Trends in Computer Architecture

Observed that the number of transistors on a chip double every 18 months.
Memory capacity doubles every 2 years.
Disk capacity doubles every year.
Components: CPu, Memory, System Bus, IO bus, storage, peripherals.
Moore's law: number of transistors on a chip double every 18 months.
Von Neumann Model: single memory space store both instructions and data.
Multicore crisis: diminishing returns for multiple core clock speed.
Power Wall: Power consumption increases with clock speed. it levels out because cooling and heat become an issue **Memory Wall:** Memory access time is slower than CPU speed.

C info

Pointer: int *p; (declare pointer which points to an int), p = &x; (point to x), *p = 5; (set x to 5). &x is the address of x.
Array: int a[20]; (declare array of size 20), int *ptr = a; (pointer to array) acts as a[0].
Multi Dimensional Array: int a[2][3]; (2D array), int *ptr = a; (pointer to 2D array) acts as a[0]. a[0] $\sim$ first row, a[0][0] $\sim$ first element of first row. *(m+2)+1 = &m[2][1]
Memory: Malloc: create objects on the heap: int *numbers = (int *)malloc(sizeof(int) * n); free to deallocate. free(numbers)
Struct: struct point { int x; int y; }; struct point p; p.x = 5; p.y = 10; if we pass a pointer of a struct we do struct point *p; p = (struct point *)malloc(sizeof(struct point)); p→x = 5; p→y = 10;
Strings: Treat as array of chars followed by null. "\0". `strcpy`(s, "bar") to copy value but not same object. `strlen`(s) to get length. `strcmp`(s, t) to compare. `strcat`(s, t) to concatenate.
Input/Output: `scanf` and `printf` for input and output. %d for int, %f for float, %s for string. add f to the front of most operation to put them to a file.

Assembly

Cycle: Fetch, Decode, Execute. CISC $\sim$ complex, RISC $\sim$ simple. `opcode src dest` where opcode is the operation, src is the source, dest is the destination.
Registers: EAX, EBX, 32 bit, AX, BX, 16 bit, AH, AL, 8 bit. end x is like variable, end p/i is pointer/index.
Type of operands `movl $eax, %ebx` copy contnet of %eax to %ebx; `movl $0x1, %eax` copy 0x1 to %eax; `movl %eax, 0x1` copy %eax to memory location 0x1; `movl (%ebp, %esi), %eax` copy value at address = ebp + esi to %eax; `movl 8(%ebp, %esi), %eax` copy value at address = ebp + esi + 8 to %eax; `movl 0x80 (%ebx, %esi, 4), %eax` copy value at address = ebx + esi*4 + 0x80 to %eax
Operations: `movl, src, dest` dest = src. `pushl src` esp = esp-4 then move M[esp] = src `popl src` src = M[esp] then esp = esp+4. `leal` compute address using addressing mode without accessing memory **Flags** `ZF` zero flag it is zero, `SF` sign flag if it is negative, `OF` overflow flag if it is too big in 2s complement, `CF` carry flag if it is too big in unsigned.

Number theory

Hexadecimal: $A_{16} = 10_{10}$, $F_{16} = 15_{10}$, $10_{16} = 16_{10}$, Uses 16 bits so 2 bytes. (think float = 32 bit, 4 bytes, 4 digits)
Binary: $1010_2 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 = 10_{10}$
Sign-Magnitutde: First bit is sign, rest is magnitude. $-5_{10} = 1001_2$ Arithmatic is odd, distinct ± 0
1's Complement: Negate by flipping bits. $-5_{10} = 1110_2$ Arithmatic is mostly normal, distinct ± 0
2's Complement: Negate by flipping bits and adding 1. $-5_{10} = 1011_2$ Arithmatic is normal, distinct 0
IEEE: sign, exp, mantissa. $\rightarrow$ sign $\cdot\ 2^{e-b} \cdot$ mantissa Bias: e = num exp bits, b = $2^{e-1} - 1$. subnormal, with all 0 exp leads to 0 as first digit
Special values: 0 = all zeros, $\pm\infty$ = all ones in exp, 0 in mantissa, change sign. NaN = all ones in exp, non-zero mantissa.

Digital Logic

Gates: AND, OR, NOT, NAND, NOR, XOR, XNOR.
Decoder: n inputs, 2^n outputs, for the input in binary the same index is 1, rest are 0.
Encoder: 2^n inputs, n outputs, inverse of decoder.
Mulitplexer: n selector, 2^n inputs, 1 output, output is equal to one of the inputs based on the selector.
Minterms: Product of all variables in a truth table. Ie $A\bar{B}C$ is a minterm for A=1, B=0, C=1 O=1.
SOP and POS: Sum of Products, Product of Sums. SOP to POS: multiply through, add terms. POS to SOP: Compliment, multiply through, compliment using DM's law.
Karnaugh Maps: 2D representation of truth table. Grouped in terms of 1, 2, 4, 8. The idea is to simiply the expression. The "variant" terms are the one removed and the rest is convered to minterms to lead to the minimal expression.
SR Latch: We have two NOR gates with an S,R and Q, Q'. Essntionally if S=1 then Q is set, and if R=1 then Q is reset. If both are 1 then it is invalid.
D Latch: 2 inputs C, D, C for control, D for data. If C=1 then Q = D, if C=0 then Q = Q.
D Flip Flop: 2 inputs Clk, D, Clk for control, D for data. If Clk=1 then Q = D, if Clk=0 then Q = Q. The difference is that the D flip flop is edge triggered.
Finite State Machines: Has state regsiter which store next state, and load the next state at clock edge. and combinatorial logic.
n-type Mos transistors: when gate has poitive voltage, short circuit between 1 and 2, then zero voltage, open circuit.
p-type Mos transistors: when gate has positive voltage, open circuit between 1 and 2, then zero voltage, short circuit.
CMOS: Complimentary Mos, n-type and p-type in series.

Caches

RAM: static: retains value with power, faster and more expensive, dynamic: needs to be refreshed, slower and cheaper.
Memory Hierarchy: Registers, L1 Cache on chip, L2 Cache off chip, main memory (DRAM), Disk, Remote server.
Cache Memory Small, fast, Look highest level: $1 \rightarrow 2$. We want memory to be local temporally and spatially as it addresses the gap between CPU speed and RAM speed.
Cache Hit/Miss Hit: data is in cache at level k, Miss: data is not in cache at level k so we must go to level k+1, if level k is full, the ome current block must be evicted.
Cache Miss Compulsory: first access, Conflict: when k level i large enough but multiple blocks map to the same location, Capacity: when the set of active data blocks is larger than the cache.
Cache Organization $S = 2^s$ sets, E lines per set, $B = 2^b$ bytes per block, with 1 balid bit per line, t tag bits. Cache size $= S \cdot E \cdot B$.
Addressing Caches: $< tag, set, block >$, The word at address A is in the cache if the tag bit in oneof the $< valid >$ lines in set $< set >$ matches $< tag >$
Direct Mapped Cache: Simplest Cache with only one line per set. Line matching Find a valid line in the selected set with a matching tag.
Set Associative Cache: Multiple lines per set. Set selection is the same, but line matching must compare the the in each valid line.
Replacement For Direct Mapped, only one choice, for Set Associative we can use FIFO, LRU, or random.
Fully Associative Cache: Set selection is trivial,
Write and Cache Hit: Write through: write to both cache and memory, write back: defer the write to memory for as long as possible. Miss: Write allocate: load the block into cache and update, no-write allocate: write directly to memory.
Write-back Cache: What to do if need to evict a block that has been written into. Write a caches copy to a lower level. Use a dirty bit to keep track of this.