**pC Programming** (ANSI C, C90)

* age vs &age (address)
* no boolean, use int 0 = false
* 0 for octal, 0x for hexadecimal
* ASCII contains 1 parity bit
* Pointers
  + %p to print address in 0x
  + int \*a\_ptr;
* Functions
  + Include function prototype before main
  + void swap(int \*, int \*)
* Arrays
  + Value: arr[0] or \*(&arr[0])
  + Address: &arr[0] or arr
* Strings are null-terminated
* Structs are passed by value
  + (\*ptr).name = ptr->name

**Number Systems**

* Sign-and-magnitude
  + Invert sign bit
* 1s complement
  + -x = 2n – x – 1 (invert all bits)
  + Add carry out of MSB
* 2s complement
  + 1s negative, then add 1
  + Discard end-carry in addition
* Excess-x representation
  + -x = 0
* IEEE 754: sign, exponent, mantissa
  + 1-8-23, add 127 to exponent, normalise mantissa

**MIPS**

* 32 registers x 32 bits
  + 1 word = 4 bytes
* General purpose registers, no data type associated
* Variable mapping: load initial val
* Label (control flow use) = anchor, not an instruction
* R, I, J instructions
  + RISC to optimise hardware

Note

* Immediate: 2s signed integer
* subi does not exist
* Shifting fills with 0s, rs = 0
* shamt is 0 in non-shift cases
* lw/sw: Base addressing (displace)
* Branching: PC-relative addressing, assume PC+4 always done
* Jumping: 4 MSBs taken from PC+4

ISA

Amdahl’s law – common cases fast

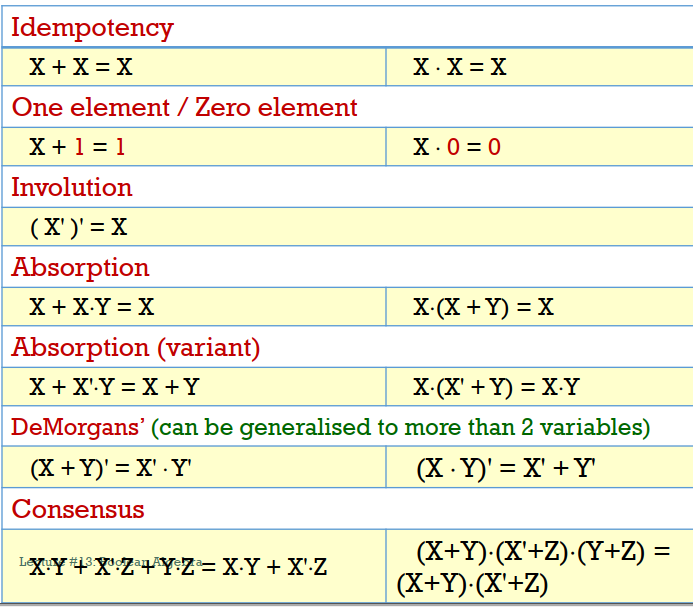
* Expanding op scheme

Datapath and Control

* Use control signals as MUX selectors, only MtR is reversed
* Multilevel decoding
  + ALUop + Func = ALUcontrol
* Find critical path for latency

**Boolean Algebra**

* Duality: Theorems are valid by reversing 1/0, +/.
* POS/ SOP standard forms
* F = m0 + m1, F’ = M0.M1
* NOT, AND, OR priority



Logic Gates

* {NAND}, {NOR} = complete logic

Simplification

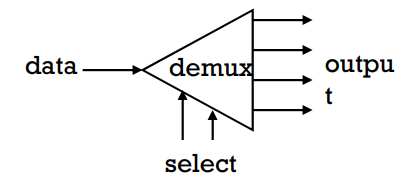
* Gray-code on K-maps
* Group cells including wrapping
* Prime implicants (PI) – max group
  + EPIs – essential, not overlap
* Don’t care = X or d

**Combinational Circuits**

* SSI -> MSI (scale of component)
* Input independent of present state
* Half-adder has no carry-in
* Parallel/ Ripple Carry Adder
* Circuit delay = wait for all inputs to be stable, then logic gate delay

MSI Components

* Enable pin, 1-enable = active-high
  + AND input with all output lines
* Decoder: n to m, m <= 2n
  + One line has output 1
  + 4x16 decoder using 2x4: select upper 2-bits first
  + Combine outputs with OR for SOP expression
* Encoder: only one input is high
  + Priority encoder used when more than 1 input is high
* Demux: data line, select pins



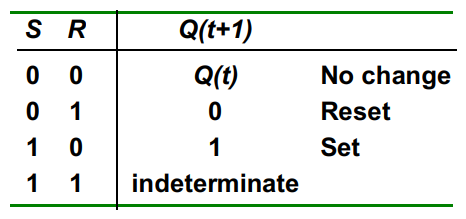
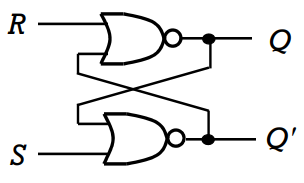
* + Select which output to follow the data line
  + Same as decoder with enable
* Multiplexer: data selector
  + 2n-1-1 mux for n variables
  + Use n-1 variables as selectors

**Sequential Logic**

* Synchronous? (follows clock)
* Level/ pulse-triggered?

S-R latch

* Synchronous, level-triggered



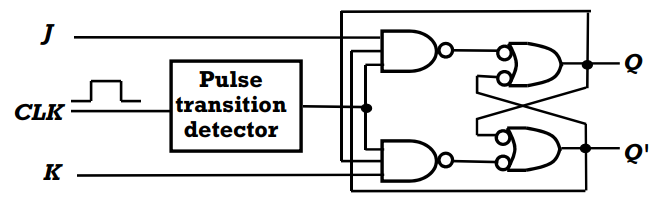
D latch

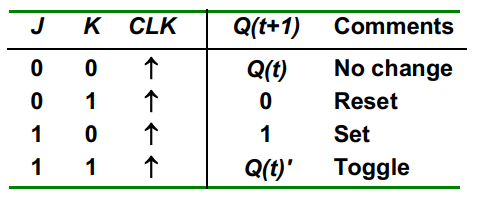
* R = S’, same input line
  + No invalid condition
* Gated = has enable pin

S-R flip-flop

* Pulse-triggered, changes at clock edge
* = positive-clock (clock goes from LOW to HIGH)
*  = negative-clock

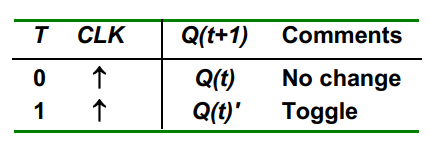
J-K flip-flop





T flip-flop

* J = K

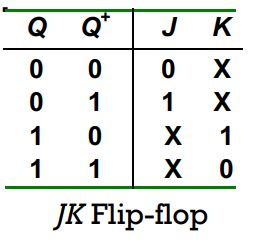
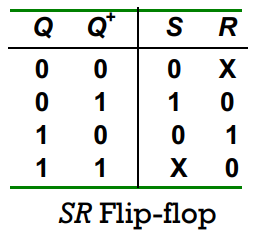


Flip-flops can include asynchronous inputs

* PRE, CLR, SD (set direct), RD
* Normal operation = all LOW

Sequential Circuit Analysis

* Given inputs -> Flip-flop inputs -> next input state (A+)
* State table -> state diagram
  + Input/ output
  + Stable? Self-correcting?
* Excitation tables
  + Derive circuit from state diagram

Memory

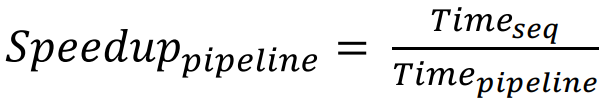
* Enable, Write = 0, Read = 1
* 1k x 8-bit RAM = 210 address, 8-bit data

**Pipelining**



* Improves throughput (total time)
  + Potential speedup ~ no. stages
* Steady state input (pipeline filled)
* Data stored separately (needed over multiple stages)
  + E.g. data values, PC + 4, ALU result, write register
  + Store in pipeline registers along w control signals



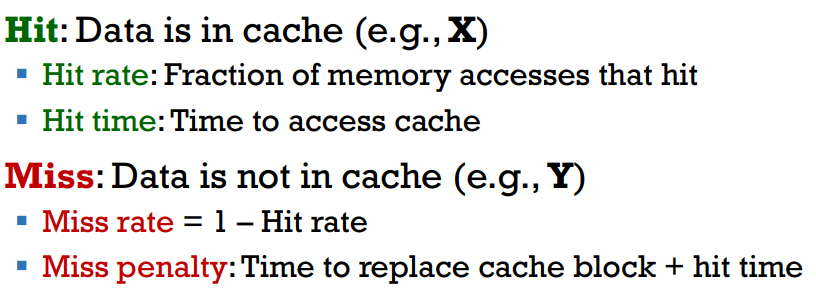
* Speed improvement
  + CTpipeline must include Td (pipeline delay)
  + 

Hazards

* Data dependency
  + Read-after-write (RAW)
  + Data can be *forwarded* from ALU to next ALU
  + MUX must cater to new input
  + Load: data only ready after MEM, stall 1 cycle
* Control dependency
  + Branching only decided in MEM, stall to next IF = 3 cycles
  + Early branching: decide branching in ID
    - Register comparison in ID
    - 1 cycle stall
    - Dependency on prev ALU result -> +1 cycle stall with forwarding
    - Dependency on previous load = wait for MEM, +2 cycle delay
  + Branch prediction: same prediction for all branches
    - Flush if incorrect
    - w/ early prediction: 1 flush, else: 3 flush
  + Delayed branching: execute instructions regardless of branch result

**Cache**

* DDR (double data rate)
* SRAM >> DRAM speed
* Temporal/ Spatial Locality of working set



Average Access Time

* Calculate using hit rate and hit time and miss penalty

Memory to Cache Mapping

* Line/ block size -> offset
* No. cache blocks -> index
* Else -> tag
* Valid bit catches cold miss
* Tag mismatch = conflict miss
* Writing data
  + Write-through cache (directly write to cache + memory), using a write buffer that functions independently of the processor
  + Write-back cache (only write to cache, write to memory when cache is evicted), uses dirty bit to determine if write operation occurred
  + Cache misses: write allocate (fetch data to cache) or write around (directly change memory)