Reminder

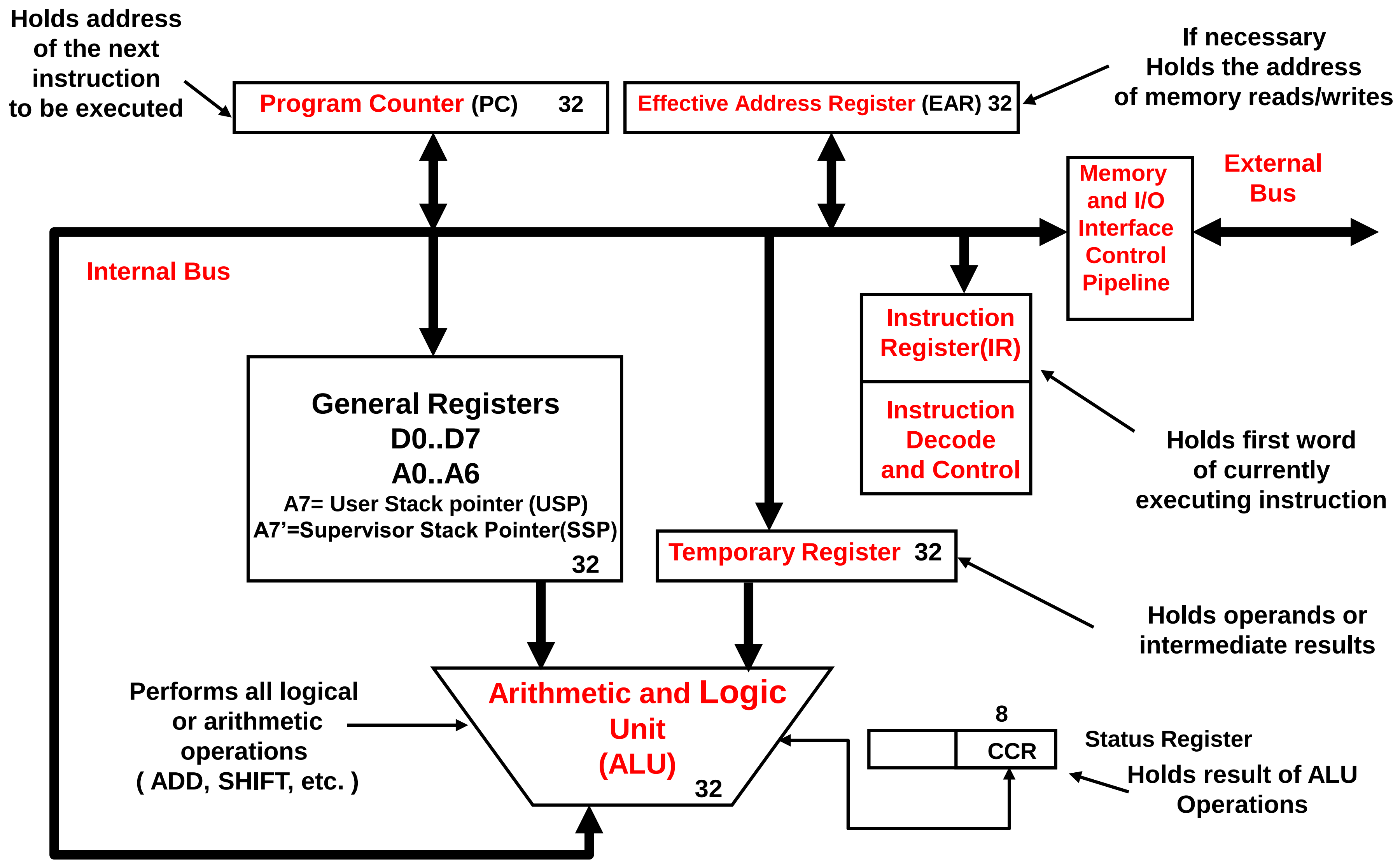
- Course Evaluation
  - https://uwb.iasystem.org/survey/19213

- Course Project
  - https://canvas.uw.edu/courses/1232689/assignments/4397073
  - Presentation and Demo
    - https://canvas.uw.edu/courses/1232689/assignments/4397074
    - Peer evaluation form
  - Due on Thursday December 6th 11:50pm

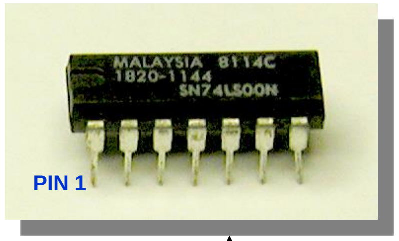
- Final Exam
  - https://canvas.uw.edu/courses/1232689/assignments/4397072
  - All homework and exercise solutions on Canvas
  - Final exam is on Tuesday December 11th 8 pm – 10 pm

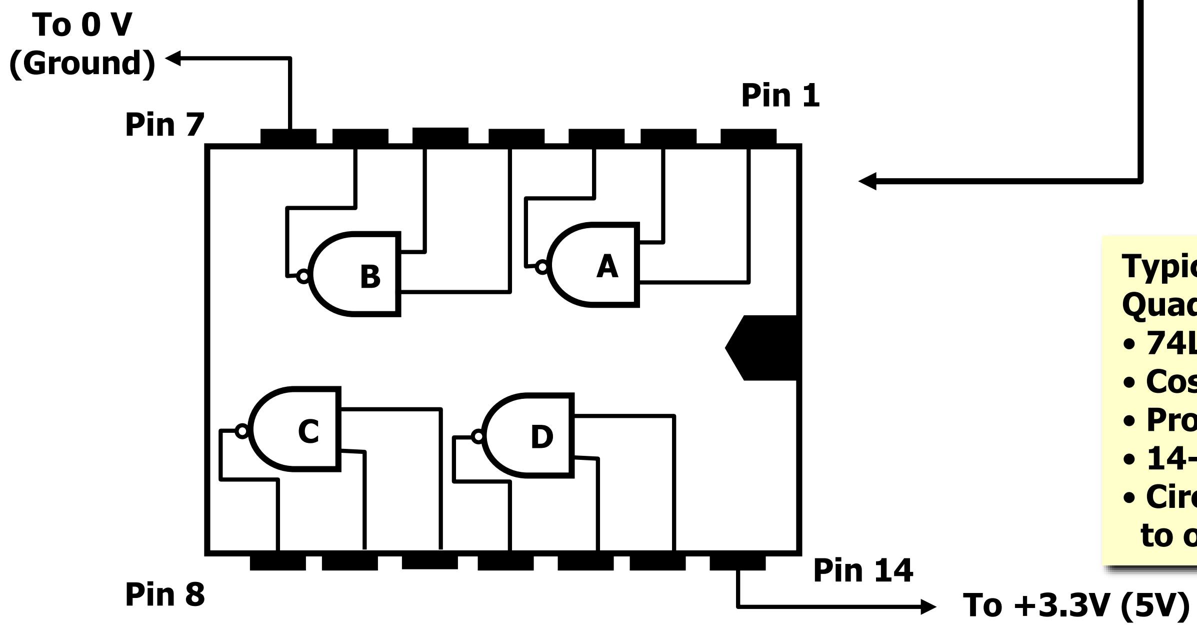
FINAL REVIEW

Hardware Organization of the MC68000


Holds address of the next instruction to be executed
Program Counter (PC) 32
Effective Address Register (EAR) 32
If necessary Holds the address of memory reads/writes
Memory and I/O Interface Control Pipeline
External Bus
Internal Bus
Instruction Register(IR)
Instruction Decode and Control
Holds first word of currently executing instruction
General Registers
D0..D7
A0..A6
A7= User Stack pointer (USP)
A7'=Supervisor Stack Pointer(SSP)
32
Temporary Register 32
Holds operands or intermediate results
Performs all logical or arithmetic operations (ADD, SHIFT, etc.)
Arithmetic and Logic Unit (ALU)
32
8
CCR
Status Register
Holds result of ALU Operations

Circuit



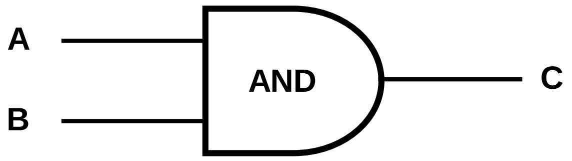


Typical NAND gate circuit
Quad, 2-input NAND gate

- **74LS00**
- **Cost: ~ $0.10**
- **Propagation delay ~ 5 nsec**
- **14-pin Dual in-line package (DIP)**
- **Circuits vary from *8* pin DIP packages to over *600* pin high-density packages**

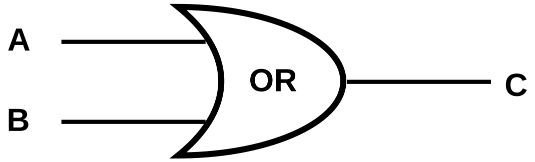
Logical Gates

- The ***Gate*** is the basic element of all digital systems
  - Three types of gates form the “atomic” elements



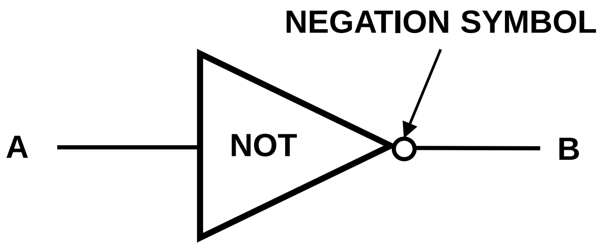
C = A•B

C is *TRUE* if A is *TRUE* AND B is *TRUE*



C = A + B

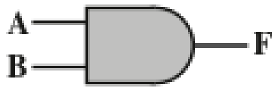
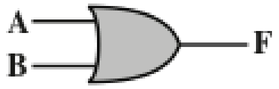
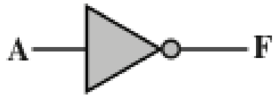
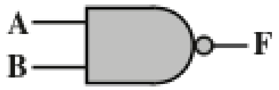
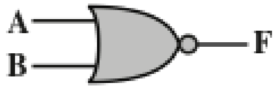
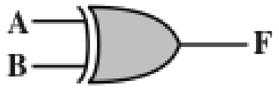
C is *TRUE* if A is *TRUE* OR B is *TRUE*



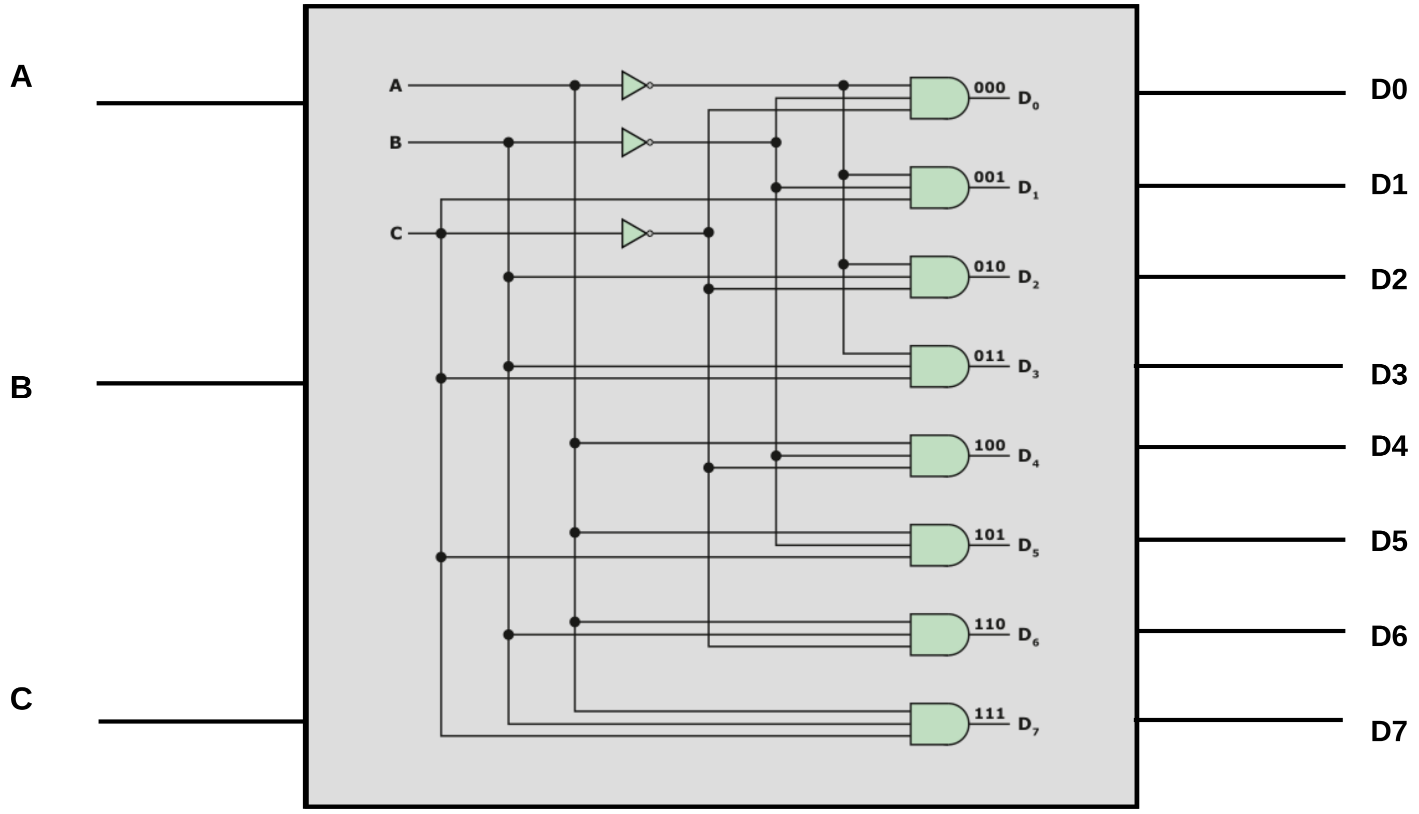
$B = \overline{A}$

B is *TRUE* if A is *FALSE*

Logic and Gate

Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \bullet B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND		$F = \overline{AB}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{A + B}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		$F = A \oplus B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

