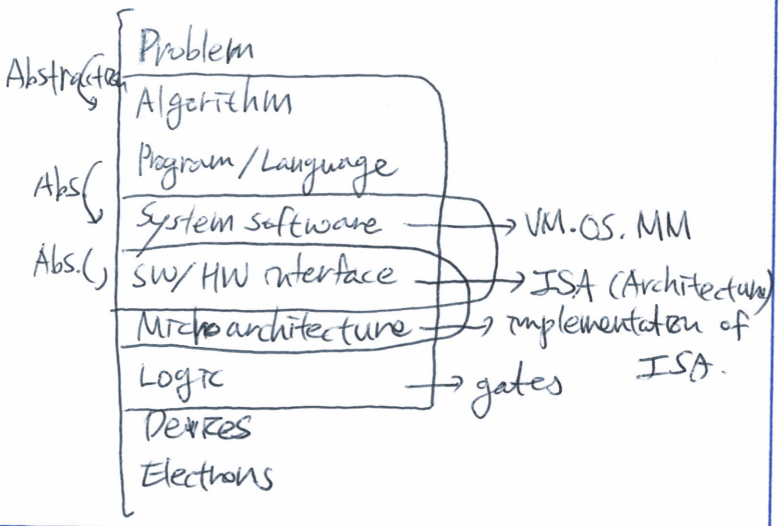
[Design of Digital Circuits Lecture]

✱ Lecture 1 · Intro & Basics, prof Onur Mutlu.



✱ Lec. 2 · Mysteries in Comp. Arch.

① Meltdown & Spectre due to speculative execution

② RowHammer : disturbance errors near rows hammered repeatedly.

✱ Lec 3. Intro. to the Labs & FPGAs

- Basys 3 (Artix-7 FPGA)

LAB1 Base Circuit → Comparison (>, <, =, !=)

LAB2 Mapping circuit to FPGA → Addition (+ ...)
⇒ 1-bit Full adder ⇒ 4-bit adder

LAB3 Display LAB2's result on a seven Seg. Display

LAB4 Finite state machine by using "memory"

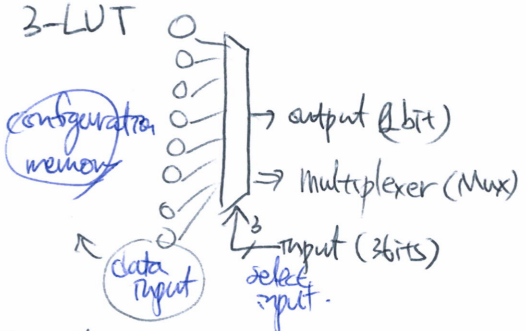
LAB5 Implementing an ALU (+ - × ÷ & OR ...

LAB6 Testing ALU → Simulate LAB5 / debug

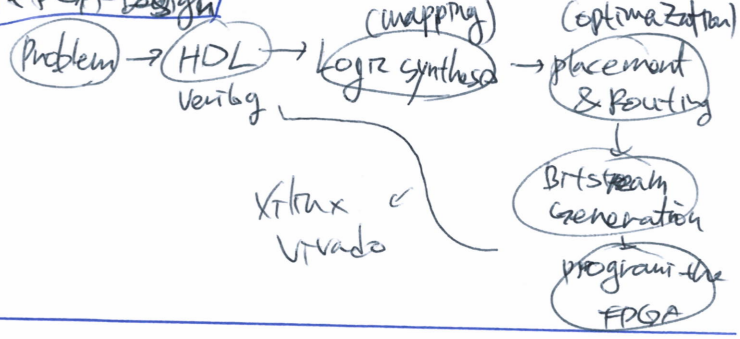
LAB7 Programming in Assembly Language. (MIPS)

LAB8 Full System Implementation (MIPS processor)

- **FPGA?**
  - ↳ Look-Up Tables & Switches. (LUT)



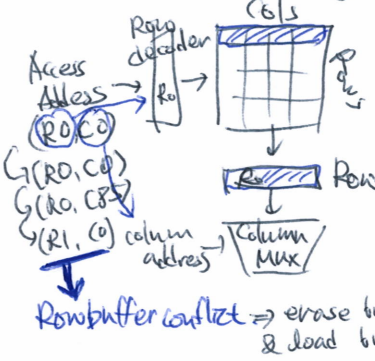
  - Typically 6-LUTs & MB-distributed memory
- **Computer-Aided Design (CAD) Tools**
  - ↳ High level description using FPGA
  - then → map the circuit & optimize the interconn.
- **FPGA Design**



✱ Lec4. Mysteries in Comp. Arch.

