Lecture 04

Sequential Circuits & Computer Architectures

Flip-Flops · Registers · FSMs · CISC vs RISC · Von Neumann vs Harvard

SENG 21213 · Computer Architecture & Operating Systems

Learning Objectives — By the end of this lecture you should be able to:

- 1. Describe SR latch, D flip-flop, and JK flip-flop behaviour from their characteristic tables
- 2. Design synchronous registers, shift registers, and binary counters using T flip-flops
- 3. Follow the 7-step FSM design procedure and distinguish Moore from Mealy machines
- 4. Compare CISC and RISC architectures on 10 key attributes
- 5. Explain Von Neumann vs. Harvard architectures and how modern CPUs use a Modified Harvard design

Section 1 · Sequential Logic — Flip-Flops and Registers

📖 **Stallings Reference**

- Appendix B: Digital Logic — Sequential Circuits
- Flip-flops, registers, and counters are the building blocks of the CPU register file and memory

Sequential circuits differ from combinational circuits in one critical way: their output depends on both the current inputs AND the history of past inputs, stored in memory elements called flip-flops. The clock signal synchronises state changes.

1.1 SR Latch — The Most Basic Memory Cell

S	R	Q (next)	Q' (next)	State
0	0	Q (unchanged)	Q' (unchanged)	Hold — no change
1	0	1	0	SET — force Q=1
0	1	0	1	RESET — force Q=0
1	1	Invalid	Invalid	Forbidden — both outputs identical

1.2 D Flip-Flop (Data Flip-Flop)

The D flip-flop is the standard building block of CPU registers, pipeline registers, and digital state machines. It has a single data input D and a clock CLK.

CLK Edge	D	Q (after edge)	Behaviour
Rising edge	0	0	Q captures D on the rising clock edge
Rising edge	1	1	Q captures D on the rising clock edge
Between edges	X	Q unchanged	Q holds its value until the next clock edge

★ Why D Flip-Flops?

- Setup time (t_su): D must be stable for t_su before the clock edge. Violation → metastability.
- Hold time (t_h): D must stay stable for t_h after the clock edge.
- Propagation delay (t_pd): Output Q changes within t_pd after the clock edge.
- Maximum clock frequency: f_max = 1 / (t_pd + t_su + routing delay). Exceeding f_max causes timing violations.

1.3 JK Flip-Flop

J	K	Q (next)	Operation
0	0	Q	Hold — no change
1	0	1	Set — Q becomes 1
0	1	0	Reset — Q becomes 0
1	1	Q'	Toggle — Q flips (eliminates forbidden state of SR)

1.4 Registers and Shift Registers

Type	Structure	Operation	Use Case
Parallel Register (n-bit)	n D flip-flops sharing one clock	All n bits load simultaneously on clock edge	CPU general-purpose registers, ALU operand buffering
Serial-in Parallel-out (SIPO)	n flip-flops chained; D_{n} = output of FF_{n-1}	One bit shifts in per cycle; after n cycles all bits available	Converting serial data (UART) to parallel bus
Parallel-in Serial-out (PISO)	Load logic + chain	Load all n bits in one cycle; shift out one per cycle	Converting parallel bus to serial transmission
Bidirectional	MUX on each D input selects shift-left or shift-right	Shifts left or right by one per cycle	Arithmetic shift for multiply/divide by 2

1.5 Binary Counters

A synchronous binary up-counter with n T flip-flops counts from 0 to 2^n-1. The toggle enable for each flip-flop:

★ Counter Toggle Equations (T Flip-Flop)

- $T_0 = 1$ (always toggles — LSB flips every cycle)
- $T_1 = Q_0$ (toggles when $Q_0 = 1$)
- $T_2 = Q_1 \cdot Q_0$ (toggles when $Q_1 = Q_0 = 1$)
- T_k = Q_{k-1} · Q_{k-2} · … · Q_0 (toggles when all lower bits = 1)
- Synchronous: all flip-flops share the same clock edge — no cumulative delay.
- Asynchronous (ripple): Q_0 clocks FF_1, Q_1 clocks FF_2, etc. — delays add up; timing issues at high frequencies.

Section 2 · Finite State Machines

2.1 Moore vs. Mealy Machines

Property	Moore Machine	Mealy Machine
Output depends on	Current state only	Current state AND current input
Output changes when	State changes (on clock edge)	Input changes (asynchronously)
Stability	Glitch-free outputs	May have momentary glitches on input change
Number of states	Typically more states needed	Fewer states for same function
Diagram notation	Output written inside state circles	Output written on transition arrows
Real-world use	Traffic lights, vending machines	Handshaking protocols, data parsers

★ FSM Design Procedure

- Step 1: Define what the system needs to REMEMBER — these are the states.
- Step 2: Identify all inputs that cause transitions and all outputs produced.
- Step 3: Draw the state transition diagram (STD): circles = states, arrows = transitions labelled with input/output.
- Step 4: Encode states as binary numbers (n flip-flops for 2^n states).
- Step 5: Build the state transition table — for each (current state, input) pair, what is (next state, output)?
- Step 6: Derive next-state logic equations using Boolean algebra or Karnaugh maps.
- Step 7: Implement with flip-flops + combinational logic (AND/OR/NOT gates or multiplexers).