Design a Combinational Circuit
3 x 8 Line Decoder



3 X 8 line decoder

Truth Table vs. Boolean Equation

abc = 000 → D0 =1
abc = 001 → D1 =1
abc = 010 → D2= 1
abc = 011 → D3= 1
abc = 100 → D4 =1
abc = 101 → D5 =1
abc = 110 → D6= 1
abc = 111 → D7= 1

a	b	c	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Simplification using K-Maps

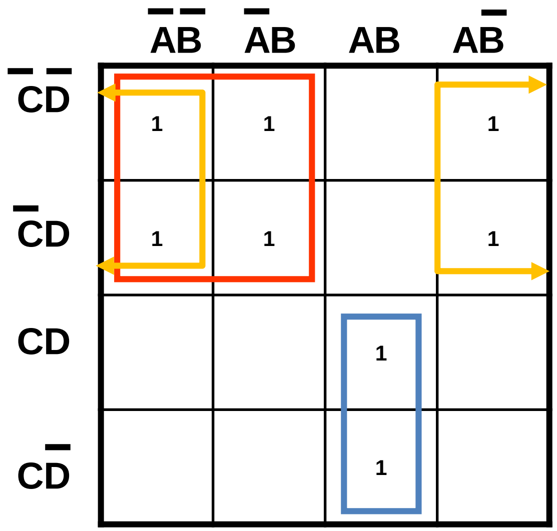
1. Construct **one Karnaugh Map for each output variable**
2. Place a “1” in every cell that has a 1 in the corresponding row of the truth table
3. Form **the largest possible loops** of cells containing 1, 2, 4, 8,16, etc., ***adjacent*** “1” terms
4. **Any cell can be involved in any number of loops**, but **each new loop must contain at least one entry that is not contained in any other loop**, in order to avoid a redundant loop
   - **“loop within loop” is NOT ALLOWED!**
5. Inspect the map for any loops whose terms are all enclosed in other loops and remove those loops
6. Each loop represents a simplified minterm (sum of product) of the logic equation
   - Simplify the loop by **removing any variable that appears in its complemented and un-complemented form**

Karnaugh Maps

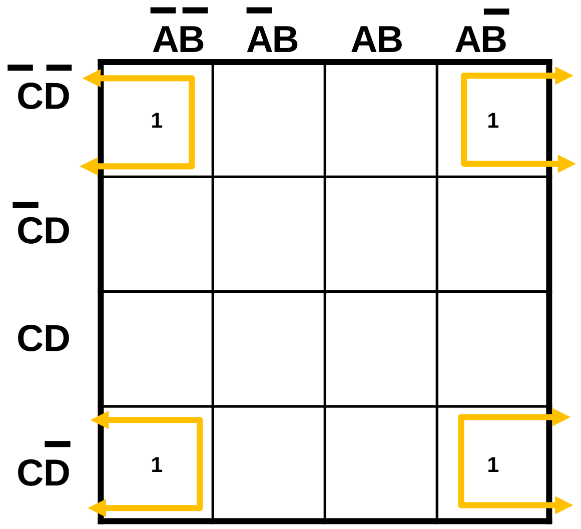
- We want to systematically derive a Boolean equation from a truth table
- The Karnaugh Map (pronounced "car know") is a graphical method of simplifying the "Sum of Products" (minterm) truth table
- **Based upon Boolean algebra simplification** $A*B + A*\overline{B} = A$
  - Why?
    - First Distributive Law: $A*(B + C) = (A*B) + (A*C)$
    - Third Law of Complementation: $B + \overline{B} = 1$
    - Finally: $A*1 = A$
- Therefore

  $A*B + A*\overline{B} = A*(B + \overline{B}) = A*1 = A$
- http://sourceforge.net/projects/k-map/files/k-map/0.4/Kmap-04-setup.exe/download

K-Map Examples



This K-Map has three loops. Notice that the yellow loop is actually adjacent.
The equation is **X = ~A*~C + ~B*~C + A*B*C**



This K-Map first appears to have two loops. The diagonal are not adjacent,
but by first rotating the map around the vertical axis and then around a
horizontal axis, we can cluster the four terms into a single 2 by 2 loop.

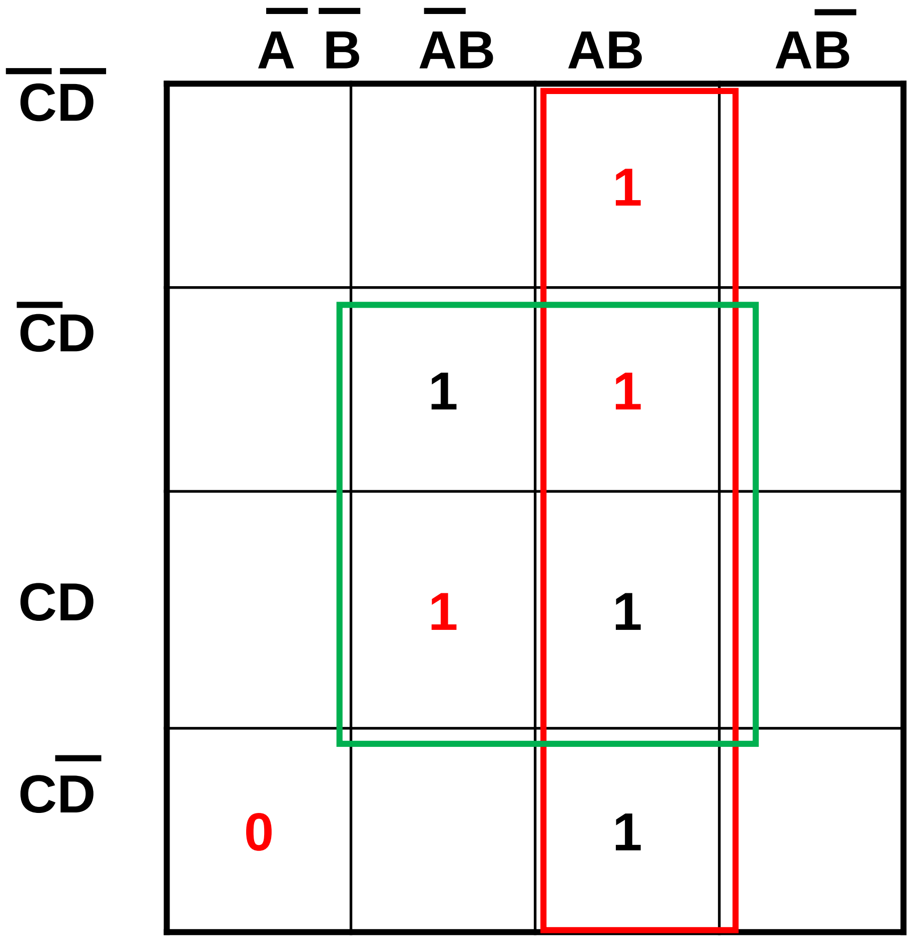
The equation is **X = ~B*~D**

K-Map with “don’t care” Condition

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$			x	
$\bar{C}D$		1	x	
CD		x	1	
$C\bar{D}$	x		1	

- X can be 0 or 1
- Choose so the expression can be as simple as possible.
- What’s the Boolean equation?

K-Map with “don’t care” Condition



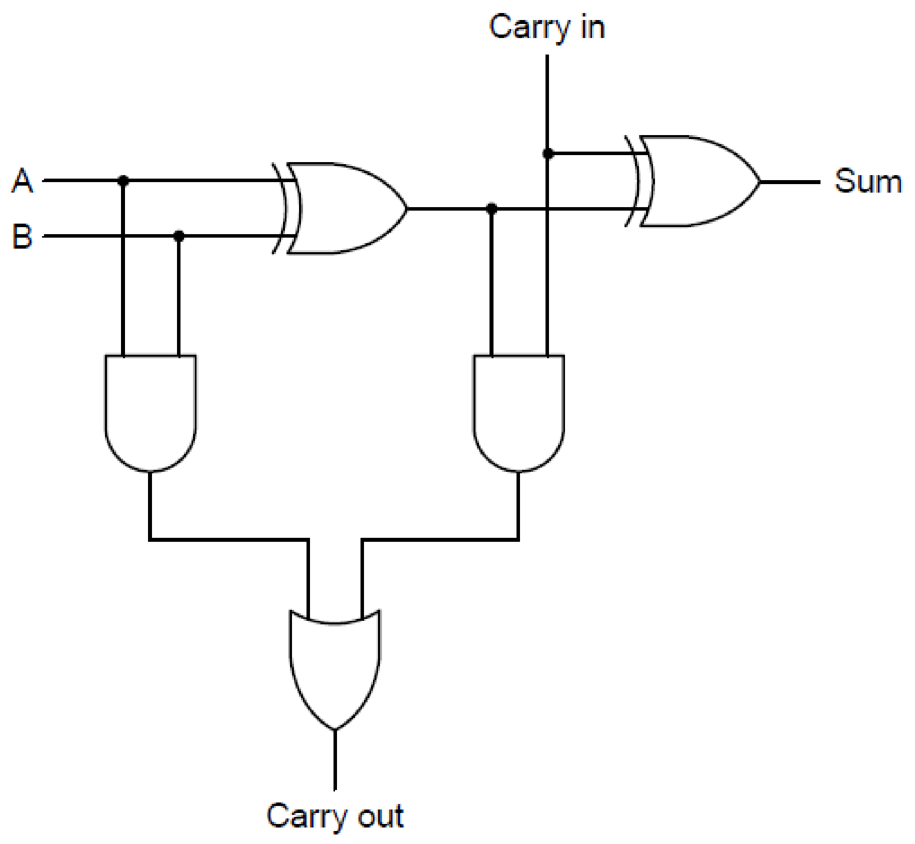
	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$			1	
$\bar{C}D$		1	1	
CD		1	1	
$C\bar{D}$	0		1	

- X can be 0 or 1
- Choose so the expression can be as simple as possible.
- What’s the Boolean equation?

Arithmetic Circuits – cont'd

A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)

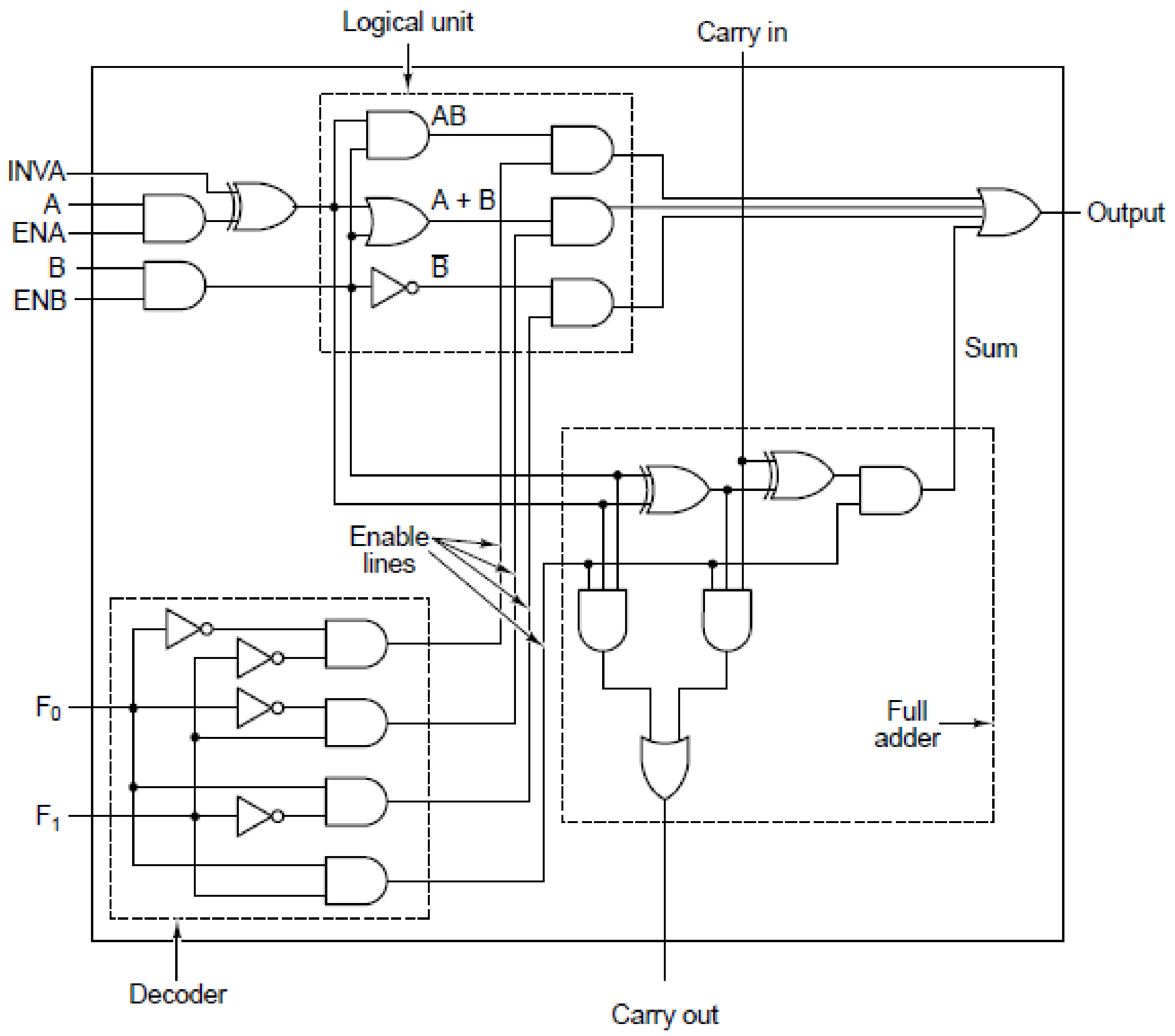


(b)

(a) Truth table for a full-adder.