③ Memory performance Attack ⇒ Memory Hog
- ↳ multicore slowdown due to the Disparity
- ↳ unfairness in DRAM Memory Controller.

- DRAM bank operation → priority on app with high row buffer locality ⇒ very slow random access

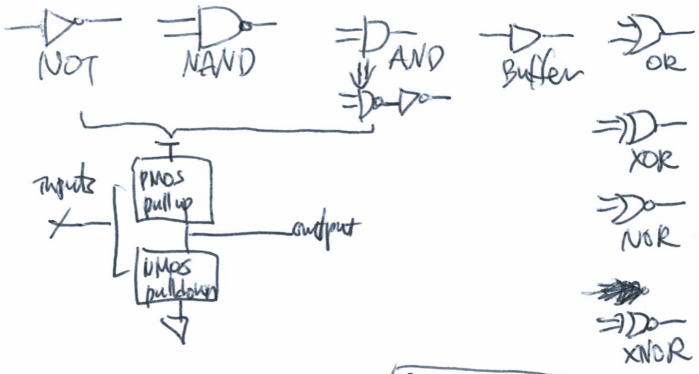


Mystery ④ DRAM Refresh.

- 64 ms! ⇒ Refresh overhead ↑ as memory size ↑
- Consider "manufacturing process variation"
- Refresh weak rows more frequently!
- Bloom filter operation & RAIDR

Lec 5. Combinational Logic.

- 64bit DP ADD ⇒ 20pJ & DRAM access ⇒ 16nJ ~x1000
- intel i-7 Broadwell-E (2016) ⇒ 3.2 billion MOS



- A Logic Circuit ⇒ input → [functional spec, timing spec] → output.
- Boolean equation ⇒ Functional spec! (no memory)

Duality (AND ⟷ OR, 1 ⟷ 0)

a·(b+c) = a+(b·c)

(useful eq.)	(dual)
X·Y + X·Ȳ = X	(X+Y)·(X+Ȳ) = X
X + X·Y = X	X·(Y+X) = X
(X+Ȳ)·Y = X·Y	(X·Ȳ)+Y = X+Y

a+(b·c) = (a+b)·(a+c)

DeMorgan's Law

- $\overline{(X+Y+Z)} = \bar{X}\cdot\bar{Y}\cdot\bar{Z}$ vice versa
- NOR = $\overline{(X+Y)} = \bar{X}\bar{Y}$ = AND with input complemented
- NAND = $\overline{(XY)} = \bar{X}+\bar{Y}$ = OR " "

- Words: (Complement, Literal, Implicant, Minterm, Maxterm)

- Truth table / Canonical Form.
- Sum of Products Form (SOP) = disjunctive normal Form, minterm expression
- Minterm form ⟷ Maxterm Form
- Combinational Building Blocks
  - Decoders + Multiplexers + Full Adder + PLA
  - log N-bit control wire.
  - PLA ⇒ programmable logic Array
- Logic simplification ⇒ Karnaugh map

MDC ②

Lec 6. Comb. Logic + Hardware Desc. Lan. (Verilog)

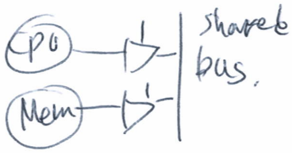
- power = CV^2f (Dynamic) + VI_{leak} (Static)
- HDL: Verilog vs UHDL
- Hierarchical Design.

TOP-DOWN: top level module → submodule, sub, sub → leaf cell (gate/cell library)

Bottom-Up: Top, sub, leaf

- Module – Name, Direction & its ports, Name of its ports
  - Then, functionality
  - Then "end module"
- multibit input/output [end : start] = (e−s+1) bits
  - ↳ Bit slicing, Concatenation, Duplication
- Behavioral VS Structural.
  - Behavioral: Define functionality (assign)
  - Structural: Instantiation & Wiring

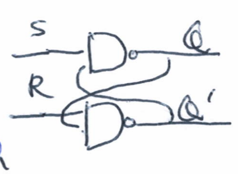
- Conditional Assignment ⇒ A ? B : C;
  (if A then B else C)
- Precedence of operation
  "~" (Not) ⇒ "*, /, %" (mult, div, mod) ⇒ "+, -" (add, sub)
  ⇒ <<, >> (shift) ⇒ <<<, >>> (arithmetic shift)
  ⇒ <, <=, >, >= (comparison) ⇒ ==, !=
  ⇒ &, ~& AND, NAND ⇒ ^, ~^ (XOR) (XNOR)
  → |, ~| (OR, NOR)
  → ?: (ternary op)
- Number N'Bxx
  bit, Base, number
- Floating signal (Z) from tri-State Buffer ☆



