- 1's complement : Positive nums → Same as 2's complement
  Negative nums → Flip all bits

  - n-bit 1's comp cover: $[-(2^n-1),\ 2^n-1]$

- Overflow
  - For signed magnitude: it is a carry out of the leading digit
  - For 2's comp: a sum too large becomes negative
    a sum too small ____ positive

- Fractionary Binary Number:

  - 1.M: Ex: $1.0101 = 1*2^0 + 0*2^{-1} + 1*2^{-2} + 0*2^{-3} + 1*2^{-4}$
    (0101 = Mantissa (M))

  - IEEE-754: Expand the range of 1.M, use 32 bits

    - Rule: $(-1)^{s} * 1.M * 2^{E-127}$
      ($(-1)^s$: 1 bit; 1.M: 8 bits; $2^{E-127}$: 23 bits)

    - Drawbacks: big value sacrifices pricision because big value means E large → shifting the floating point to the right

    - Comparing 2 FL nums:
      - Positive > Negative
      - If both positive, compare from left → right, which has larger bit first is larger
      - If both negative, ____ ____ smaller

- ) Bitwise operator :

  - AND : AND every bit between A and B , denoted as A & B
  - OR : OR every bit between A and B , denoted as A | B
  - NOT : Flip every bit of A , denoted as ~A or A' (not for computer)
  - XOR : XOR _____ between A and B , denoted as A ^ B
  - NAND : A NAND B is denoted as ~ (A & B)
  - NOR : A NOR B is denoted as ~ (A | B)
  - left shift : shift all bits of A to the left B( decimal ) places, the gap created filled by 0 , denoted as A << B
  - Right shift : shift all bits of A to the right B _____ , the gap created filled by the leading digit , denoted as A >> B

- ) Manipulating bits using bool funcs :

  - CLEAR : zero out a bit → use AND with 0 at the same place
  - SET : make a bit become → use OR with 1
  - TOGGLE : flip a bit → use XOR with 1

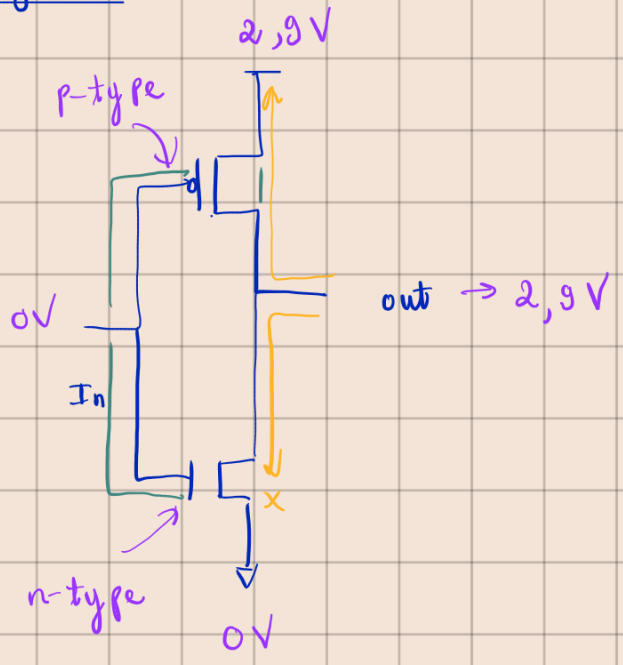
- ) Change from base $2^n \rightarrow 2^m$ : $2^n \rightarrow 2 \rightarrow 2^m$

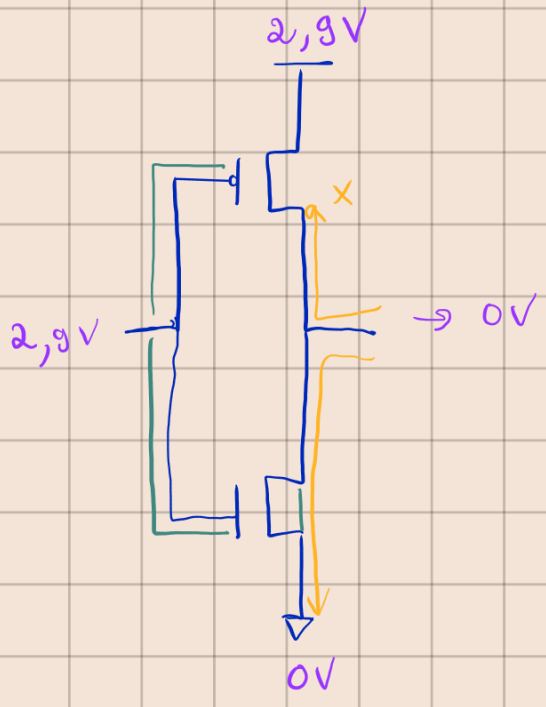
- ) ASCII : Binary can be used to repre char, by assign a char with a binary num
  - uppercase → lowercase : SET bit 5
    lowercase → uppercase : CLEAR bit 5

- ) Transitor : The bitwise operators can be encoded by circuits , by using transitor

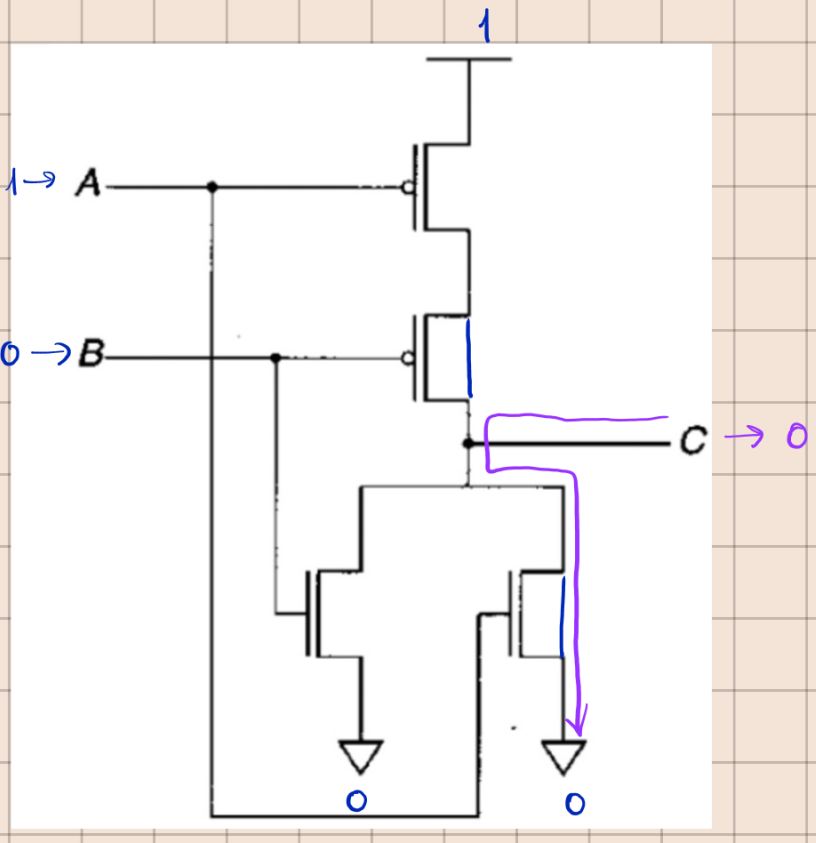
- N-type transitor : 1
- P-type transitor : 0

- NOT gate :

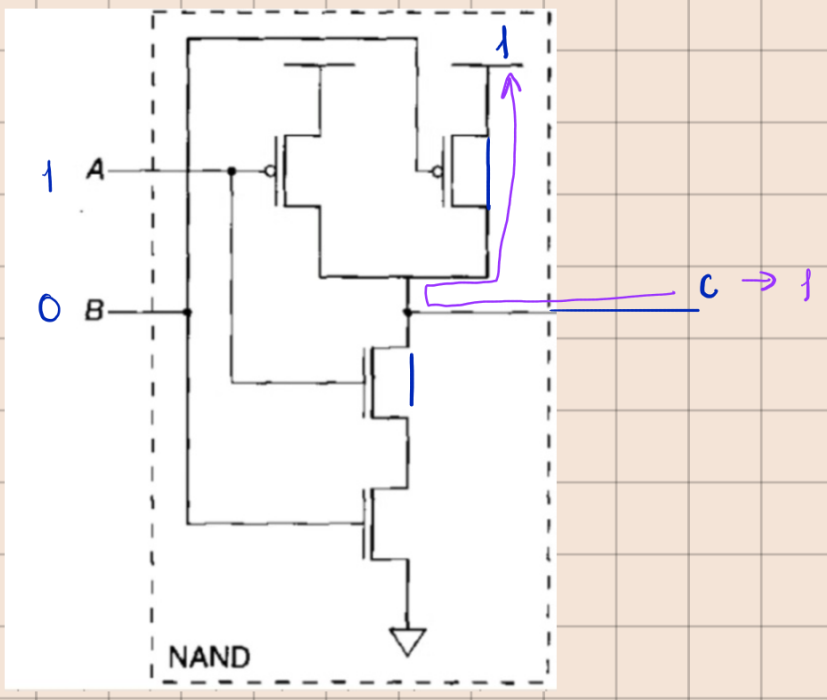



- NOR gate




- NAND gate :

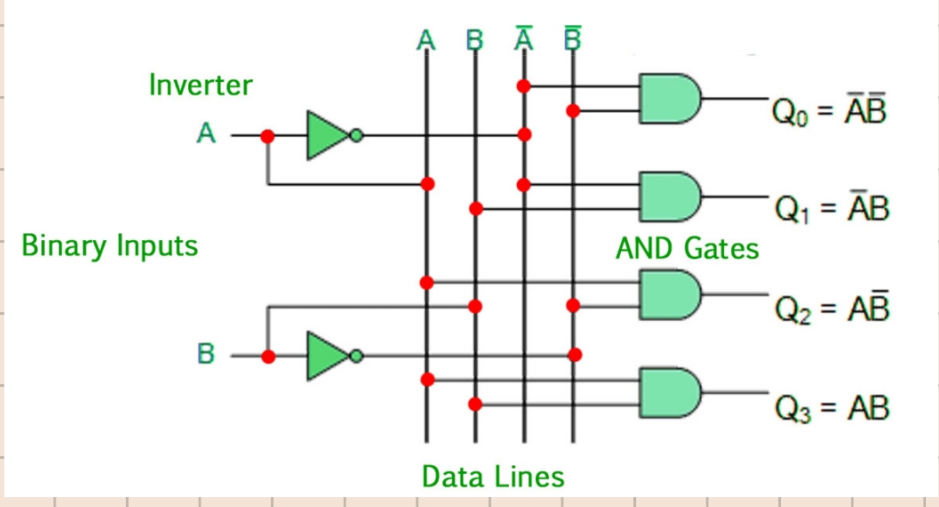



Note : Don't connect { N-type transitor to power
P-type ——— ground

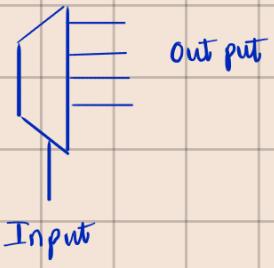
Because there will be floating voltage across the transitor

·) Combinational Logic Circuits : output depends only on the inputs

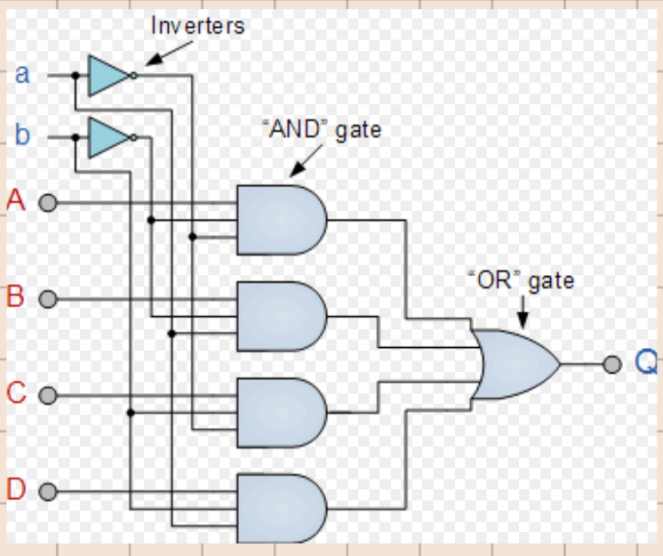
- Decoder : Output has one bit 1 and the rest are 0 repre the chosen combination of value of the input bits
  - n-bit input
  - $2^n$-bit output