(b) Logic diagram for a full-adder.

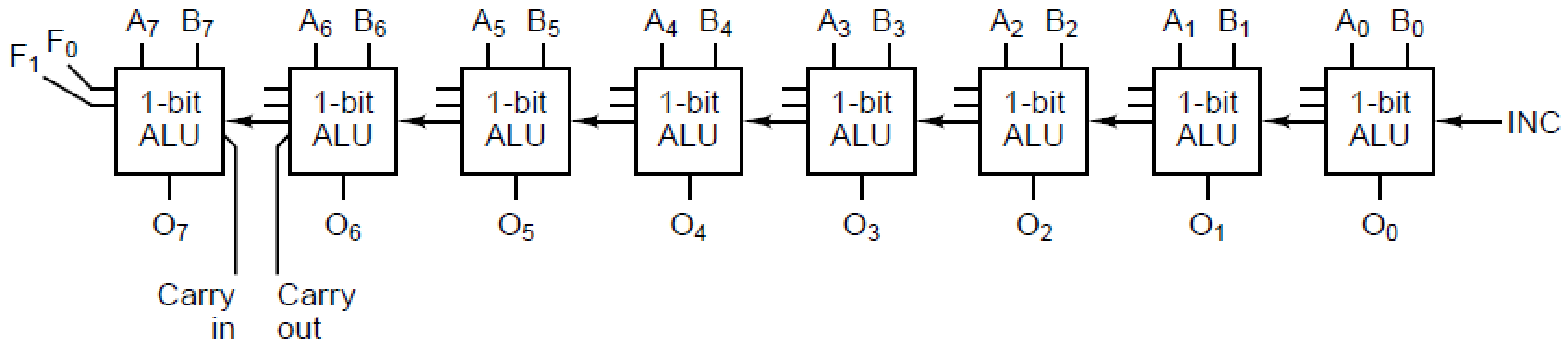
ALU: Arithmetic Logic Units



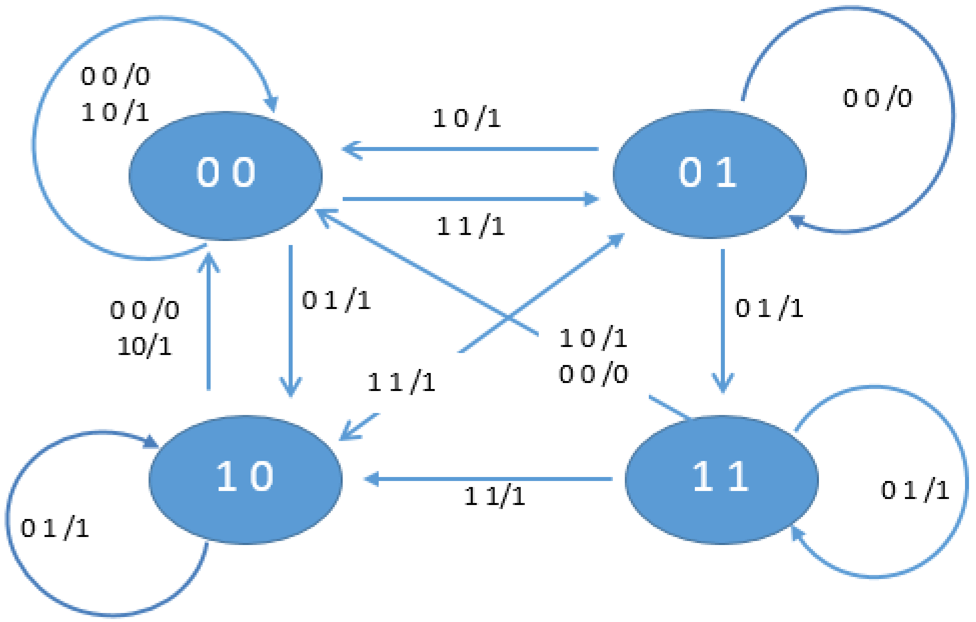
A 1-bit ALU

From 1-bit ALU to 8-bit ALU

Eight 1-bit ALU slices connected to make an 8-bit ALU. The enables and invert signals are not shown for simplicity.

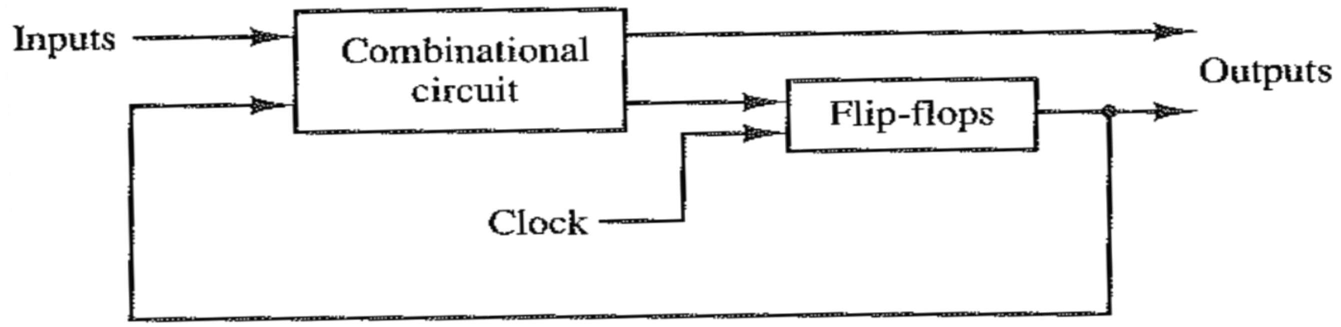


How to design a state machine?



Sequential Circuit Design

- A sequential circuit can be a combination of combinational circuits and flip-flops



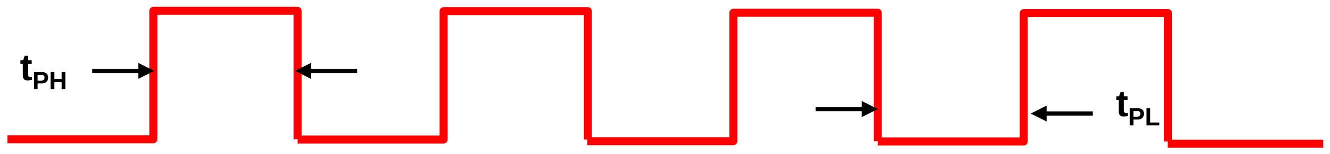
Block diagram of a sequential circuit

- Because the sequential circuit involves state transition, we have different types of tables for this circuit

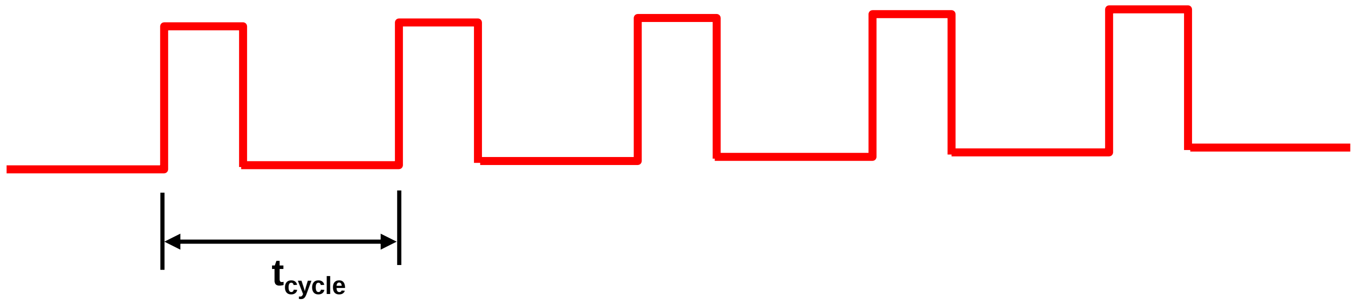
- **State Table**: Based on the **input and current state**, give **the next state** information (like truth table)
- **Characteristic Table**: Give the state transition information based on inputs
- **Excitation Table**: Inputs are the current and next states, and outputs are input signals

Clocks and Pulses

- Clocks are continuous streams of pulses

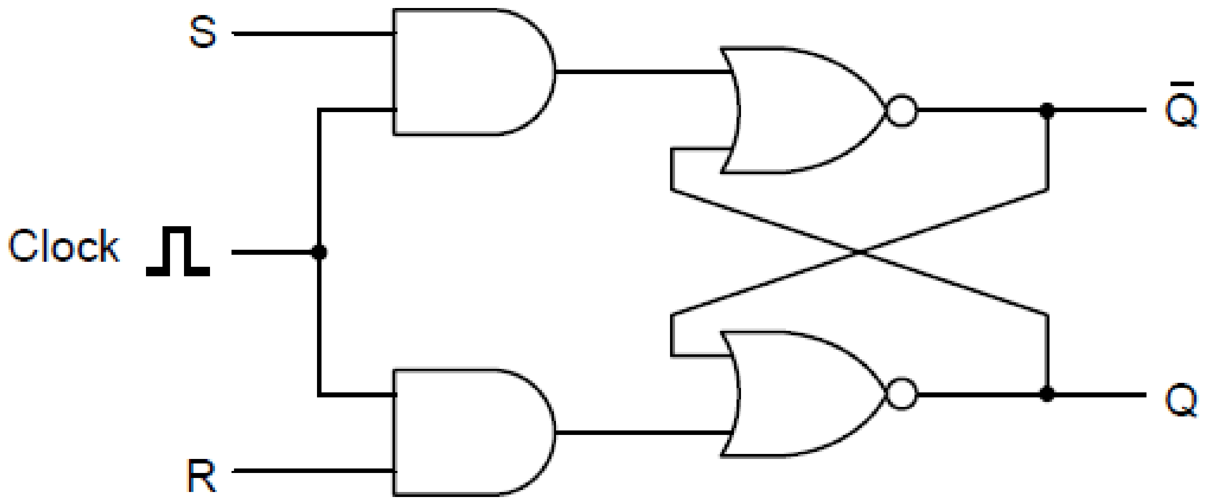


- Duty cycle = t_{PH} **/ (**t_{PH} **+** t_{PL}) x 100%
  - The clock signal shown above has a 50% duty cycle
  - The clock signal shown below has a 25% duty cycle
  - ***Period:*** The time to complete one clock cycle
    - Period = $\mathbf{t_{cycle}}$
  - ***Frequency:*** The inverse of the period, f = 1/period



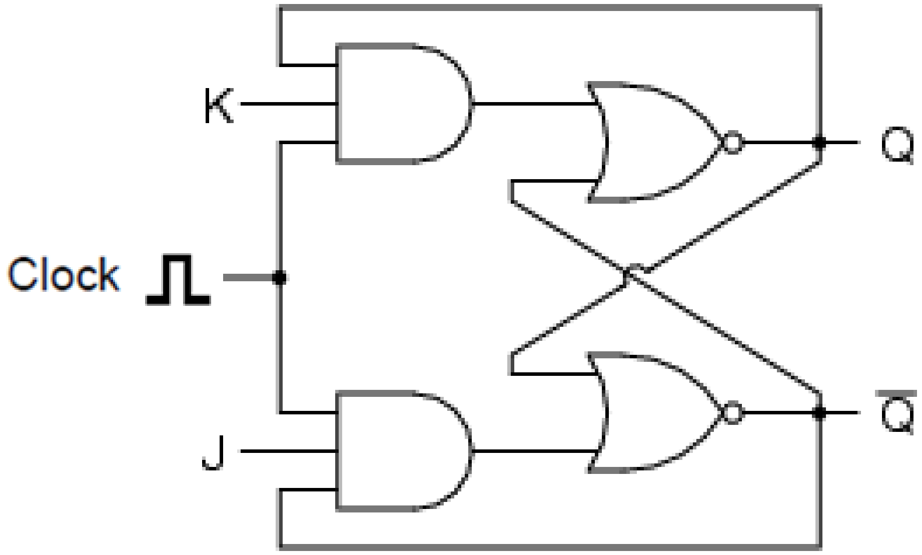
Clocked SR Latches

- SR Latch is an asynchronous operation, yet
- Preventing the latch from changing state, except at "**certain specific time**"
  - Only when Clock is 1, the latch is sensitive to S or R
  - How to resolve the forbidden condition when S=R=1?
    - When the clock is "up"



Clocked JK Latch

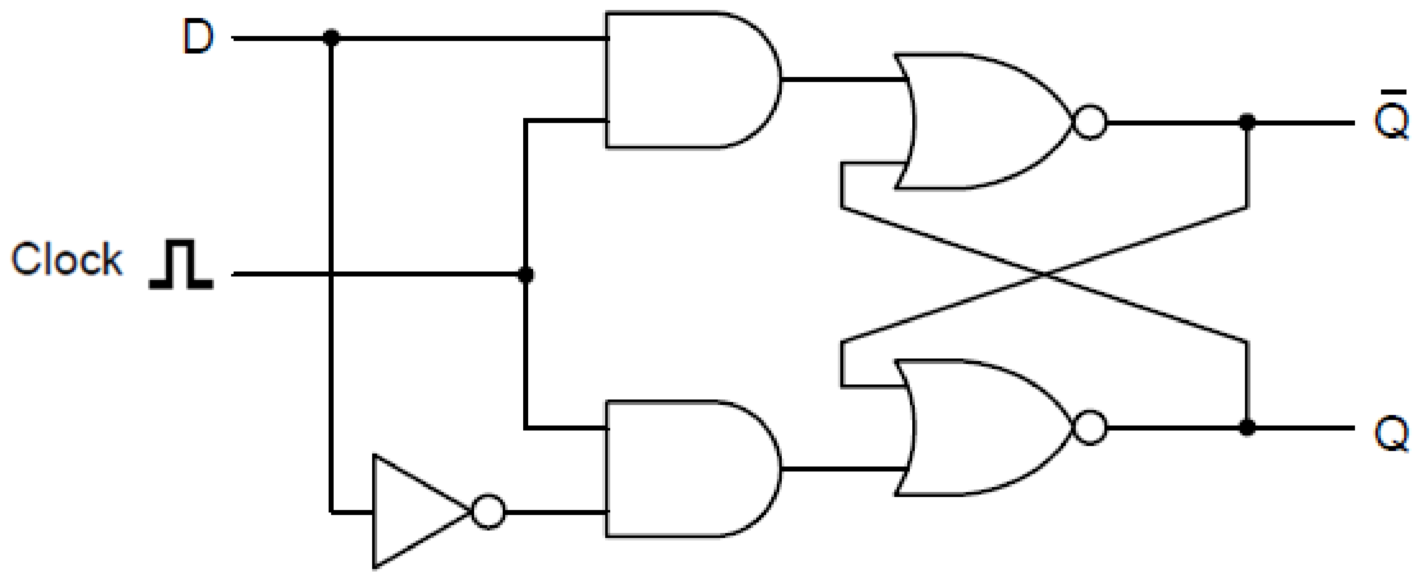
- Avoid the SR latch’s instability by preventing the inputs being 1 at the same time
- In this case, all possible combinations of input values are valid



C	J	K	Q	$\overline{Q}$
↑	0	0	latch	latch
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	toggle	toggle
x	0	0	latch	latch
x	0	1	latch	latch
x	1	0	latch	latch
x	1	1	latch	latch

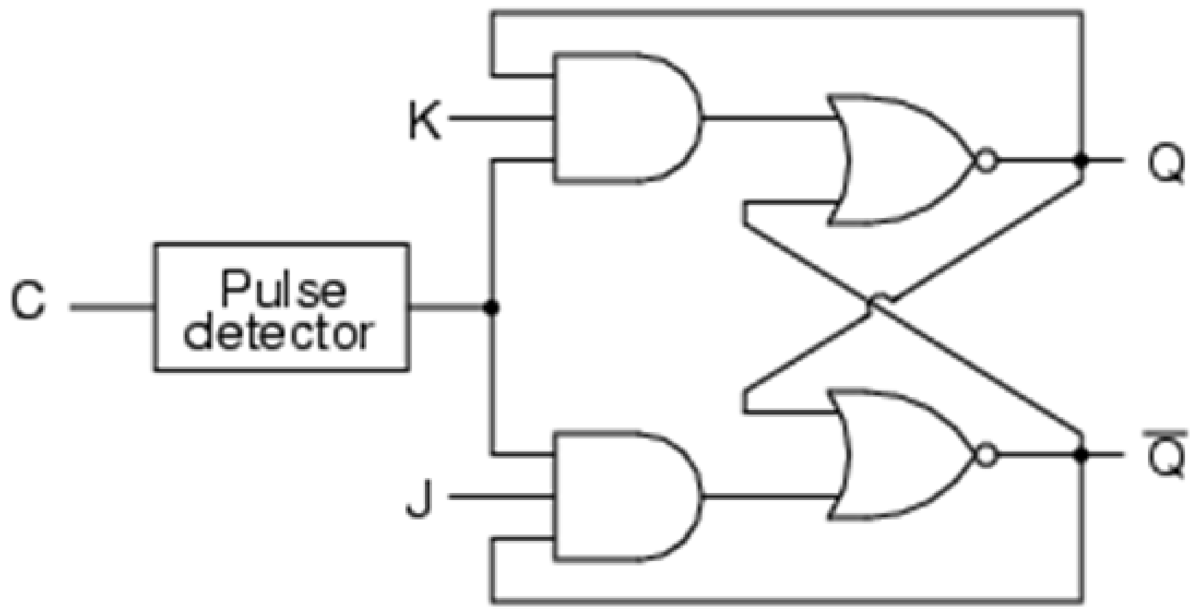
Clocked D Latches

- Avoid the SR latch’s instability by giving R as the inverse of S
- D: input – the current data
- Q: output – the stored value
- To **load** the current value of D, just give a **positive pulse** on the clock line
- ***What if D changes while the clock stays in HIGH (active)?***



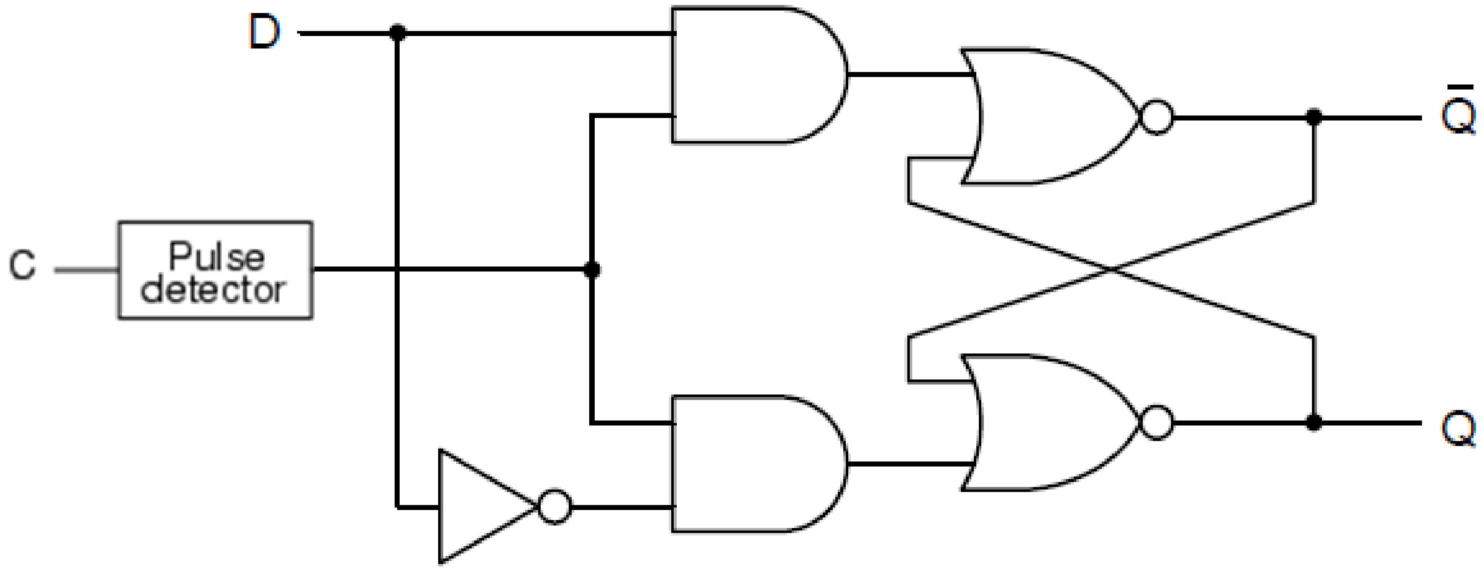
J-K Flip-Flop

- Only allow the output to change at a clock pulse



D Flip-Flop

- Only allow the output to change at a clock pulse



Latches vs. Flip-Flops

- Latch lacks a mechanism to shift control to the **clock edge**
- The state changes when the clock is active

  Level-Triggered

  - E.g., if J=K=1, and the clock is up, then the output of J-K latch will ***flip continuously***, and it is not what we can control easily

- **Flip-Flop**: State transition occurs **when the clock transitions from 0 to 1 (rising) or from 1 to 0 (falling)**

  Edge Triggered

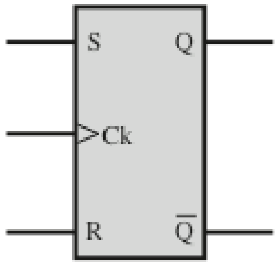
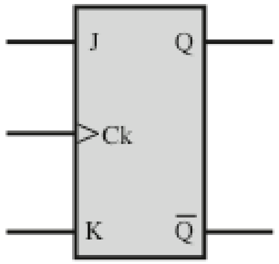
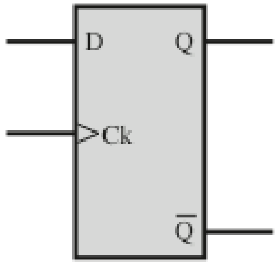
  - It's called a Flip-flop because **output Q is flipped back and forth**

- Sometimes Flip-Flops and latches are used as the same
- **But in our class, we make the difference clear**

State Table

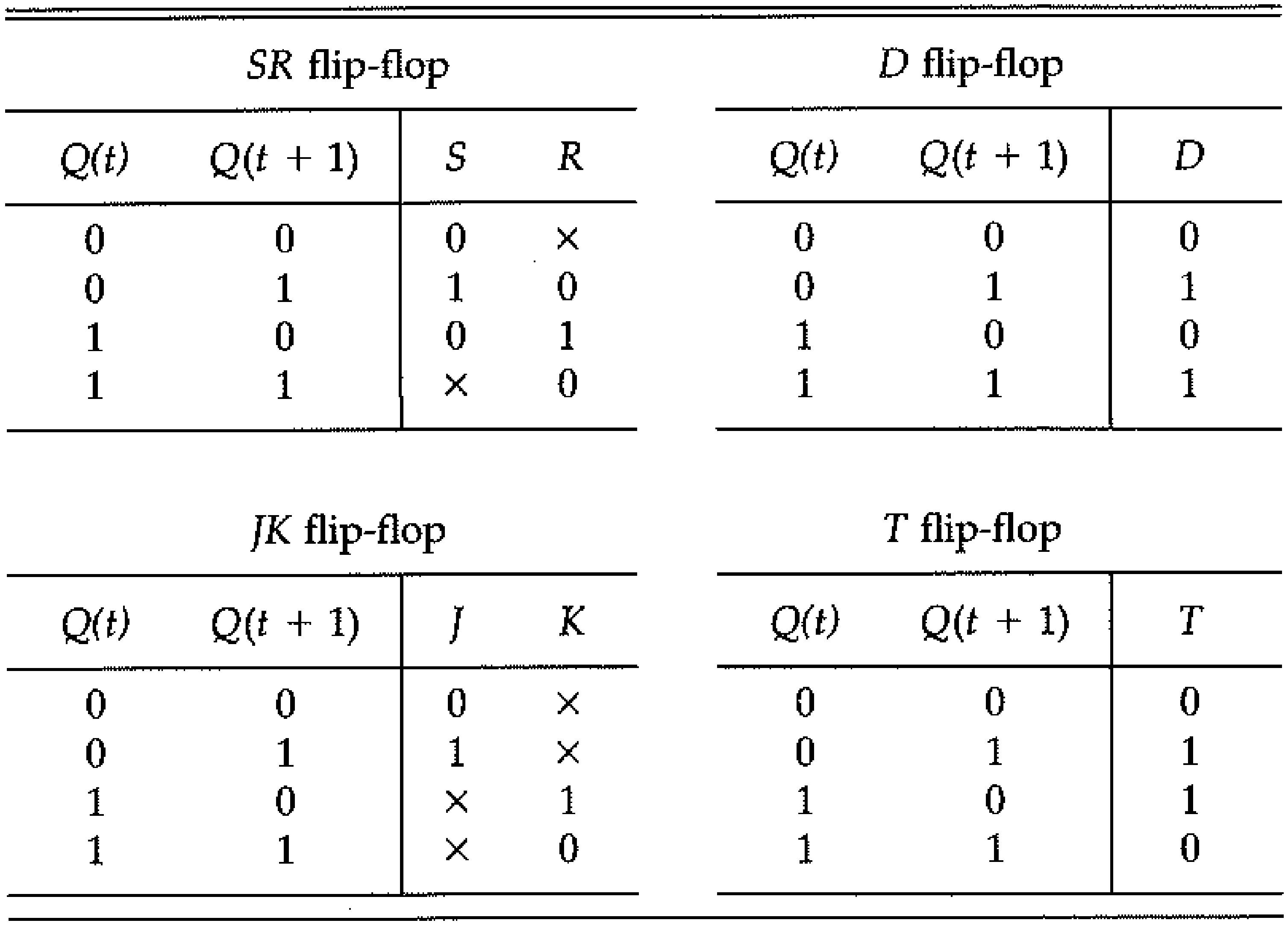
A(t)	B(t)	(input) X	(output) Y	A(t+1)	B(t+1)
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	0	1	0

Characteristic Table

Name	Graphical Symbol	Characteristic Table
S-R		$\begin{array}{cc\|c} S & R & Q_{n+1} \\ \hline 0 & 0 & Q_n \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & - \end{array}$
J-K		$\begin{array}{cc\|c} J & K & Q_{n+1} \\ \hline 0 & 0 & Q_n \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & \overline{Q_n} \end{array}$
D		$\begin{array}{c\|c} D & Q_{n+1} \\ \hline 0 & 0 \\ 1 & 1 \end{array}$

Basic Flip-Flops and the characteristic table

Excitation Table



SR flip-flop

$Q(t)$	$Q(t+1)$	S	R
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

D flip-flop

$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

JK flip-flop

$Q(t)$	$Q(t+1)$	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

T flip-flop

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

"X" is "don't care"

Design a Sequential Circuit

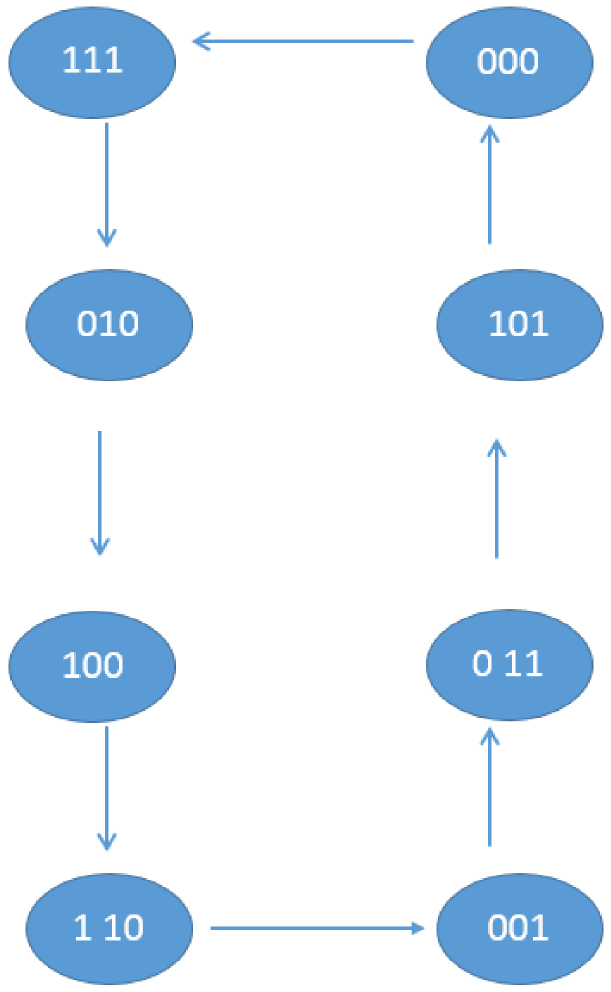
- Design a sequential circuit

1. Draw a ***State Machine*** (state diagram)
2. Figure out the ***Inputs*** and ***Outputs***
3. Build a ***State Table*** and derive an ***Excitation Table***
4. Derive a ***Boolean Equation*** using ***K-map***
5. Build the **sequential circuit**

Let’s design a 2-bit binary counter

- A sequence of repeated binary states 00, 01, 10, 11 whenever the input is 1.

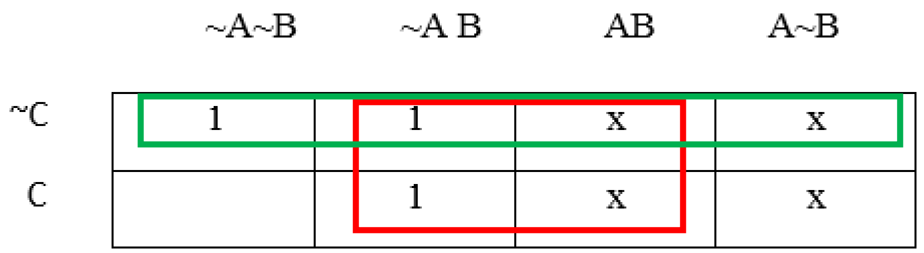
Draw a State Machine


$$\begin{tikzcd}
111 \arrow[d] & 000 \arrow[l] \\
010 \arrow[d] & 101 \arrow[u] \\
100 \arrow[d] & 0\ 11 \arrow[u] \\
1\ 10 \arrow[r] & 001 \arrow[u]
\end{tikzcd}$$

Build a State Table and derive an Excitation Table

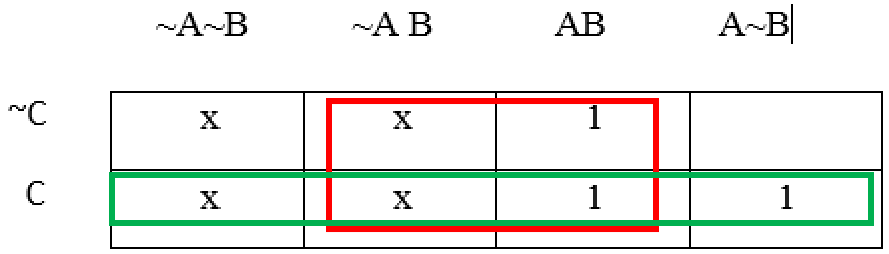
A	B	C	A_{t+1}	B_{t+1}	C_{t+1}	J_a	K_a	J_b	K_b	J_c	K_c
0											
0											
0											
0											
1											
1											
1											
1											

Derive a Boolean Equation using K-map



	~A~B	~A B	AB	A~B
~C	1	1	x	x
C		1	x	x

Ja = B + ~C



	~A~B	~A B	AB	A~B
~C	x	x	1	
C	x	x	1	1

Ka = B + C

How about

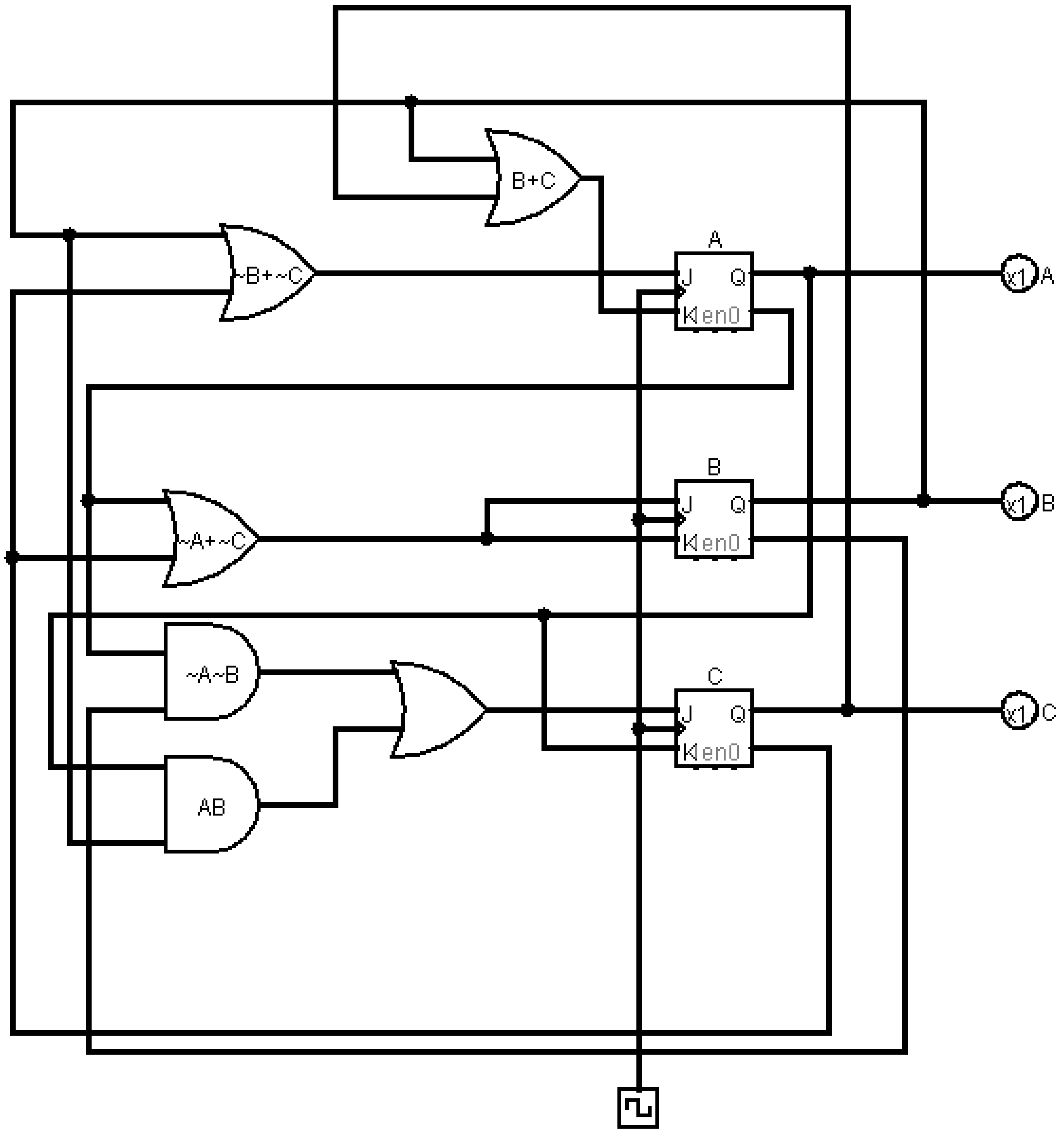
Jb

Kb

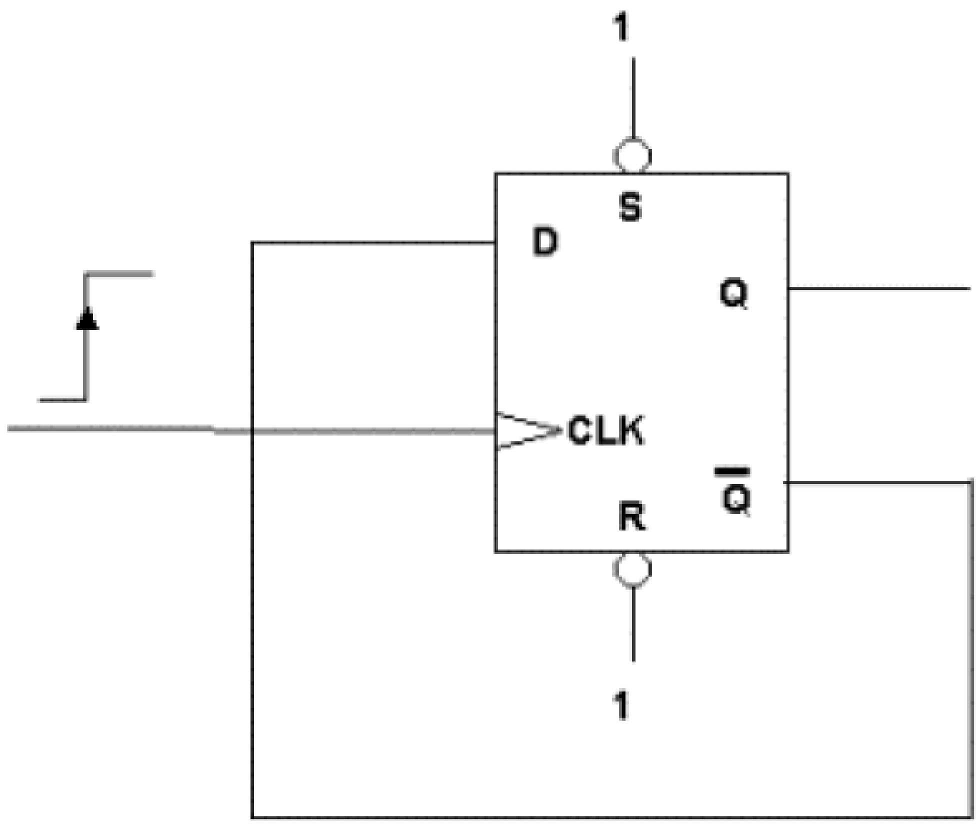
Jc

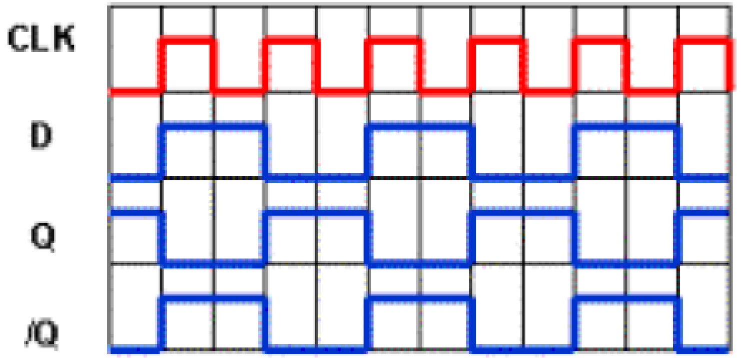
Kc

Build the Sequential Circuit



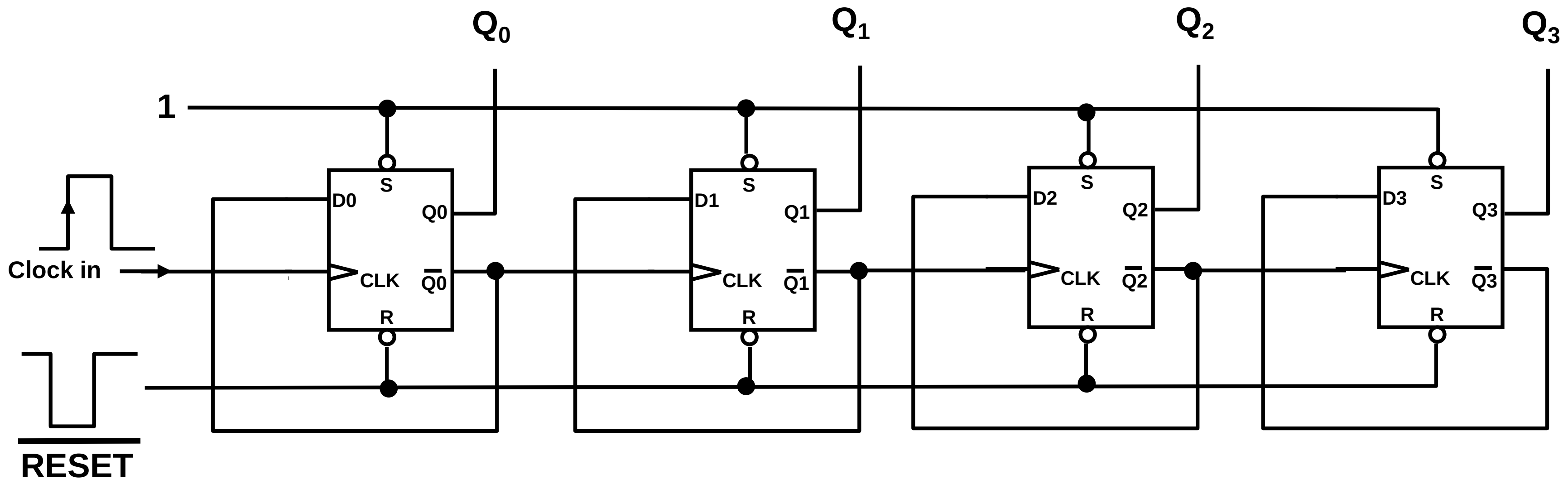
D Flip-Flop using Its Own Output as Input





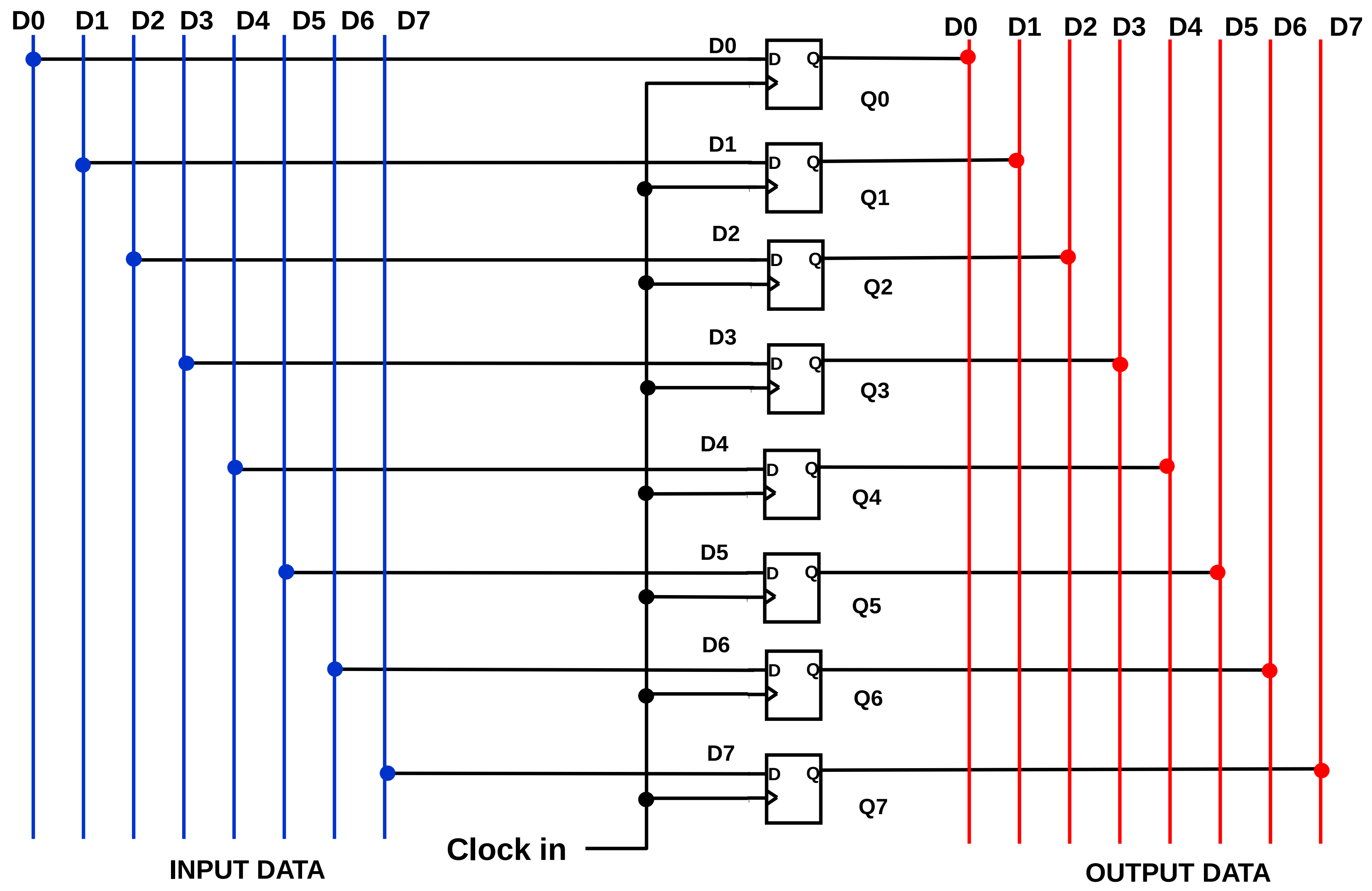
The “D” FF as a Counting Element

- **A 4-bit binary *ripple counter***
  - *the pulses “ripple” through the circuit*



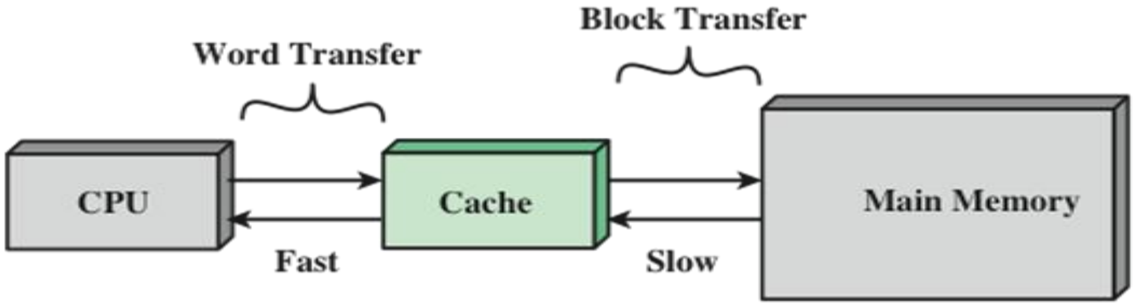
- Each “D” FF **divides the incoming clock frequency by 2**
- **RESET** sets **all Q output to 0** without a clock signal (asynchronous)
- Counts as fast as the first stage can toggle, but cannot be read until the count has rippled through to the last stage
- Can build counter/dividers of any length, any binary divisor
  - **Clock frequency at output Q3 equals $f_{\text{Clockin}} \div 16$**

“D” Flip-Flop as a Storage Register

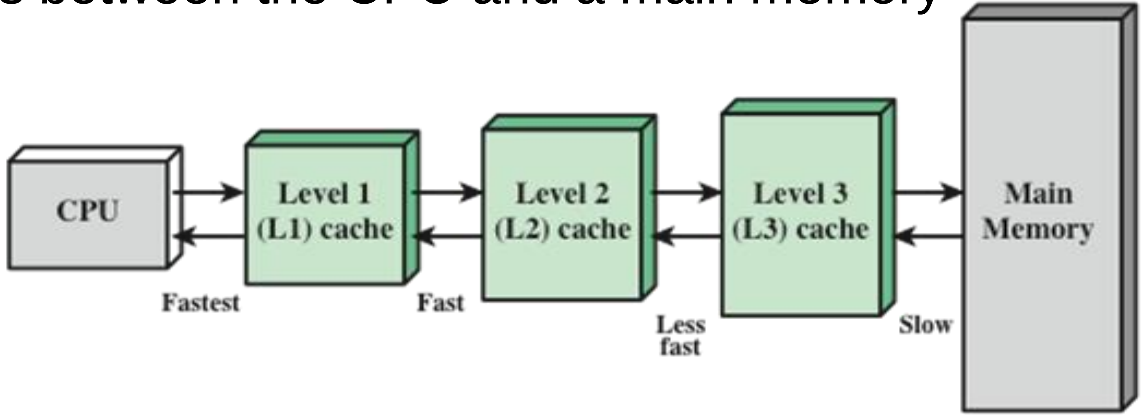


Cache and Main Memory

- A computer might have one or more caches between CPU and main memory



  - L1 cache usually means “On-chip” cache
  - L2/ L3 caches between the CPU and a main memory



Effective Execution Time (EET)

- **Block**: unit of data transfer (called ***refill line*** as well)
- **Hit Rate**: percentage of accesses found in cache
- **Miss Rate**: percentage of accesses not in cache (1 - hit rate)
- **Hit Time**: time to access cache
- **Miss Penalty**: time to replace block in cache with appropriate one (replace a block, not a data), tends to be large
- **Effective Execution Time (EET):**
  - ***hit rate * hit time + miss rate * miss penalty***
  - **The goal is to achieve the minimum EET!**
  - Decreasing the miss rate (simultaneously increasing hit rate) is an important method

Cache Design

- Goal: To increase the performance of using Caches
- Issues:
  - How to map the data in memory to the data in cache? How do we know if a data item is in the cache? If it is, how do we find it? (***Mapping Function***)
  - What happens if we change a data value in cache? (***Write Policy***)
  - What to replace when the cache is full? (***Replacement Algorithm***)
  - Is it a physical cache, or virtual cache? (***Cache Address***)
  - How big should be the Cache? (***Cache Size***)
  - How many adjacent data should come together in a cache? (***Line/Block Size***)
  - What is the optimal number of caches? (**Number of Caches**)

Mapping Function

- Processor can access the data/instruction by its **address in main memory.**

- Cache **DOES NOT** have an address!