2.2 Worked FSM Example — Traffic Light Controller

A 4-state traffic light cycles: GREEN (30 s) → YELLOW (5 s) → RED (30 s) → YELLOW (5 s) → GREEN... A timer signal T goes high when the current state's time has elapsed.

State	Encoding (Q1 Q0)	Timer Input T	Next State	Lights
GREEN	00	0	GREEN (hold)	G=1, Y=0, R=0

State	Encoding (Q1 Q0)	Timer Input T	Next State	Lights
GREEN	00	1	$YELLOW_1$	G=1, Y=0, R=0
$YELLOW_1$	01	0	$YELLOW_1$ (hold)	G=0, Y=1, R=0
$YELLOW_1$	01	1	RED	G=0, Y=1, R=0
RED	10	0	RED (hold)	G=0, Y=0, R=1
RED	10	1	$YELLOW_2$	G=0, Y=0, R=1
$YELLOW_2$	11	0	$YELLOW_2$ (hold)	G=0, Y=1, R=0
$YELLOW_2$	11	1	GREEN	G=0, Y=1, R=0

Section 3 · CISC vs. RISC — Computer Architecture Philosophies

📖 Stallings Reference

- Chapter 13: Reduced Instruction Set Computers (RISC)
- Chapter 14: Instruction Level Parallelism and Superscalar Processors

Attribute	CISC	RISC
Instruction count	Small — many operations in one instruction	Large — simple ops require more instructions
Instruction length	Variable (x86: 1–15 bytes)	Fixed (MIPS, ARM: 4 bytes; RISC-V: 4 or 2)
Addressing modes	Many (10–20+)	Few (3–5); Load/Store only for memory
Register count	Few special registers (AX, BX, CX, DX)	Many general-purpose (32 on MIPS/ARM)
Memory access	Most instructions can access memory	Only LOAD and STORE access memory
CPI	Variable; often >1	Close to 1 (designed for pipelining)
Control unit	Microprogrammed (complex decoder)	Hardwired (simple, fast)
Compilation	Dense code; less register pressure	More instructions; compiler does more work
Examples	x86, x86-64, VAX	MIPS, ARM, RISC-V, SPARC, PowerPC
Pipeline support	Difficult — variable instruction length	Designed for pipelining from the start

★ Von Neumann vs. Harvard

- Von Neumann: Instructions and data share ONE memory space and ONE bus. Bottleneck: instruction fetch and data access compete for the same bus.
- Harvard: Separate instruction memory (ROM/flash) and data memory (RAM) with separate buses.

CPU can fetch next instruction while reading/writing data — no contention.
- Modified Harvard (modern CPUs): physically one main memory, but separate L1-I (instruction cache) and L1-D (data cache) at the top level — eliminates the bus contention for most accesses.
- Your bootloader lives in Flash (instruction side) and uses RAM for stack and variables — a real modified-Harvard system.

Section 4 · Practice Exercises

✎ Exercise 4.1 [12 marks] — Flip-Flop Timing

- (a) [4] A D flip-flop has t_pd = 3 ns, t_su = 1 ns, t_h = 0.5 ns. The combinational logic between two flip-flops has a delay of 5 ns. What is the maximum clock frequency? Show the critical path calculation.
- (b) [4] A 4-bit synchronous binary counter uses T flip-flops. Write the Boolean expression for T3, T2, T1, T0. Draw the timing diagram for 10 clock cycles starting from state 0000, showing all four Q outputs.
- (c) [4] An 8-bit SIPO shift register holds the value 11010101. On each of the next three clock cycles, bits 0, 1, 1 are shifted in serially (LSB first). What is the register content after 3 cycles? Show the shift operation step by step.

✎ Exercise 4.2 [14 marks] — FSM Design

- (a) [6] Design a Mealy FSM for a serial bit-stream detector that outputs 1 whenever it detects the pattern '101'. The circuit has one input (one bit per cycle) and one output. Draw the complete state transition diagram, encode states, and build the state transition table.
- (b) [4] Using the traffic light FSM from Section 2.2, derive the next-state equations D_1 and D_0 for the two D flip-flops. Use Q_1, Q_0 and T as inputs. Minimise using Boolean algebra.
- (c) [4] Convert your Mealy FSM from (a) to a Moore FSM. How many states does the Moore version require? Explain why Moore typically needs more states than Mealy for the same function.

✎ Exercise 4.3 [9 marks] — CISC vs. RISC

- (a) [3] The C statement a = b + c is compiled. On x86-64 (CISC): show 3 possible instruction sequences of decreasing length (from 3 instructions to 1). On MIPS (RISC): show the necessary 3-instruction sequence. Explain why MIPS always requires separate LOAD and STORE.
- (b) [3] Explain the 'compiler does more work' trade-off in RISC architectures. What does a RISC compiler need to do with registers that a CISC compiler does not? Name two compiler techniques specific to RISC.
- (c) [3] A benchmark runs 10^9 RISC instructions (each 1 cycle) at 2 GHz vs. 4×10^8 CISC instructions (each 2.5 cycles on average) at 1.5 GHz. Calculate execution time for each. Which is faster? What metric is misleading for this comparison?