**: Hold data values such as operands for arithmetic operations and the result.
- **Internal connections**: Transfers data values between units, like from local storage to the ALU. AKA data paths/Bus/Control lines
- **External interface**: Handles all communication between the processor and the rest of the computer system.

Fetch execute cycle

There is a **instruction pointer** which moves through the program performing every step. The cycle never ends while the system is runing.

1. Fetch the next instruction
2. Decode the instruction and fetch operands from registers
3. Perform the arithmetic operation specified by the **opcode**
4. Perform memory read or write, if needed
5. Store the result back to the registers
6. go to next instruction, Repeat forever.

Program Translation



- **Preprocessor:** Expands macros, producing modified source program.
- **Compiler:** Translates it to assembly.
- **Assembler:** Translates it to relocatable object code which contains references to external library functions.
- **Linker:** Replaces external function references with its code.

Fetch execute cycle notes

- Instructions are stored as binary data defined in a instruction set.
- The program starts at a constant point in memory which could be 0 or any other point, that part of memory should always have the necessary instructions to boot the system.
- Fetch execute cycle runs indefinitely, that's why in general purpose computers an OS is running at all times that manages the execution of programs.

CISC vs RISC

CISC

- Each instruction performs a complex operation
- Instructions may take multiple clock cycles
- Fewer instruction calls

RISC

- Each instruction performs a simple operation
- Instructions all take the same number of clock cycles
- Many instruction calls needed
- Allows for pipelining, as each part of the instruction takes the same amount of time

Pipelines

Allow for more than one instruction to be "processed" at the same time. Generally 5 stages:

1. Fetch next instruction
2. decode & fetch operands
3. perform arithmetic operation
4. read or write memory
5. store result

Pipeline Stalls

Also known as hazards. 3 main types:

- **Data Hazards:** Waiting for data from an earlier instruction. Can be dealt with using data forwarding (allowing data to be used before it exits the pipeline), re-arranging instruction order.
- **Control Hazards:** Incorrect instruction is in the pipeline. Occurs during jump instructions/branching. Jumps are not executed until the fifth stage, so instructions directly after are fetched inside the pipeline. Can be dealt with using conditional branch prediction, flushing pipeline if prediction is wrong.
- **Structural Hazards:** Resource conflict (usually from external source) (eg sombody else is accessing the same register bank). Can be dealt with by loading data in parallel, eg using multiple banks.

Time	clock	stage 1 fetch instruction	stage 2 fetch operands	stage 3 ALU operation	stage 4 access memory	stage 5 write results
	1	inst. K	inst. K-1	inst. K-2	inst. K-3	inst. K-4
	2	inst. K+1	inst. K	inst. K-1	inst. K-2	inst. K-3
	3	inst. K+2	(inst. K+1)	inst. K	inst. K-1	inst. K-2
	4	(inst. K+2)	(inst. K+1)	–	inst. K	inst. K-1
	5	(inst. K+2)	(inst. K+1)	–	–	inst. K
	6	(inst. K+2)	inst. K+1	–	–	–
	7	inst. K+3	inst. K+2	inst. K+1	–	–
	8	inst. K+4	inst. K+3	inst. K+2	inst. K+1	–
	9	inst. K+5	inst. K+4	inst. K+3	inst. K+2	inst. K+1
	10	inst. K+6	inst. K+5	inst. K+4	inst. K+1	inst. K+2

Figure 5.5 Illustration of a pipeline stall. Instruction K+1 cannot proceed until an operand from instruction K becomes available.

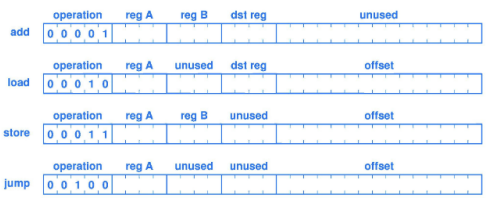
Branching

Moving the instruction pointer to a different location in program. Can be either absolute branch, or relative branch. Branch prediction can be used to try to run code from a branch before the processor has the data needed to evaluate it, speeding up runtime.

Instruction Sets

Generally has the following parts: Operation number, registers, offset.

- **Opcode (operation code)**: Specifies the operation to be performed.
- **Registers**: Specifies the operands and the destination.
- **Offset**: Think of it like array indexes. Can be a signed integer to move backwords.



Design choices

Encoding length

Variable-length encoding can improve instruction density, but **fixed-length** instructions are simpler to implement in hardware, and are thus more performant. Unused bits are ignored by the instruction.
Offsets are used to encode immediate values (generally used for jumping).

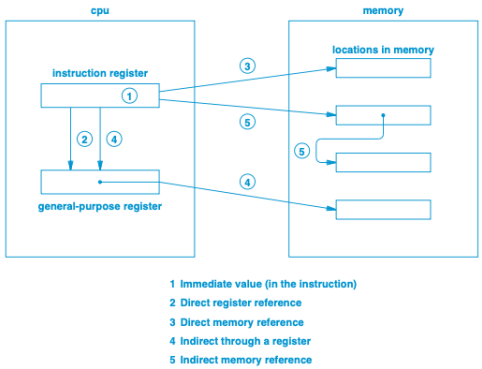
Number of Operands

Zero operands: Stack architecture, using push and pop. All operands are implicit. **One operand:** Implicit destination (usually a special accumulator register) **Two operands:** Specified destination, but uniary operations (eg `add rA, rB #rA=rA+rB`) **Three operands:** Specified destination, binary operations TL;DR, more operands = more flexible instructions, but more space taken up by operands

Implicit vs Explicit Encoding

Implicit Encoding: Operand types are always the same for a given opcode. Different opcodes are used for different types. **Explicit Encoding:** Operand field specifies what type of operands are being provided.

Operand Adressing Modes



Orthogonality

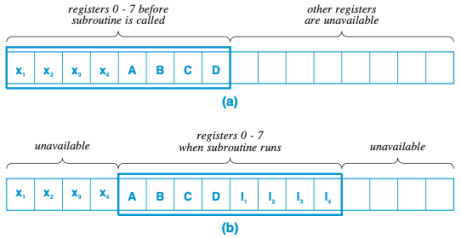
Each instruction should perform a *unique task*, without duplicating or overlapping the functionality of other instructions. Advantages: Orthogonal instructions can be understood more easily, and programmers don't need to pick between functions that perform the same task.

Registers

- **General Registers**: Fixed size (usually 32 or 64 bits), 2 basic ops, fetch and store. Numbered from 0 to $N-1$.
- **Floating Point Registers**: Separate set of registers holding floats, but numbering overlaps. Floating point registers are automatically used if instruction requires FP.
- **Special Registers**:
  - **Program Counter (*pc*)** - Stores the address of the next instruction to fetch.
  - **Comparer (*cmp*)** - Stores the result of the last comparison operation. (1 for true, 0 for false).
  - **Accumulator (*acc*)** - For zero and one-operand architectures to store the result of the last command.

Subroutines and Register Windows

When calling a subroutine, registers in use will partially shift down, making some of them unaccessible, some new registers available, and keeping some between both calls. This allows for values to be passed to and from the subroutine, while keeping some values separated.



Register Banks

- Allows parallel access within same clock cycle -> efficiency
- Some operations require operands from banks
- Register bank conflicts

Register Conflicts

Accessing 2 registers from the same bank simultaneously causes a register conflict. Best case, it causes a stall in the pipeline. Worst case, it causes the system to crash.

Solution

- reassign
- moving registers
- insert an instruction to copy values to the opposite register.

Physical Implementation

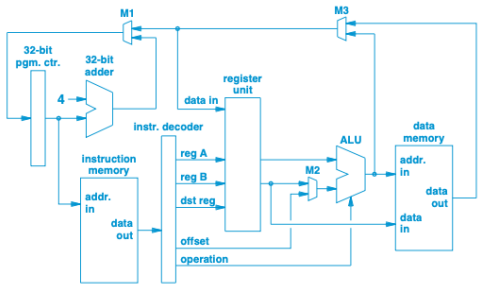


Figure 6.9 Illustration of data paths including data memory.

Instruction	M1	M2	M3
add rX,rA,rB	0	1	0
load rX,off(rA)	0	0	1
store rb,off(rA)	0	0	X
jump off(rA)	1	0	0

- Core loop between M1, 32-bit pgm. ctr., 32-bit adder
- Instruction memory returns instruction at given address
- Instruction decoder takes instruction and decodes it into individual parts
- Register fields used to select registers used in instructions, register unit takes fields and returns contents
- M2 Multiplexer takes auxiliary adding functions (such as adding an offset) and passes it through ALU
- ALU performs operation. For addresses, it's passed directly to data memory to get data out, for operations, data is passed to multiplexer to be stored into register for future use.

Binary String	Unsigned (positional) Interpretation	Sign Magnitude Interpretation	One's Complement Interpretation	Two's Complement Interpretation
0000	0	0	0	0
0001	1	1	1	1
0010	2	2	2	2
0011	3	3	3	3
0100	4	4	4	4
0101	5	5	5	5
0110	6	6	6	6
0111	7	7	7	7
1000	8	−0	−7	−8
1001	9	−1	−6	−7
1010	10	−2	−5	−6
1011	11	−3	−4	−5
1100	12	−4	−3	−4
1101	13	−5	−2	−3
1110	14	−6	−1	−2
1111	15	−7	−0	−1

Figure 3.9 The decimal value assigned to each combination of four bits when using unsigned, sign-magnitude, one's complement, and two's complement interpretations.