- How to map the address of memory to some data in cache?
- Mapping Schemes:
  1. Direct Mapping
  2. Associative Mapping
  3. Set-Associative Mapping

Cache Mapping Function – Summary

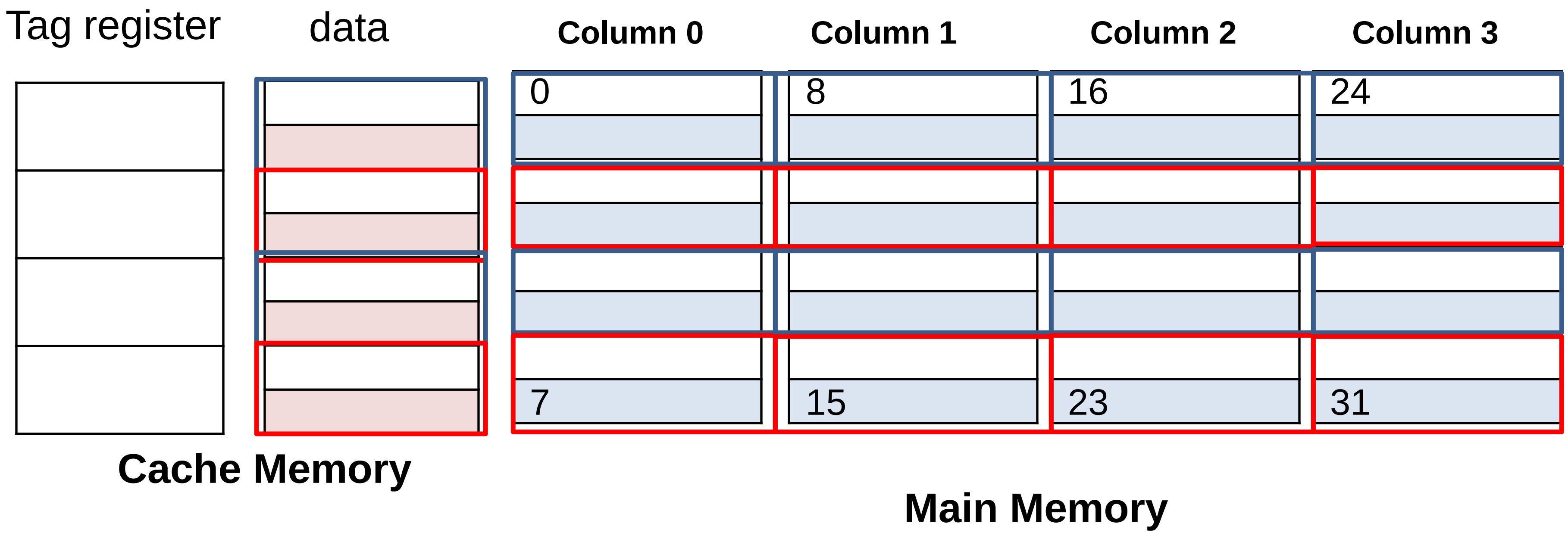
- **Directed mapping**
  - The main memory address is divided into **Tag**, **Row** and **Offset** bits
  - Tag bit matches the log of the number of columns in a main memory
  - Each column has the same size with the cache
  - One to one mapping between blocks in memory and cache - restrictive

- (Fully) **Associative mapping**
  - The main memory address is divided into **Tag** and **Offset** bits
  - Conceptually each block is one column
  - Any block from main memory can be mapped any available cache block
  - Expensive as search is needed to find matching tag

- **N-way set-associative mapping**
  - The main memory address is divided into **Tag**, **Set** and **Offset** bits
  - Each column is same as the cache size divided by N
  - Blocks are mapped within a set

Directed Mapping

- Rearrange main memory as a number of columns (pages) so that **each column (page) has the same size as the cache**
- Main memory and caches are divided into equally sized blocks (refill lines)
  - Block (Line) size: Typically between 4 and 64 bytes long (**must be power of 2**)



Directed Mapping

Divide the address into **tag**, **row** and **offset bits**

- Memory size: **L** bytes
- Direct-mapped Cache size: **M** bytes
- The Cache block size: **K** bytes

➔ # Columns in a main memory: **L/M** → **# tag bit** is log$_2$(L/M)

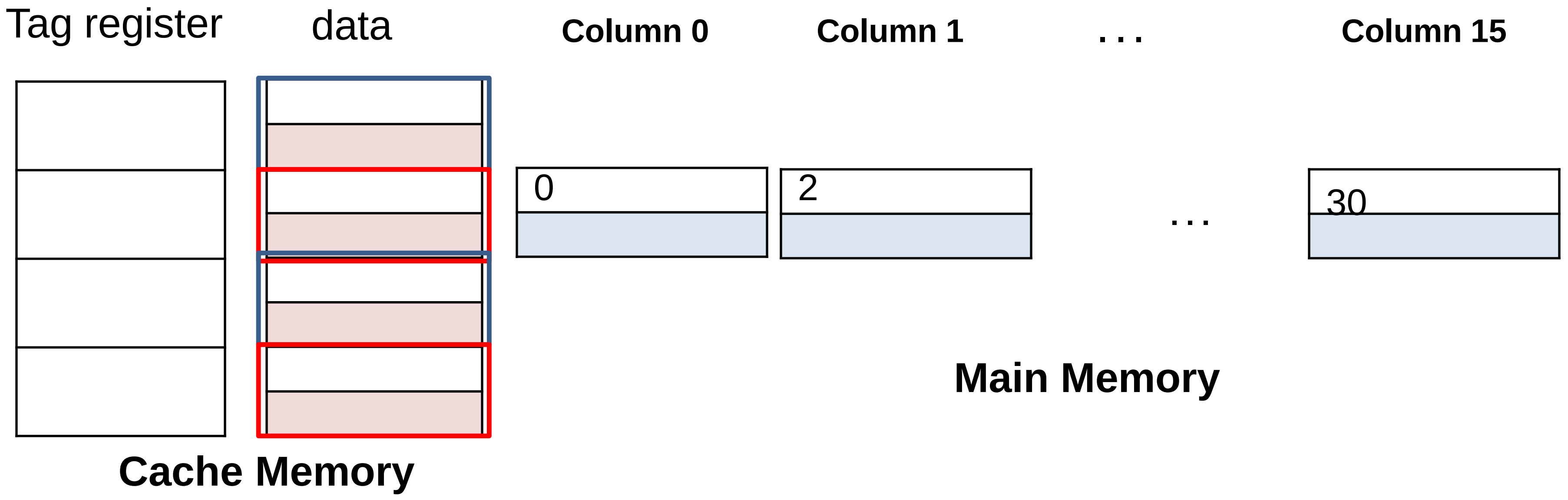
➔ # Blocks in a cache memory: **M/K** → **# row number bit** is log$_2$(M/K)

➔ block size in a main memory: **K** → **offset bit** is log$_2$K

Tag	**Row/line**	**Offset**

Associative Cache

- Rearrange the main memory based on the size of a block (refill line size)
- Each column is one block
  - number of columns = number of blocks
- Each block can be mapped to any available block



Associative Cache

Divide the address into tag and offset (**No ROW number!!**)

- Memory size: **L** bytes
- ***Associative*** Cache size: **M** bytes
- Cache block size: **K** bytes

➔ # Columns in a main memory: **L/K** → # tag bit is $\log_2(L/K)$

➔ Block size in a main memory: **K** → offset bit is $\log_2(K)$

Tag	Offset

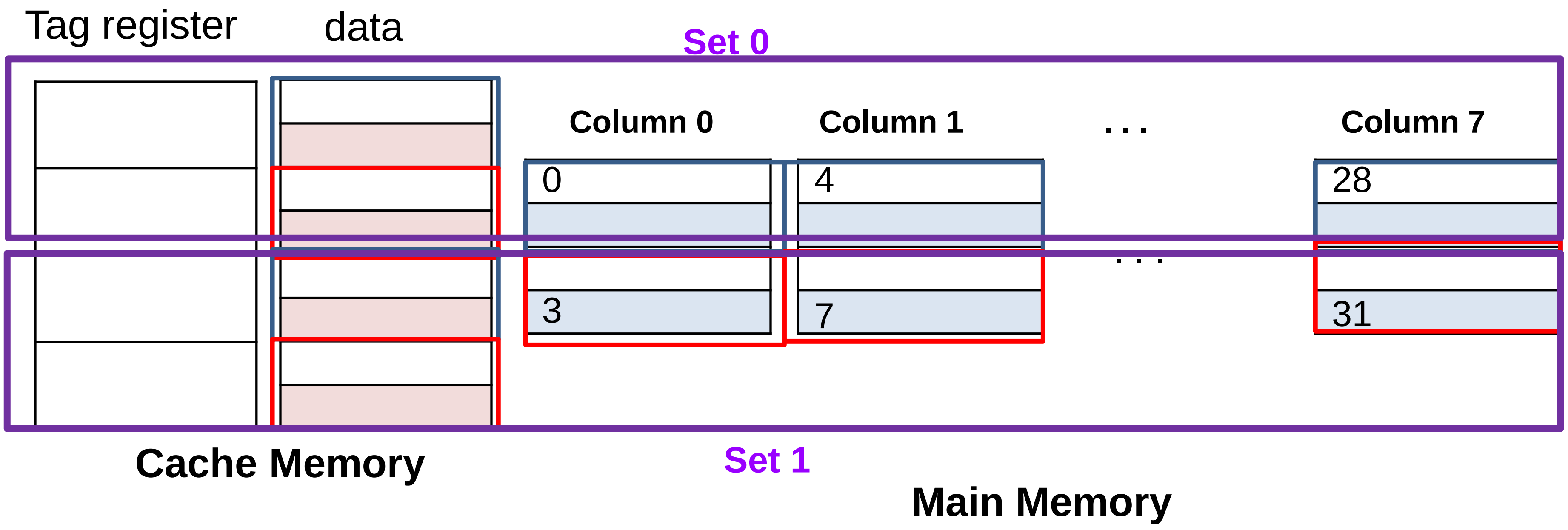
N-way set-associative cache

- **Combines the properties of the direct-mapped and associative cache** into one system
  - Direct-mapped Cache is very restrictive
  - Associative Cache is very expensive (in terms of search)
  - So, combine those two → Set-Associative Cache
  - Most commonly used in modern processors (4-way set-associative cache)

- **N-way set-associative**
  - Divide the cache into **multiple sets**
  - Each set contains **N number of blocks**

2-way Set-Associative Cache

- Want to read data at address 5(00101_2) in main memory into a cache



N-way Set-Associative Cache

Divide the address into **tag**, **set** and **offset** bits

- Memory size: **L** bytes
- ***N-Way Set-Associative*** Cache size: **M** bytes
- The Cache block size: **K** bytes

➔ Block size in a main memory: **K** → offset bit is $\log_2(K)$

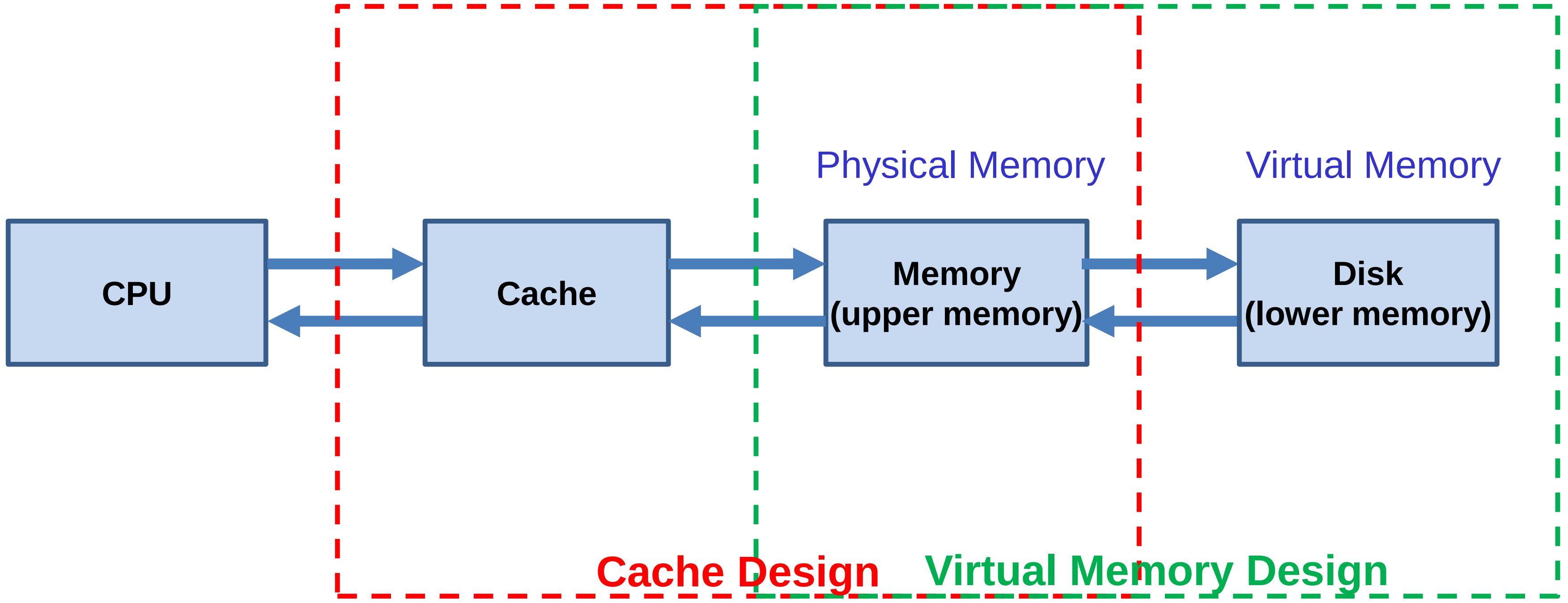
➔ # of sets in a cache: **# blocks/N** = **(M/K)/N** → # set bit is $\log_2((M/K)/N)$

