Data

Physical representations of data

- **Data loss is inevitable with analogue signals as noise is inevitable in any electronic circuit and in a analogue signal the amplitude of the current encodes the data**
- **Digital signals are resilient to electrical noise as the amplitude of the current can vary without affecting what data is represented** (provided it does not cause it to end up the wrong side of the threshold)
- For 5V digital logic:
  - 0-0.8V is a 0
  - 2.4-5V is a 1
  - 0.8-2.4 is undefined
- When a digital signal goes wrong it really goes wrong whereas analogue signals go wrong more gradually
- In 1965 Gordon Moore theorized that the number of transistors in integrated circuits doubles every year — in 1975 he changed it to every two years and it has held ever since
- Word size = the number of bits that a computer can process simultaneously — currently typically 32 or 64
- Each wire in a bus transfers 1 bit per clock cycle — bus width is typically but not always equal to word size

Representing numbers: Integers

- **To convert a number into a different base: divide it by the new base, record the remainder, then repeat with the quotient, once you have have a quotient of 0 stop and write the number left to right using the remainders bottom to top**
- Using two's complement as the representation for negative numbers allows the same circuitry to be used for addition and subtraction which saves on materials cost and energy consumption
- **An unsigned int with n bits has range 0…0** (0) **to 1…1** ($2^n - 1$)
- **A sign and magnitude int with n bits has range 11…1** ($-(2^{n-1} - 1)$) **to 01…1** ($2^{n-1} - 1$)
- **A two's complement int with n bits has range 10…0** (-2^{n-1}) **to 01…1** ($2^{n-1} - 1$)
    - If all bits are 1 the number is −1
    - **To turn an unsigned binary integer into a negative two's complement integer: flip all bits to the left of (but <u>not including</u>) the <u>least</u>-significant (ie rightmost) 1— if there isn't a leading 0 add one before flipping (so you end up with a leading 1)**
- **A biased two's complement int** has range 00… ($-(2^{n-1} - 1)$) to 11… (2^{n-1})
    - **Number is stored in unsigned binary after having a bias added to it to ensure it is positive — bias is often $2^{n-1} - 1$**
        - Can do >, < etc by pretending they are unsigned and get the correct answer — saves cpu cycles
- To subtract in binary, convert the subtrahend to be negative in two's complement then carry out binary addition
    - **Discard the overflow bits if one occurs, this is not an overflow in the traditional sense — you need to discard it to get the correct answer**

Representing numbers: Floats

- **Stored in binary as SEE…MM…** — mantissa is in sign and magnitude but they are separated by the exponent
- **Number = sign ✕ (1 + mantissa) ✕ 2^{exponent}**
  - If exponent is 00… then it is treated as 00…1 and 1 + mantissa becomes just mantissa — this allows 0 and subnormal numbers (numbers smaller than $2^{-\text{bias}}$ to be represented)
  - If exponent is 11… then number is either infinity (mantissa 00…) or NaN (mantissa is flags showing cause)
- **Exponent is stored in biased two's complement**
- **To convert from denary to IEEE754 float:**
  1. **Set S (1 for negative, 0 for positive)**
  2. **Write in fixed point binary as a positive number**
  3. **Similarly to base 10 scientific notation, normalise number to be greater than or equal to 1 and less than 2 — get into the form …001·… adjusting the exponent from an initial value of 0 to reflect how the binary point has been moved**
  4. **Mantisaa = everything after binary point rounded to have correct number of bits**
  5. **Add the bias ($2^{n-1} - 1$) to the exponent and represent as an unsigned int**
- The smallest non-zero non-subnormal positive normal floating-point number has exponent 000…$\underline{1}$ and mantissa 000…1 and the largest has exponent 111…$\underline{0}$ and mantissa 111…
- Single precision uses 8 bit exponent, 23 bit mantissa, 1 sign bit
- Half precision uses 5 bit exponent, 10 bit mantissa, 1 sign bit
- Double precision uses 11 bit exponent, 52 bit mantissa, 1 sign bit
- Quadruple precision uses 15 bit exponent, 112 bit mantissa, 1 sign bit

Logic

Boolean Algebra

- Karnaugh maps are very useful but remember that they don't capture XORs well
  - Each axis of the grid must have only one digit flipping at a time (eg 00 01 11 10)
  - Split the expression wherever the is an OR to form multiple expressions
  - One expression at a time put 1s in the grid where that expression is true
  - Draw a box around horizontally or vertically adjacent 1s to form a group that is a power of 2 in size and is a large as possible — boxes can wrap round from one side to the other
  - Construct boxes until every 1 is in at least one box — overlapping boxes are fine, each box just need to be as big as possible and a power of 2 in size
  - Create a boolean expression that exactly describes each group and puts an OR between the expressions for each group
  - X denotes a do not care condition (undefined behaviour) — pick whether to treat as a 1 or 0 to make the best groupings
- Distributivity: $X \land (Y \lor Z) \equiv (X \land Y) \lor (X \land Z) \equiv X \land Y \lor X \land Z$
- Distributivity: $X \lor (Y \land Z) \equiv (X \lor Y) \land (X \lor Z)$

Computational logic

Adders

- **1 bit half-adder** with inputs a and b
  - **Sum = a XOR b**
  - **Carry = a AND b**
- **1 bit full-adder**
  - **Sum = a XOR b XOR carry$_{in}$**
  - **Carry$_{out}$ = at least 2 inputs are T** = ((a XOR b) AND cin) or (a AND b)
  - **Can be built by passing a and b into a half adder to produce s_1, c_1; then passing s_1 and c_{in} to a second half adder to produce s_{out}, c_2; then passing c_1 and c_2 into an OR** (or equivalently an XOR) **to produce c_{out}**
- **N-bit full adder and subtractor**
  - To subtract we need to convert subtrahend to twos complement, this can be achieved by passing each bit through a NOT gate and passing in 1 as carry$_{in}$ to the first 1-bit adder
  - An XOR will flip the other input iff one of its inputs is a 1

Decoders and muxes

- **A decoder has input pins $x_0, \ldots, x_{n-1}$ and output pins $y_0, \ldots, y_{(2^n)-1}$**
  - **Let m be the denary number represented by the input, then y_m is active and every other y is inactive**
- **An encoder is the inverse of a decoder**
  - If more than one input bit is active, is a do not care
- **A multiplexer** (mux) **has input pins $x_0, \ldots, x_{(2^n)-1}, s_0, \ldots, s_{n-1}$ and an output pin y**
  - **Let m be the denary number represented by $s_0 \ldots s_n$, then $y=s_m$**
- **A de-multiplier** (demux) has input pins **$x, s_0, \ldots, s_{n-1}$ and output pins $y_0, \ldots, y_{(2^n)-1}$**
  - **Let m be the denary number represented by the input, then y_m is x and every other y is inactive**

Sequential logic

Misc

- **In computational logic the output is solely dependent on the inputs (time independent) whereas sequential logic has state and so output can also depend on what inputs were provided in the past**

Latches and flip-flops

- **A SR latch can be built by taking two NAND gates and connecting the output of each to one of the inputs of the other and connecting two active low inputs**
  - **If exactly one input is 1, Q=R, P=¬Q**
  - **If both inputs are 1, P and Q keep their previous values**
  - **Both inputs 0 is undefined behaviour**
  - Adding some extra gates allows a D-latch to be formed, has an input D and an enable E — sets Q to D if E is high and keeps current value of Q if E is low
- Latches are transparent (level-triggered), output changes effectively instantly when the input does unless they are in storage mode
- Flip-flops are edge-triggered (not transparent), output can only be affected by input for an instant on each clock tick
- **A d-type flip-flop has inputs D and clock — sets Q to D if clock is on a rising edge and keeps current value of Q otherwise**
- A t-type flip-flop has inputs T and clock — Q and hence P flip on rising clock edge if T is high and keep their current values otherwise
  - If T is high, then Q is a clock with half the frequency of the clock in

Registers

- **n-bit register: Connect n d-type flip-flops to the same clock but each with their own inputs and outputs**
  - **Requires n input lines and can store n bits every clock cycle**
- **n-bit shift register: Connect n d-type flip-flops to the same clock each with their own outputs but with only one taking an external input and all others taking their input from the output of their predecessor**
  - **On each clock pulse, all bits shift by 1 place** (to the right in the diagram below)
  - **Requires 1 input line and can store n bits every n clock cycles**

Latches and flip-flops

- **n-bit count-down counter: Connect the T of a t-type flip-flop to high and the clock to a clock then take n−1 more t-type flip-flops and connecting each's T to high and clock to the Q of its predecessor**
- n-bit bi-direcitonal counter: Take an n-bit count-down counter and connect every output to its own XOR with a common mode line
  - High mode = count up
  - Low mode = count down

Buses

- **3-state buffer: Active-low enable pin as well as input pin and output pin**
  - **Disconnects output from input if enable is high**
  - **Allows whether a component is connected to a bus to be controlled electronically**

Memory

Memory hierarchy — very important!

- **Registers — random access — volatile — fastest — smallest capacity**
- **Cache — random access — volatile**
  - **L2 is larger, slower, and further away from the CPU than L1 and the same for L3 to L2**
- **Main memory — random access — volatile in general purpose computers, may be non-volatile in embedded systems**
  - Where we treat most data as being when using high-level languages
- **Secondary storage — sequential access — non-volatile**
- **Tertiary storage — sequential access — non-volatile — slowest — largest capacity**
  - Used for backups and archival

Cache misses

- **Compulsory miss = the data/instruction is not in cache because it has never been accessed before e.g. when a program starts running** — program is cold starting
- **Capacity miss = data/instruction was in cache but it was removed to make room for something else** as it was not frequently used — there is more data in active use than there is in the cache
- **Conflict miss = data/instruction was in cache but the mapping from data/instructions to cache locations caused the data/instruction to be overwritten when something else was added to cache** — there may have been free space elsewhere
- **Coherency miss = Data/instruction was in cache but was removed as it may have become stale due to use of concurrency**
- **Hit rate (h) = (number of requested words that were in cache)/(number of words requested)**
    - **Miss rate = 1−h**
- **Average memory access time = hit time (time to access word if is in cache) + miss rate x miss penalty (time taken to copy a word into cache)**

RAM

- **Static RAM uses cross-coupled transistors to store data** — has to be supplied with power continuously
- **Dynamic RAM uses the charge in a capacitor to store data**
  - Has to be refreshed after a read as the act of reading causes the charge to leak out
  - A read is forced every 50ns to force a refresh to avoid losing data to natural discharge
- DRAM is cheaper but SRAM is faster
- DRAM uses less power than SRAM
- DRAM cells have a smaller physical footprint than SRAM cells so can fit larger capacity in same area
- **RAM cells are stored on a physical grid to make one block — can select a column and a row and hence address each cell uniquely**
  - Typically arranged to be a square
- **Need $\log_2$(number of words) address lines and (word size) data lines**
- **Blocks are also arranged on a grid addressed by column and row**

Error correction

- Errors in volatile memory typically occur when the magnitude of electrical noise (caused either by heat or induction) is large enough to flip bits
  - Noise attends such mangitudes for short random bursts
- Could send entire message multiple times but this is ineffecent — using parity bits is more efficent
- **Even parity: an MSB is added that causes there to be an even number of 1s**
- **Odd parity: an MSB is added that causes there to be an odd number of 1s**
- A single parity bit detects 100% of single bit errors and 50% of burst errors but can't fix any of them
- ECC (error correcting code): Put each byte in a row, have a parity bit for each row and for each column
  - Can detect <u>and fix</u> 100% of single bit errors and detect over 50% of burst errors

10

Memory mapped IO

- **IO components are connected to the address bus and the data bus and each component is assigned a unique memory address (or several depending on how much data needs to be transmitted to/from it at once)**
- **To access any memory address (in main memory or some IO extension of it): Address is sent along address bus to a decoder which converts it into input to the enables of the 3 state buffers connecting components to the data bus in order to only access the memory at that address**
- **The CPU interacts with it as if it is main memory — easy to write code for**
- **Uses existing CPU instructions (reduces complexity of circuitry)**
- **Uses up some of the CPUs ability to address main memory — more of an issue for cpus with smaller word sizes**
- Alternative is to use special buses and special instructions (port mapped IO)

DMA (direct memory access)

- **A special purpose processor called a DMA** (Direct Memory Access) **controller (DMAC) deals with IO instead of the CPU**
- **Detached DMA: CPU, DMAC, IO, and main memory all share the same buses — easy to implement but inefficient — only one DMAC for all IO devices — DMAC is part of the computer**
- **Integrated DMA: Multiple DMACs are connected to the system bus, IO devices are connected to the DMACs over dedicated buses — DMACs are part of the devices**
- **Dedicated IO bus: CPU, a single DMAC, and main memory are all connected to the system bus and IO devices are connected to the DMAC over a shared IO bus — fastest transfer but most complicated to implement — DMAC is part of the computer**
- **CPU can get on with other tasks while the DMAC is in operation but only for as long as it only needs data that's already in registers/cache**
- **DMAC and CPU have to communicate to decide who gets to use buses when**
    - **Cycle stealing: DMAC waits for the system buses to be free and uses them speculatively — doesn't work well if have multiple DMACs trying to do this**
    - **Burst mode: DMAC requests use of system buses and keeps control until either it is finished or an interrupt occurs**

IO synchronization

- IO devices are nearly always slower than the CPU, so it has to be the IO device that controls the rate of data transfer for no data to be missed or corrupted
- **Handshaking: IO device sets a bit to say its ready and CPU sets a bit to say it has sent some data**
  - **Can be done in hardware or software**
  - Open-ended: Only CPU sets a bit
- **Polling checks the ready bit at regular intervals**
  - **Can be implemented in hardware or software and is easy to implement either way**
  - **Busy wait polling constantly checks the ready bit until it the device is ready — wastes CPU cycles and hence power**
  - **Interleaved polling carries out other tasks for a bit then checks the ready bit then carries out other tasks for a bit and so on until the device is ready — device can be stuck waiting for CPU to act for longer than desirable**
- **To avoid using polling can connect ready to a CPU interrupt line**
  - **More complex to implement**
  - **More time efficient and more power efficient**

Choosing secondary storage: Magnetic

- Lowest cost per GB — good for data archival
- Produces noise
- Has lots of moving parts — moderately large physical footprint and vibration, drops, and knocks can all result in data loss — not good in mobile devices or embedded systems
- Requires periodic defragmentation

Choosing secondary storage: Optical

- Lowest absolute cost
- If you make backups onto a CD-R or a DVD-R you don't have to worry about anyone overwriting them
- Slowest read/write speeds
- If left in sunlight heat can warp leading to data loss
  - The UV light causes damage to rewritable discs on top of the damage from warping

Choosing secondary storage: Flash

- No moving parts - small physical footprint and low susceptibility to damage
- Very fast read/write speeds
- Very low power consumption and noise production
- Each cell can only be written to a finite number of times
- Will lost data if persistently without power for a long time as the charge in the cells slowly bleeds away and the data is encoded in the charge — not suitable for archival
- Highest absolute cost and cost per GB
- Useful in mobile devices and embedded systems

Processors

RTL (Register Transfer Language)

- Pseudocode for assembly
- **Main memory is represented as MS(*address)***
- [a] <- [b] means <u>copy</u> the contents of b to a
- **FDE cycle in RTL:**
    - **[MAR] <- [PC]**
    - **[PC] <- [PC] + 1**
    - **[MBR] <- [MS(MAR])]**
    - **[IR] <- [MBR]**
    - **CU <- [IR]**
    - **//Read in required data using MAR and MBR**
    - **//Carry out operations using ALU**
    - **//Do any required writes using MAR and MBR**

Assembly addressing modes (simplified slightly)

- **Immediate**
  - **The operand is the data**
- **Direct**
  - **The operand is the register that data is stored at**
- **Absolute**
  - **The operand is the address in main memory that data is stored at**
- **Indirect**
  - **The operand is the register that stores the address in main memory that data is stored at**
- **Relative**
  - **2 operands — 1 register and 1 constant**
  - **Indirect addressing but a constant** (an immediately addressed value) **is added to the register pre/post using the address in it** — e.g. copying instruction pointed to by pc to ir and post-incrementing pc
- **Indexed addressing**
  - **3 operands — 2 registers and 1 constant**
  - **Address in main memory that data is stored at is calculated using the sum of two registers and an immediately addressed value**

Interrupt handling

- After it has finished executing the current instruction and before fetching the next, the CPU checks if there are any interrupts by checking contents of a special register — if there are it will:
    1. **Push the contents of registers onto a stack in RAM**
    2. **Copy address of interrupt handler for the highest priority interrupt into program counter**
    3. **Interrupt handler executes until complete unless it gets interrupted by an interrupt with higher priority**
    4. Interrupt flag of interrupt that has finished being handled is cleared from register
    5. **Previous contents of the registers is popped off the stack and loaded back into registers and execution continues**
- When executing an interrupt handler, the CPU still checks for interrupts with higher priority at the end of every FDE cycle
    - This is why a stack is used — ensures that the previous interrupt is handled before execution of original task resumes

ALU

- Inputs:
  - Two data buses (one for each input)
  - Function select lines — determine which operation is performed
- Outputs
  - One data bus
  - Flags e.g. overflow

Control unit

- **The job of the control unit is to decode the instruction currently present in CIR and coordinate the rest of the processor to execute it**
- Inputs to control unit:
  - Opcode
  - Clock
  - ALU flags
- Outputs from control unit:
  - Enable signals
  - Clocks
  - ALU function select signals
  - Read/write control signal for main memory

Control unit: Hardwired

- Aka random logic
- Built with loads of logic gates
- Very fast
- Very complicated to design
- Very complicated to test
- Is set in stone, cannot change much with firmware updates
- **Opcode goes into a special decoder which then goes into the control signal generator (which produces all the outputs)**
- **Control signal generator also takes inputs from a flip flop that changes between fetch and execute and a sequencer which is connected to the clock**
- Sequencer has several output lines and enables one per clock cycle in sequence

Control unit: Microprogrammed

- I think this is how this works?
- Mappings between machine code instruction and microprogram instructions are stored in ROM
- Slower
- Less complicated to design and test
- Can add and modify instructions after manufacture through firmware updates
- **Opcode goes into a decoder which converts it to a microadress which goes into microPC**
- **Value at address in microPC is fetched from microprogram memory into microIR and microPC is incremented ready to to fetch the next microinstruction**
- **microinstructions are simply all the outputs of the CU concatenated together so microIR simply has a bunch of lines coming out of it**
    - Several microinstructions are used in the same way a hardwired moves through several sequencer states

RISC vs CISC

RISC	CISC
Used by ARM	**Used by Intel and AMD**
Hardwired CU — easier to manufacture	**Microprogrammed CU — harder to manufacture**
Fixed length instructions — pipelining is easy	**Variable length instructions** — pipelining is hard
Smaller instruction set — programs are expressed using a larger number of instructions — more RAM required (only really an issue in embedded systems and in them power consumption is more important)	**More instructions — programs are expressed using a smaller number of instructions** — easier to write/generate assembly for
	Has to run at a higher clock speed to get same performance as RISC — **consumes more electricity** and generates more heat

Design limitations

- Transistor size is lower bounded by the wavelength of the laser used to create the transistors
- **Transistor size is lower bounded by quantum tunneling**
  - The smaller a transistor is the more susceptible to noise it is
- **If the atoms in transistors gain too much (thermal) energy the transistor won't operate properly**
- **Wires with smaller cross sectional area have higher resistance and hence produce more heat**
  - Wires with larger cross sectional area have higher capacitance which is undesirable
- **temperature ∝ power consumption ∝ clock speed ∝** voltage^2
  - Clock speeds have stopped significantly increasing as going higher would require impractical amounts of cooling