1. Abstraction

Problem – Algorithm – Language – Instruction Set Architecture – Microarchitecture – Circuit – Device

Turing Machine: simplest computer

Undecidable Problem: not Turing machine computable

2. Beauty is in the eyes of beholder

PC contains the address of the next instruction to be executed

Execution of instruction determines next PC value

Instruction may access data in memory

Operations

Arithmetic: add, subtract, sign extension

-> overflow

Logical: and, or, not

Fractions: 1bit sign + 8bit exponent + 23bit fraction

N = (-1)^s \* 1.(fraction) \* 2^(exponent-127) (1<=exponent<=254)

N = (-1)^s \* 0.(fraction) \* 2^(exponent-127) (exponent==0)

Exponent==255

+infinity (fraction==0 && sign==0)

-infinity (fraction==0 && sign==-1)

NaN(not a number) (fraction!=0)

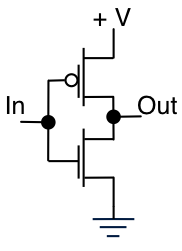
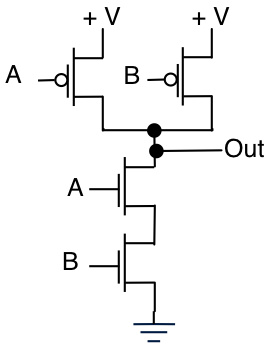
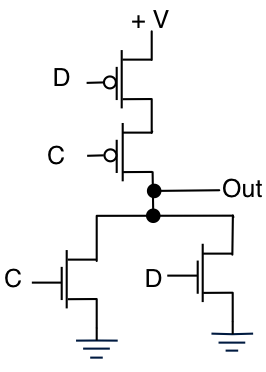
ASCII code (‘0’==48, ‘A’==65, ‘a’==97)

Software bug -> from amusing to nightmare

3. n-type MOS transistor: 1 = close, 0 = open, attached to ground

p-type MOS transistor: 1 = open, 0 = close, attached to + voltage

NOT, NAND, NOR gate: composition, duality, analogy

Combinational Logic Circuit: output depends only on the current inputs, stateless

-> always gives the same output for a given set of inputs

Sequential Logic Circuit: output depend on the sequence of inputs (past and present)

* Output depends on stored information plus input

Decoder: n inputs, 2^n outputs

Multiplexer(MUX): n-bit selector and 2^n inputs, one output

Full Adder: add two bits and carry-in, produce one-bit sum and carry-out

R-S Latch: R = reset or clear, S = set, active low logic

R=0, S=1: reset, R=1, S=0: set, R=S=1: maintain

Gated D-Latch: input D(data) and WE(write enable)

WE=1: set D, WE=0: maintain

Register: multi-bit value

Memory: address space(2^n locations), addressability(m bits)

SRAM: fast, data maintained as power applied

DRAM: slow but denser, storage decays

State Machine: type of sequential circuit

State Diagram: show states and actions that cause transition

Finite State Machine: states, inputs, outputs, state transitions, what determines output value

Clock circuit triggers transition

Master-Slave Flipflop: pair of gated D-latches, isolate next state from current state

4. ALU(arithmetic and logic unit): ADD, AND, NOT

Registers: operands and results of functional units, 8 registers, 16 bits

Word size: number of bits normally processed by ALU in one instruction or width of registers (16bits)

Instruction Register(IR): current instruction

Program Counter(PC): address

Instruction processing

Fetch(PC++) – Decode – Evaluate Address – Fetch Operand – Execute – Store Result

Opcode: operation to be performed

Operands: data/locations to be used for operation

3 kinds of instructions: computational, data movement, control

Stopping the clock: AND with 0 or control unit stops to seeing CLOCK signal

6. sequential, conditional, iterative

Debugging

Display/Deposit values in memory and registers

Execute instruction sequence in a program

Stop execution when desired

Errors

Syntax errors: 오타

Logic errors: 알고리즘 오류

Data errors: 입력 오류

Tracing

Single-stepping: one instruction at a time

Breakpoints: stop when it reaches a specific instruction

Watchpoints: stop when a register or memory location changes or when it equals a specific value.

* Don’t know where or when a value is changed

7. label, opcode, operands, comments

1st pass: scan & calculate corresponding addresses of labels(symbol table)

2nd pass: convert instructions to machine language, using information from symbol table

8. Memory-mapped vs special instructions

Asynchronous vs synchronous

Polling(CPU): keep checking status vs interrupts(device): receive special signal

Atomic instructions: CPU check signal between STORE and FETCH

9. service routine or system call: perform low-level, privileged operations safely and conveniently

TRAP: set service routines -> table of starting address -> instruction -> return to user program

Saving and restoring registers

called routine save any registers that will be altered

calling routine save registers destroyed by own instructions or called routines if values needed later

arguments: value passed in to a subroutine

return values: value passed out of a subroutine

generally use callee-save strategy except for return values

must save R7 if you call any other subroutine or service routine(TRAP)

* Otherwise, unable to return to caller

10. stack = LIFO, PUSH(overflow) & POP(underflow)

R6 = Top of Stack

Processor state(PS): memory, PC, GPR, PSR

Save processor state at stack

Supervisor stack: special region of memory used as the stack for interrupt service routines

Initial SSP stored in saved.SSP, other register for storing USP: saved.USP

Use R6 as stack pointer -> save R6 to saved.USP when switch from user mode to supervisor mode

RTI: return from interrupt, POP PC and PSR from supervisor stack, if PSR[15]=1, R6=saved.USP

* can only be executed in supervisor mode, If used in user mode, causes an exception

Exception: internal interrupt

In LC-3: executing an illegal opcode, executing RTI in the user mode

Else: divide by zero, accessing an illegal address