☆ Lecture 7. Sequential Logic Design

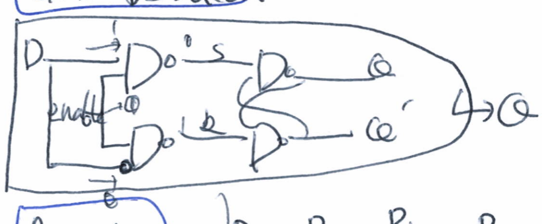
⇒ "Circuits that can store information"

- R-S Latch

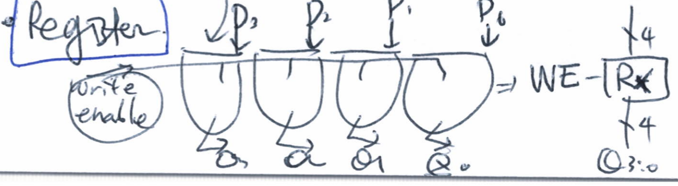


R	S	Q
1	1	Qprev
1	0	1
0	1	0
0	0	invalid

- Gated D-Latch



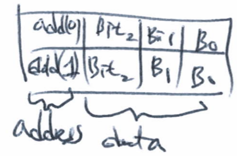
- Register



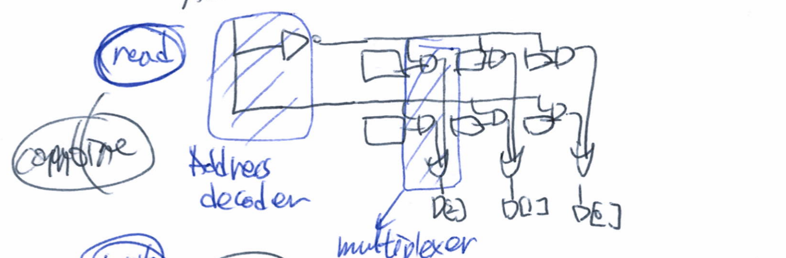
- Memory

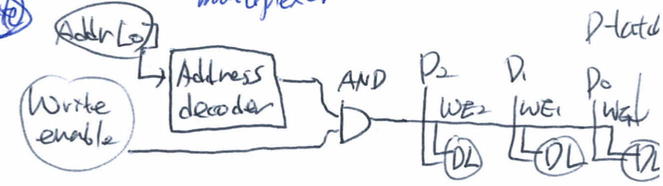
address // addressability (bits of data)

ex) address space size = 2, addressability = 3



↳ Addr [0]





- State? → Snapshot of all relevant elements of the system.
  & ⇓
  Sequence of States matters!
  ↳ example: "Clock" & clock triggers transition from one state to another & synchronize!

- Finite State Machine
  ↳ next page!

Finite State Machine → Discrete-time model of a stateful system.

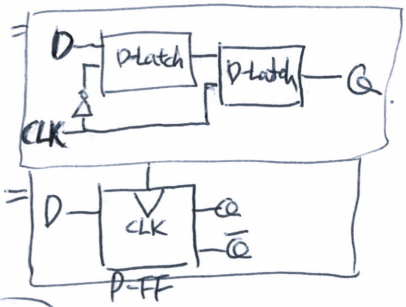
- finite: states, inputs, outputs, state transitions, output specifications

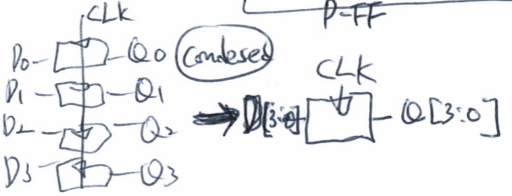
Moore FSM

★ state register ;

D-Flip Flop ≡ D-Latch + CLK

edge-triggered device (rising / falling)





Moore FSM → output = F(state)
Mealy FSM → output = F(state, input)

★ FSM state transition table

Current State	Inputs	Next state
⋮	⋮	⋮

possible State	Encoding
⋮	⋮

⇒ Another type of Truth table

∴ SOP or Boolean Logic possible to express with equation.

