Resumo para a Avaliação Final

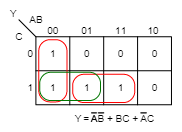
Design de Computadores

**Tempo de Atraso de Propagação:** Tempo entre a mudança da entrada e a estabilização da saída.

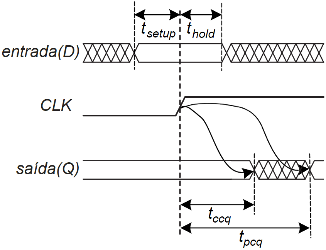
**Tempo de Contaminação:** Tempo entre a mudança da entrada e o início da mudança na saída.

**Glitch:** Conforme o caminho mais curto propaga antes do caminho longo, o tempo até o mais longo estabilizar gera ruido na saída.

O tempo de contaminação resultante é a soma das contaminações do caminho mais longo; uma forma de minimizar os Glitches é utilizar circuitos completos:



**Temporização de circuitos sequenciais**: Setup time é a janela de estabilidade de tempo antes da borda de clock; e Hold time a janela após. Neste caso, o tempo de contaminação é o quanto o circuito demora para começar a mudar de estado, e o tempo de propagação é o quanto leva para estabilizar após a borda de clock. A violação desta dinâmica é a metaestabilidade.



**R: Instrução** rs+rt=rd **assembly** rd=rs+rt.

**I**: **LW** rt = Imm + R[rs] **SW** M[Imm + R[rs]] = rt **BEQ** R[rt] == R[rt] ? PC += imm + 4 : 4.

J: PC = (PC+4)[31..28] ⊔ Imm[25..0] ⊔ “00”.

**Pipeline**: Instruction Fetch (IF); Instruction Decode (ID); Execute (EX); Memory Acess (MEM); Write Back (WB).

**Hazzard estrutural**: Recurso de hardware utilizado múltiplas vezes no mesmo clock, sendo que não aceita acessos concomitantes.

**Hazzard de dados**: instrução executada depende do resultado de outra que está no pipeline e não terminou. **Solução**: Fowarding/Bypass - criação de um desvio interno ao pipeline.

**Hazzard de controle**: Devido um desvio na sequência algumas instruções no pipeline não devem executar.

**Pontos de criação de dados**: Saída da ULA e Leitura da memória de dados.

**Pontos de consumo de dados**: Entradas da ULA (as duas).

**Problemas com o BEQ**: Desperdiça 3 clocks; instruções em sequência não devem ser executadas. **Solução**: previsão do desvio, decisão postergada/preencher instruções ou NOP.

Problemas com o J: Desperdiça 1 clock; **Solução**: decisão postergada/preencher instruções ou NOP.

Entre uma instrução LW e outra do Tipo R, consecutivas, em que o LW escreve em um registrador que será lido pela instrução R, deve haver, sempre, uma instrução NOP (ou uma bolha) ou uma instrução que não utilize o mesmo registrador.

O circuito de *stall* é responsável pela inativação do *pipeline* durante a ocorrência de um *hazard* de dependência de dados onde o *forwarding* não é suficiente para resolver.

Devido à característica, do banco de registradores, de leitura e escrita em um único ciclo, podemos eliminar uma bolha (ou NOP) nas dependências do tipo RAW.

Para a execução, no Pipeline Simples, deve-se adicionar os seguintes NOPS: Três depois do BEQ; Um depois do Jump; Dois depois de cada instrução com dependência de dados.A contagem de ciclos de um programa é dada por: