Stack Machines	Operand stack to replace accumulator More intermediate results within the CPU -> less memory access required Code density, can reuse to implement recursion Fits naturally to equations (reverse Polish notation)
Reg. Machines	Small, fast local memory for intermediate results Practical for small memories to be multiported (allows for parallelism in CPU)
Amdahl's Law	$\text{speedup} = \frac{\text{entiretask perf w/ enchance.}}{\text{entiretask pery w/outenhance.}}$ Make common case fast
CISC	Eliminate the semantic gap Instruction usefulness limited or so general it is slow
Exceptions	Hardware (e.g. divide by 0) Software (e.g. SWI #) Invoke exception hander
Memory	SRAM: maintains store when provided with power 4-6 transistors per bit, fast DRAM: requires refreshing 1 transistor per bit, fairly fast
Latencies	Register file: 1 cycle L1 cache: 1-3 cycles L2 cache: 3-9 cycles Main memory: 10-100 cycles Note that DRAM can do burst reads and writes: 2-8 cycles Hard disk: 10^6 cycles!
Locality	Temporal: access recently used memory again soon Spatial: access close to recently used memory soon
Cache Line	Group of around 4 neighbouring words stored
Fully Associative	Store any word anywhere in the cache, lookup by address Huge overhead involved!
Direct Mapping	Use part of the address to map onto a cache line Compare tag / valid flag in line to determine if it really caches the address Behaves badly when using data from overlapping addrs.
Set Associative	As direct mapping, but with a set of cache lines at each location
Victim	As direct mapped but with a one line buffer to store the last line overwritten
Cache Line Replacement	Only applied to set/fully associative caches, clearly LRU: requires more info. NLU: pass the "potato" on if a cache line is accessed Random: simple, works well
Writes	If data is already in the cache then write over it If not in cache then either fetch on write / write around
Write Through	Data is written to both cache and lower level memory Common on multiprocessor systems for cache coherency Use bus snooping on SMP (does not scale > 2 CPUs)
Write Back	Data is written to cache only Written to main memory upon replacement, can use dirty bit to prevent this when it is unnecessary
Write Buffer	Avoid CPU stalling by buffering writes Also perform write merging: can take advantage of burst
Virtual and Physically Addressed	V + P addresses only differ in upper bits, if cache is no bigger than a page then can use lower bits of V to access the cache without conflict Otherwise use P address for cache or access with V and use P (concurrently found) to compare tags in cache Still have problems with aliasing of P addresses in V
TLB	Cache recent translations Fully associative cache TLB miss must be looked up Potential control hazard
Pipelining	Increases frequency and latency (due to latch time) Best to make pipeline stages of similar length
Exceptions	Imprecise: deep pipeline Precise: when you want OS to be able to restart it
Instruction Replays	Only find out about cache misses in next stage!

	This means that the pipeline has advanced too far: must replay instructions to give cache hardware more time While waiting for cache, refill the pipeline with instructions from the I-cache
Control Hazard	Caused by branching and not clearing the pipeline Could document behaviour and use branch delay slots
Data Hazard	Caused by not taking account of results yet to be written back that is still in other pipeline stages Use feedforward/bypass Sometimes have to stall pipeline if later instructions depend on e.g. memory fetches: can introduce bubbles, so minimise this Could document behaviour and use load delay slots Otherwise have hardware detect it (e.g. scoreboarding)
Parallel	Synchronous communication Timing assumption can be violated by skew: doesn't work for high f or long s
Serial	Asynchronous transmission Use 8B/10B coding to guarantee the run length (allows clock recovery), DC balance (allows AC coupling)
Control Flow	Concurrency simulated via interrupts / scheduler Has to throw away register file, disrupt caching/pipeline Load operations cause stall
Data Flow	Model dependency graph in CPU, execute it concurrently Inherently concurrent and latency tolerant Easy to take advantage SMP Too much concurrency, so assignment is a problem Makes I/O difficult Ineffective use of very local storage (e.g. register, stack) Scheduling policies simple

More information may be needed, check what the exam questions ask on this ☺