Current state	outputs
⋮	⋮

→ Need encoding

Encoding Typ → Fully encoded : minimize # flip flops → complex
→ 1 hot encoded : Maximize " → simple
→ output encoding

MPC ⊕

• Moore vs Mealy FSM 1101

Sequential logic w/ Verilog

(Latch ; Level of CLK & Flip-Flops ; Transition of CLK)

→ 'always' Block. → always @ (sensitivity list) statements ;

ex) D Flip-flop.

```
module flop ( input clk,
              input [3:0] d,
              output reg [3:0] q);
always @ (posedge clk)
    q <= d ;
endmodule
```

q gets d.

begin ~ end

<= Non-blocking assignment (parallel assgn)
= → blocking " (wait)

< asynchronous reset
synchronous reset.

non-blocking assignment. (No "assign")

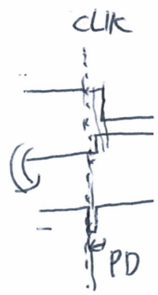
Lecture 8. Timing and Verification

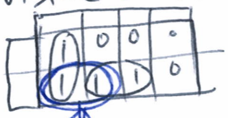
⇒ Trade offs in Circuit Design
Area / Speed (Throughput) / Power / Design Time

Part I. Combinational Circuit Timing

* Delays due to C & R
  - contamination delay (min)
  - propagation delay (max)
- Care the Longest Path in the circuit.
- Dependence on Voltage & Cap
- combination of gate delays...
- wire !!

* output Glitches
- XOR of Slow/Fast inputs
- visible in K-maps

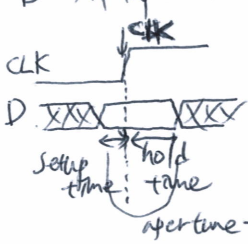




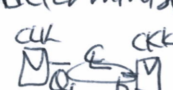
Add this comb. to ensure no transition happening

Part II. Sequential Circuit Timing

- D-flipflop: 'D' must be stable when sampled.
  → if changing at CLK edge → metastability & nondeterministic



- Sequencing overhead



⇒ $T_c >= t_{pcq} + t_{pd} + t_{setup}$
(wasted work)

$t_{hold} < t_{ccq} + t_{cd}$, $t_{cd} > t_{hold} - t_{ccq}$

- t_{pd} ⇒ delay in the longest path
- t_{cd} ⇒ " shortest "
- t_{setup} ⇒ stable time before clk edge
- t_{hold} ⇒ " after "

setup time constraint
* $T_c \geq t_{pcq} + t_{pd} + t_{setup}$

hold time constraint
$t_{hold} \leq t_{ccq} + t_{cd}$
$t_{cd} \geq t_{hold} - t_{ccq}$

t_{ccq} : Contamination delay clk to q
t_{pcq} : propagation delay clk to q

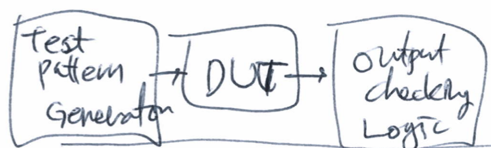
$(T_c)^{-1} = f_{max}$; maximum frequency

if hold time constraint fail (too fast)
⇒ Add buffers to make it slow

- Clock Skew ⇒ smart "clock network"
  - $t_c >= t_{pcq} + t_{pd} + t_{setup} + t_{skew}$
  - $t_{cd} >= t_{ccq} + t_{hold} + t_{skew}$

Part 3. Verification

- Functionality ? Timing ? ⇒ Simulation Tools (SAT solver, Vivado, spice)
- Test bench




```
module example_testbench(); → no input, output
  reg a; → manually assigned
  wire y; → manually checked
  initial
  begin
    a=0;
    #10 ; → wait 10ns
    a=1 ;
    $display ("printf() style message!")
  end
```

Instantiate device under test
DUTname dut (.a(a), .b(b), .c(c), .y(y));

- Simple Testbench (look clk sequence)
- Self checking testbench w/ test vector (look display after checking outputs)
- Automatic testbench
  $readmemb ("example.tv", testvectors);

Lecture 9. Von Neumann Model, ISA, LC3, & MIPS

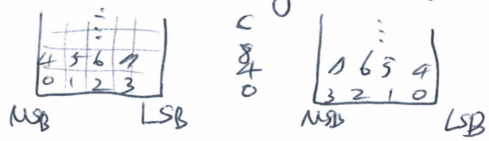
(Combinational logic / Sequential logic) ⇒ (Decision element / Storage element)

* Von Neumann → [Memory, Processing Unit, Input, Output, Control Unit]
  - LC-3
  - MIPS

* Von Neumann model: Input → Memory (address) (data) → Output; Processing (ALU, TEMP); control unit; PC or IP; Inst Reg

* memory (byte = 8bit)
  - words = 8, 16, 32 bits
  - addressability, address space
    - LC-3 → 2^{16}
    - MIPS → 32 address → 4GB
  - Word Addressable memory
    - data word
    - LC-3: 16bit - dat word
    - MIPS: 32 "
  - Byte Addressable memory
    - MIPS: Byte Addressable (4 byte = 1 word)
    - LC3b:

- Little Endians vs Big Endians



Memory Access — (Reading = Loading; Writing = Storing)

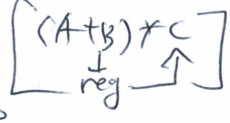
two reg → MAR (address), MDR (data)

Read → MAR Load → MDR Load

Write → MAR Load → "write Enable" signal activate. MDR Load

* Processing Unit
  - LC-3: ADD, AND, NOT (XOR - LC3b)
  - MIPS: add, sub, mult, and, nor, sll, slr, slt, ...
  - Word length: LC-3 (16) MIPS (32)

* temporary storage = Registers.