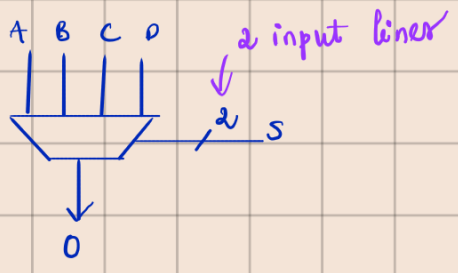

Denote as :



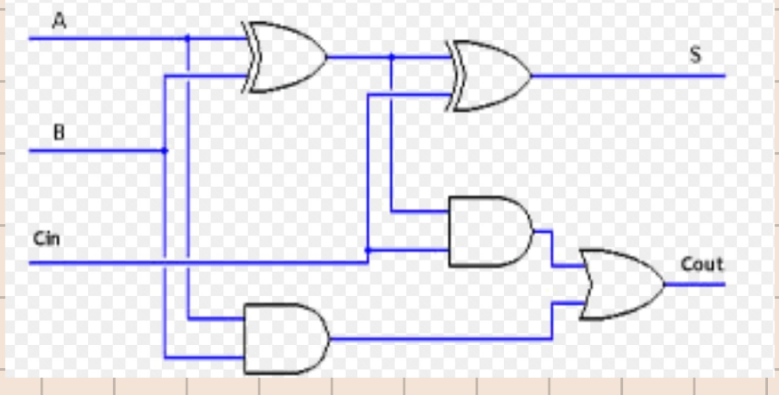
- **Mux** : select one of the inputs to be the output
  - n input lines ( inverters)
  - $2^n$ inputs to choose which to output



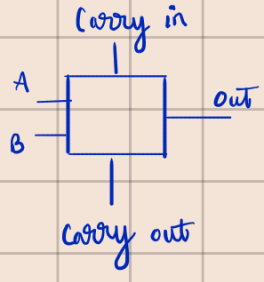

Denote as :




- **One Bit Adder** : For form A + B where A and B are 1-bit number
  - Three 1-bit inputs : A, B, and the carry-in
  - Two 1-bit outputs : output bit and the carry-out




Denoted as :




  - N-bit adder : Combine N one-bit adders, let the carry out of the previous place be the carry-in of the current place

:) **Boolean algebra** : Some useful equations

- $A + AB = A$
- $A + \bar{A}B = A + B$
- $(A+B)(A+C) = A + BC$

) From Truth Table to Circuit : Choose the rows that has output 1
- Each AND gate repre one row of the true table
- OR all the outputs of the AND gates

) K-map : A neat way to encode the Truth Table

- Valid grey code : One entity changes from one state to the next (even from the last state to the 1st stage)

- Get the boolean expression from K-map :

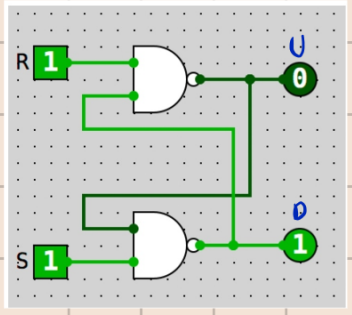
① Group the 1s on the table in group of $2^n$ $(n \in \mathbb{N})$

② Add the boolean exprs of all the groups together

Note : "X" is also counted as 1

) Sequential Logic Circuit : Output depends on both current input and stored elements ( in the past)

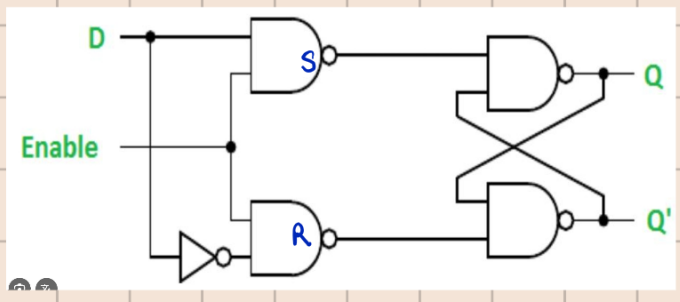
) R/S Latch : When both R and S are 1, one changes doesn't affect the output




It's possible that when R and S are both 1, output is U = 1, D = 0 or U = 0, D = 1