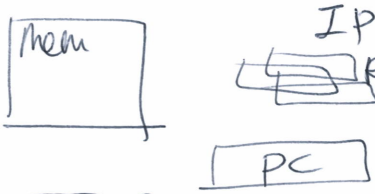
- Memory big/slow
- Register fast / one reg → store one word
  - Register Set ⇒ (LC3 = 8 general purpose registers; MIPS = 32 registers. (GPR) (size = 32 bits))

* MIPS Reg File

0	1	2-3	4-7	8-15	16-23
$0	$at	$Vx	$ax	$tx	$Sx

24-25	26-27	28	29	30	31
$tx	$Kx	$gp	$sp	$fp	$ra

* Control Units → IR (Instruction register)
  PC (program counter)
  IP (Instruction pointer)

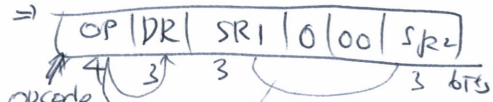


LC3 : Von Neuman Machine.

- ISA (Instruction Set Architecture)
- Assemby → Machine code.

(add a, b, c)

LC3 ⇒



Load a, A, i

destination operand, base address, offset

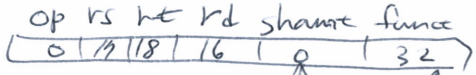
LC3 → LDR R3, R0, #2

MIPS → lw $s3, 2($s0)

base address

$s3 ← Memory [$s0 + 2]

MIPS →



Instruction cycle

- Fetch
- Decode
- Evaluate Address
- Fetch Operands
- Execute
- Store Result

(LDR → no execution
ADD → no Evaluate ADDRESS)

MDC ⑥

① Fetch : PC → MAR → MDR (data = Instruction) memory → IR (control unit)
PC+1 → PC

② Decode : (LC3) 4-to-16 decoder (16 opcodes)

IR[15:12] → Decoder → (control unit)

③ Evaluate ADDRESS : case of LDR → IR (control) + SR1out (processing unit register) → (Adder) → processor bus

④ Fetch Operands : MAR → MDR → register

In case of ADD

↳ Instruction decoding & Fetch operands concur.

⑤ execute ; ALU & Reg

⑥ Store result ; (LDR) MDR → DR

Changing the Sequence of Execution

↳ ① change PC by loading it during execute phase

② then, it wipes out incremented PC (loaded during Fetch phase)

Jump in LC-3 - JMP R2 (1100 | 000 | BaseR)

⇒ PC ← R2 (stores Base Register (Register identify by BaseR))

- RET :
- JSR, JSRR

MIPS - "j target" (J-type) [2 | target] (6bits opcode, 26bits)

$PC \leftarrow PC^{+}[31:28] \mid \text{sign-extend(target)} * 4$

(PC[31:28]: 4 most significant bit; sign-extend(target): 26 bit; * 4: 2bit shifting)

jal (jump & link)

jr (jump register)

---

① Opcodes → HP Precision Architecture
x86
VAX

↳ 3 types - Operate / Data movement / Control

(MIPS) ⇒ R-type

↳ Addressing Mode → 5 addressing mode (LC-3)

- Immediate or literal
- Register
- memory addressing mode
  - PC-relative
  - Indirect
  - Base-offset

(MIPS) + pseudo-direct addressing (J & jal)

LC-3 → NOT : unary operation
ADD, AND : binary operations

MIPS →
- R type (binary operations)
- I type
- F-type (floating point operation)

• no NOT in MIPS, use NOR.

Lecture 9. ISA (LC-3, MIPS) & Assembly Programming.

• Op Instr. with One Literal in LC-3

└ Add One Literal in LC-3, MIPS

(mips) addi $50, $51, 5

• Data movement Instr. & Addressing mode

- PC-relative addressing mode.

LD (load) & ST (store) (load data)

OP	DR/SR	PCoffset9
4bit	3bit	9bit

- Indirect addressing mode (load address & load data from that address)
- Base-offset addressing mode
- Immediate addressing mode