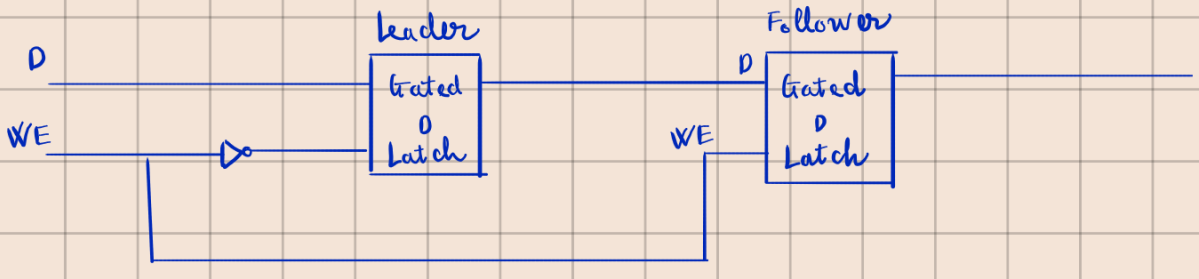
- Gated D Latch : When Enable = 1, D = 1, output is Q = 1, Q' = 0
  Enable = 1, D = 0, output is Q = 0, Q' = 1

  When Enable = 0, Q and Q' keep their previous values despite D changing




- Register : An array of Gated D Latches

- Leader - Follower Flip Flop : Allow the output to be controlled by clock cycles




- Triggering : allowing the circuit to be active, i.e. it can take input and give output

  - Level trigger : The circuit becomes active when the gating or clock pulse is on a particular level

  - Edge trigger : The circuit becomes active at negative or positive edge of the clock signal

  - Flip - flop is edge triggered

- State machine : control system with finite # states

  - How to maintain state ? Memory
  - What state to go next the circuit output depends on the current state and circuit inputs

- Moore machine : it output values are determined solely by its current state

- Algorithm for creating state machine logic :

  (1) Label the states

  (2) Create the truth table from the state diagram using all values of the input lines and current - state lines

  (3) Write a sum - of - products Boolean equation for each output value (next state and output lines) from the truth table

  (4) Simplify the equations

  (5) Implement the equations with gates

- Von Neumann Model : A fundamental model of a computer for processing comp programs, consisting of :

  - Memory
  - A processing unit
  - Input
  - Output

- A control unit

- Comp program: a set of instructions, contained in memory
  - The data the program needs is either in input or mem

- Address space: # distinct memory locations

- Addressibility: capacity of each location (# bits each location can store)

- Reading from a mem location:

  (1) Place the addr of that location in the mem addr reg (MAR)

  (2) The data in the location then put into mem data reg (MDR)

- Writing to a mem location:

  (1) Place the addr of that location into MAR

  (2) Place the data to write into MDR

  (3) Turn on Write Enable Signal (WE)

- Processing Unit: ALU + 8 general purpose register (GPR) (R0 → R7)

- Input / output: keyboard / monitor

- Control Unit: keep track of both where we are in the process of executing the program and executing each instruction
  - Inst are processed one at a time

- Program Counter (PC): Holds the current program counter (addr of the next instruction)
  - 3 ways to update PC:
    - Given a value for the PC via address adder
    - PC = PC + 1
    - Given the value for the PC via the bus

- Instruction Register (IR): holds the value of the current instruction on the processor

- Instruction set: The LC3 ISA instruction set has 15 instructions
  - Not 16 because one is reserved for future use

- Operand: An operand can be found in 3 places
  - Part of the instruction → immediate value
  - Register
  - Memory

- Addressing mode: A formula to calculate the addr of a mem location to be read / write

- PC-relative: mem_addr = PC* + offset

- Indirect mode: mem_addr = mem[PC* + offset]
  - mem: go in to see what in the mem loc

- Base + offset mode: let R0 = x3001 be the base register

  offset = 3

  mem_addr = R0 + offset = x3001 + 3 = x3004

(what inst is it?)

- Instruction: opcode + operands

  - 3 kinds of inst:
    - Operates: operate on data (ADD, AND, NOT)
    - Data Movement: move info from processing unit to and from mem and to and from I/O devices
    - Control: Altering order of instructions (normally, the inst in the next mem loc will be processed)

- ADD: 2 source operands + 1 destination operand
  - ≥ 1 of them is stored in GLR
  - GLR

  - Note: 3 bits to identify a GLR (from 0 → 7)

  - Ex: 0 0 0 1 | 1 1 0 | 0 1 0 | 0 | 0 0 | 1 1 0
    - 1 1 0: R6 ↓ dest operand
    - 0 1 0: R2
    - 0: 2 source operands taken from GLR
    - 1 1 0: R6

  Do: Add R2 and R6, store result to R6

  0 0 0 1 | 1 1 0 | 0 1 0 | 1 | 0 0 1 1 0
  - 1 1 0: R6 ↓ dest operand
  - 0 1 0: R2
  - 1: 1 source operands taken from GLR
  - 0 0 1 1 0: immediate value 6

  Do: Add R2 and 6, store result to R6

·) LD (load) : go into a mem location, read the value in there, and store it in one of the GPR

Ex : 0 0 1 0 | 0 1 0 | 0 1 1 0 0 0 1 1 0

R2 → dest operand

198 → off set, used addressing mode

• Use PC-relative addressing mode : addr = PC + off set

addr of mem loc to read

·) LEA : load a GPR with an addr

Ex : 1 1 1 0 | 1 0 1 | 1 1 1 1 1 1 1 0 1

R5

-3

Do : Store PC* + (-3) to R5

·) Condition codes : There are 3 single-bit reg N, Z, P that are individually set each time one of the GPRs is written

• 100 : negative data written into GPR
• 010 : zero data written into GPR
• 001 : positive data written into GPR

·) Instruction Cycle : 6 phases

• Fetch : obtain the next inst from memory and loads it into IR, increment PC

• Decode : 4-to-16 decoder, taking in opcode, output the line corp to

that opcode

- Evaluate addr : computer the addr of mem location that is needed for for the inst
- Fetch operands : Loading MAR w/ the addr calculated in "Evaluated addr"
- Execute : perform the task
- Store result

(*) Note : Not all inst use all 6 phases

) BR (Branch) ; If (condition ...) → do this inst

Format : 0 0 0 0 | n z p | PC offset

Use PC-relative

Po : If n/z/p = 1 , check the condition codes, if N/Z/P = 1, then set PC* to new addr

) JMP : Change PC* to data in the base register

Use Base + offset

- This helps reaching mem location outside of offset's range
- Format : 1 1 0 0 | 0 0 0 | 0 1 0 | 0 0 0 0 0 0

Base R: R2

offset = 0

- **Assembly** : a low level language

- **Instructions** : Label (optional) Opcode (Use name s.t ADD, LD) operands Comments (optional)

- **Labels** : symbolic names assigned to the mem loc
  - A location can contain an inst / value

- **Assembler** : Translate program in Assembly → machine language

- **Pseudo-Ops (Assembler Directives)** : A message from programmer to the assembler to help it process
  - . ORIG : tell the assembler where in the mem to place the program

    Ex : . ORIG x 3001
  - . FILL : set aside the next loc in the program and initialize it with a value / label
  - . BLKW : set aside a # sequential mem locations

    Ex : NEXT . BLKW 2
  - . STRINGZ : initialize a sequence of n+1 mem locs to store n chars (last loc contains x0000 - end-of-string char)
  - . END : end of program, don't process any after

- Assembly progress: The progress of running an Assembly program
  - 2 complete passes:
    1. Identify the addrs orpd to each symbolic labels and store them to a table (symbol table)
    2. Translate Assembly instrs into machine language

- Subroutines: functions for Assembly

- Call / Return mechanism: the user makes a call inst to code A, and after the comp execute code A, it makes a return instruction to the next inst in the program
  - Caller: the prog that contains the call inst
  - Callee: the function

- JSR(R): loads PC with the starting addr of the subroutine, and loads R7 with the addr of the inst immediately after the JSR(R) inst

  - JSR: use PC-relative addr mode

    Format: opcode | 1 | offset 11

  - JSRR: use BaseR addr mode

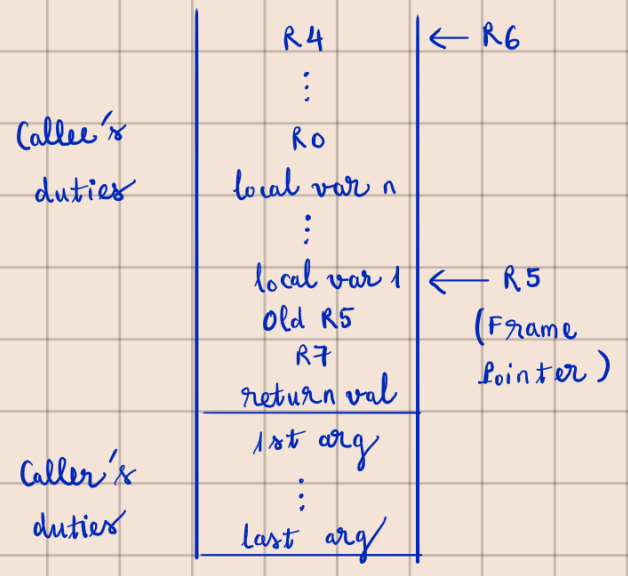
    Format: opcode | 0 | 0 0 | Base R | 0 0 0 0 0 0

- Callee Save: The subroutines have to store the existed values into memory before overwriting them

- Caller Save: The caller stores R7 into mem because it will be overwritten by JSR(R)

- Stack : last thing we stored in a stack is the 1st thing we remove from it
  - Usage for memory management : We can use stack to manage memory
    - R6 stores the addr of the top stack → Stack Pointer
    - push () : store data into memory and decrement top ≡ move top up one pos
    - pop () : take the data at top position out of stack by incrementing top ≡ move top down one pos

- Frame Pointer : A ptr used for
  - Accessing local vars from the stack
  - Saving the return value
  - Accessing arguments from the stack




- Memory - mapped IO : I/O devices and memory share the same addr space

- Synchronous ( 2 things happen at the same time)
  - Data supplied at a fixed, predictable rate
  - CPU reads / writes every X cycles ( consistently check)

- Asynchronous ( 2 things don't happen at the same time)
  - Data rate unpredictable
  - CPU must synchronize with device, so that it doesn't miss data or write too quickly ( I/O device controls the interaction )

⊛ Note : We focus on asynchronous I/O

- Data register: Used for actual transfer of data (e.g. char)

- Status register: What the device is telling the CPU

- Control register: Allow us to set changeable device characteristic

- Device register: can be memory mapped or have special I/O instrs
  - We focus on memory mapped I/O
  - Keyboard Status Reg (xFE00) (KBSR): Bit 15 is set when the keyboard has received a new char
  - Keyboard Data Reg (xFE02) (KBDR): Bits [7:0] contain the last char typed on the keyboard
  - Display Status Reg (xFE04) (DSR): Bit 15 is set when device ready to display another char on screen
  - Display Data Reg (xFE06) (DDR): Char written on bits [7:0] will be displayed on screen

- Input from keyboard: When a char is typed, KBDR and KBSR react and the keyboard is disabled. After KBDR is read, KBSR[15] is cleared and keyboard is enable

- Output to monitor: When the monitor is ready to display, DSR react. After the char is written to DDR, monitor displays the char and DSR[15] is cleared

- Purpose of OS: sharing resources and protecting users from themselves

and others

- Interupt : An unscripted subroutine call triggered by an external event
  Ex : I/O device reports a completion / error

- Types of privilege :
  - Supervisor privilege (privileged) :
    - Access to all areas in mem and can execute all instructions
    - Usually data structures or programs that are part of the operating system
  - User privilege ( unprivileged) :
    - Access to some areas in mem and can execute some instrs

- Processor State : represented by 3 items

  (1) Processor Status Reg : Privilege [15], Priority level [10:8], Condition code [2:0]
  - Doesn't really exist, the format is pushed to stack
  - Privilege = 1 : Unprivileged
    = 0 : Privileged

  (2) Program counter

  (3) General registers

- Raise Interuption : Device Status Reg set its bit 14 to raise an interupt signal

- RTI (Return from Interuption): Special inst that restores Processor State and go to the inst after the inst that raise interuption

- TRAP: An inst that call os's subroutines (privileged)

  - Format: 1111 | 0000 | trapvect 8

  - Trap vector: identify the addr of the sys call to invoke