➔ # Columns (Tag) in a main memory: **L/((M/K)/N * K) = L/(M/N)**

→ # tag bit is $\log_2(L/(M/N))$

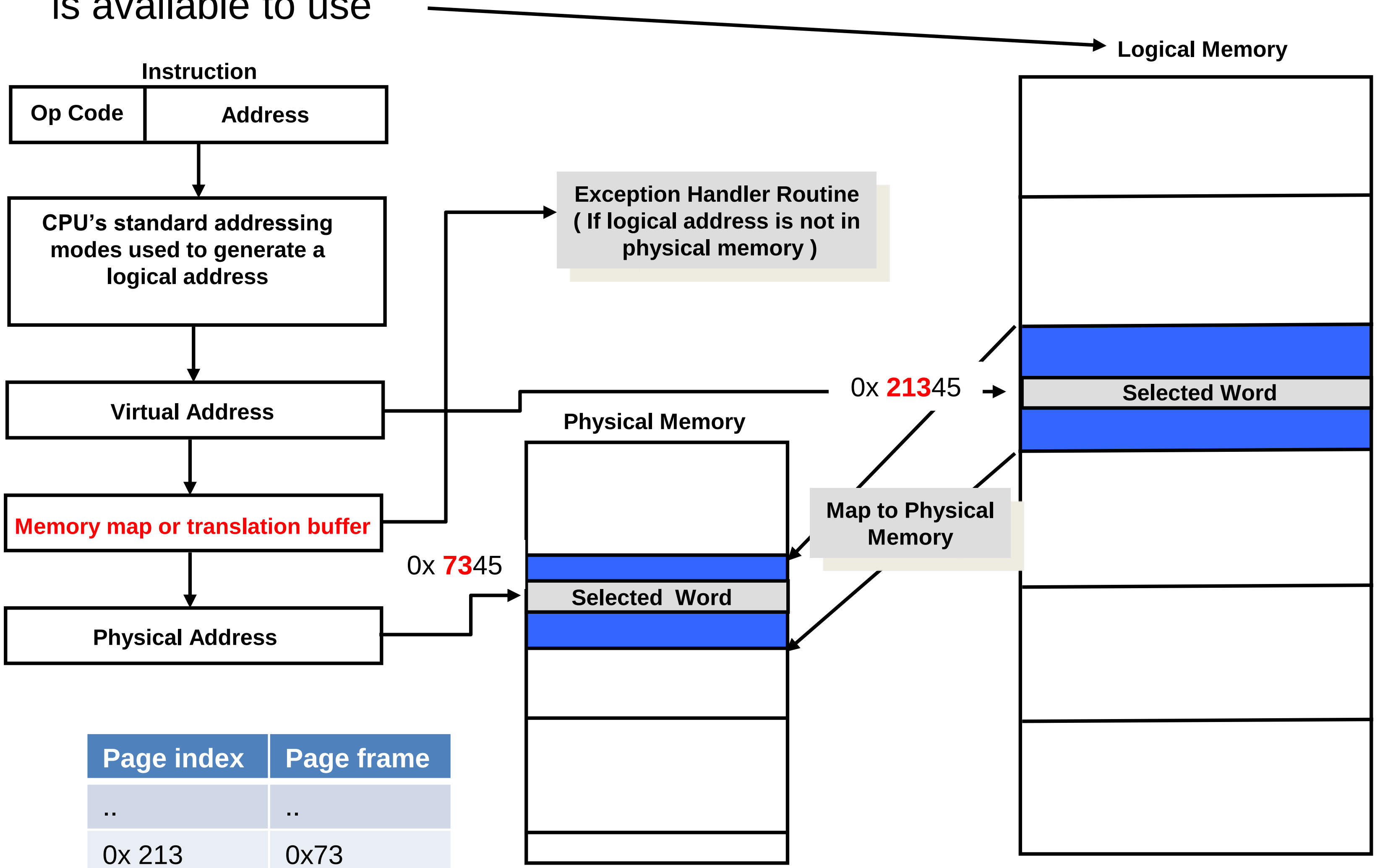
Tag	**Set**	**Offset**

Accessing Memories in a Computer System



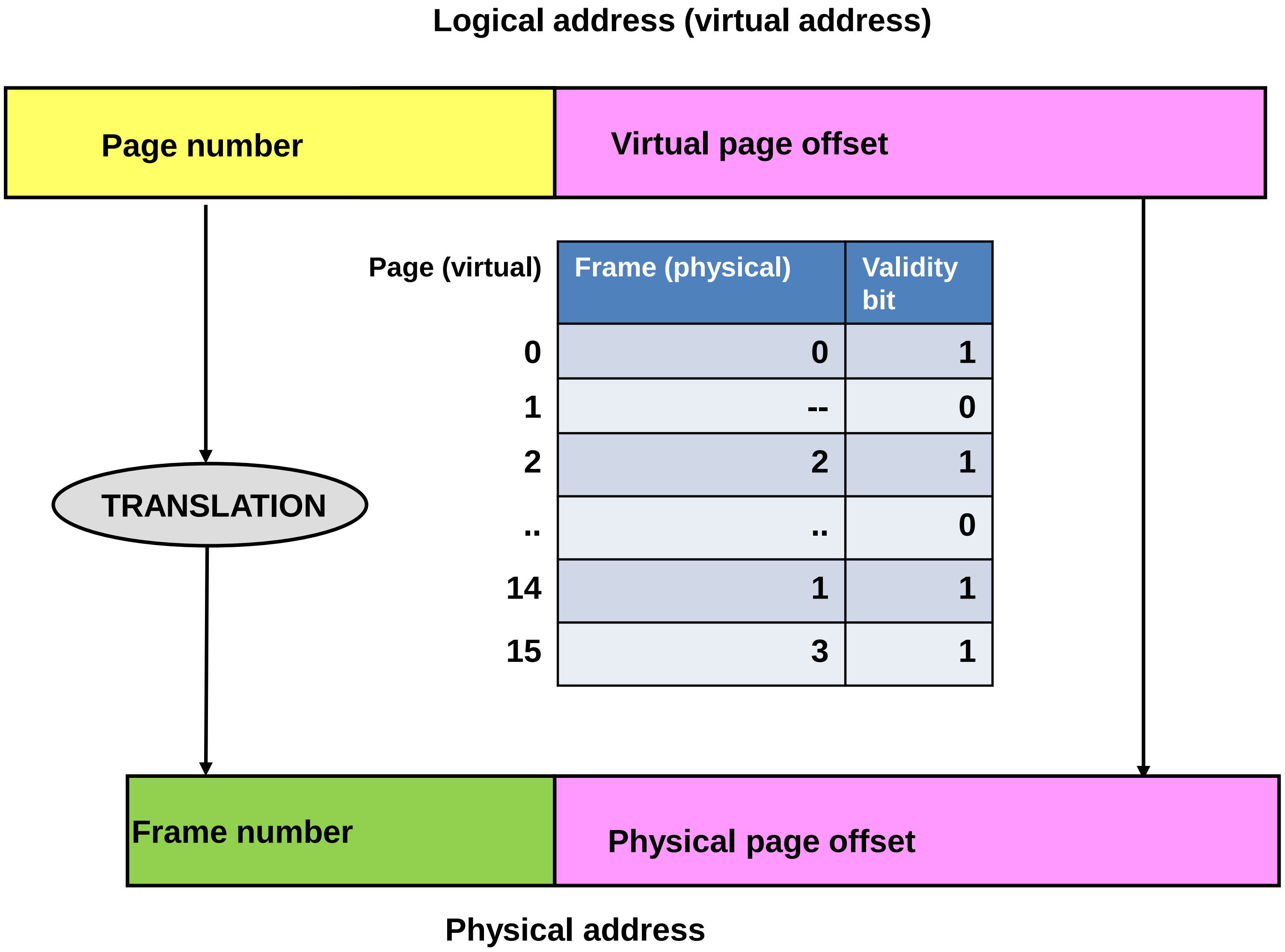
Components of a Virtual Memory System

- ***Virtual (Logical) Memory*** is the memory space that the program thinks it is available to use


Logical Memory
Instruction
Op Code
Address
CPU's standard addressing modes used to generate a logical address
Exception Handler Routine (If logical address is not in physical memory)
Virtual Address
Memory map or translation buffer
Physical Address
0x 21345
Selected Word
Physical Memory
Map to Physical Memory
0x 7345
Selected Word

Page index	Page frame
..	..
0x 213	0x73
..	..

Page Table: Map Pages to Frames (O/S)



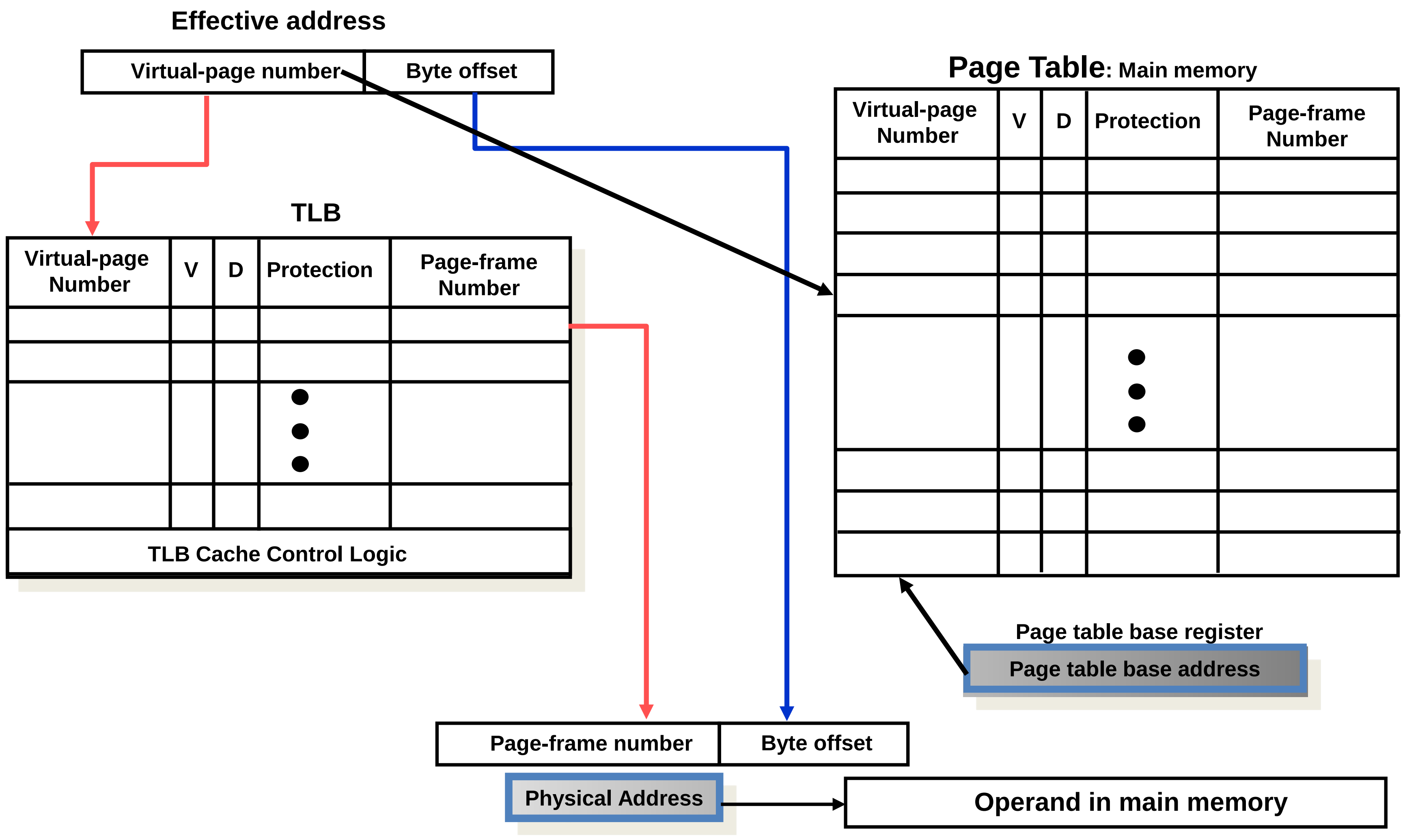
Page (virtual)	Frame (physical)	Validity bit
0	0	1
1	--	0
2	2	1
..	..	0
14	1	1
15	3	1

Translation Lookaside Buffer (TLB)

- Most computer systems keep their page tables in main memory
  - ***Page-table base register*** points to the beginning of the table
  - O/S can modify the page table base register using supervisor mode instructions
  - In theory, main memory (non-cached) accesses could take **twice** as long, because the page table must be accessed first

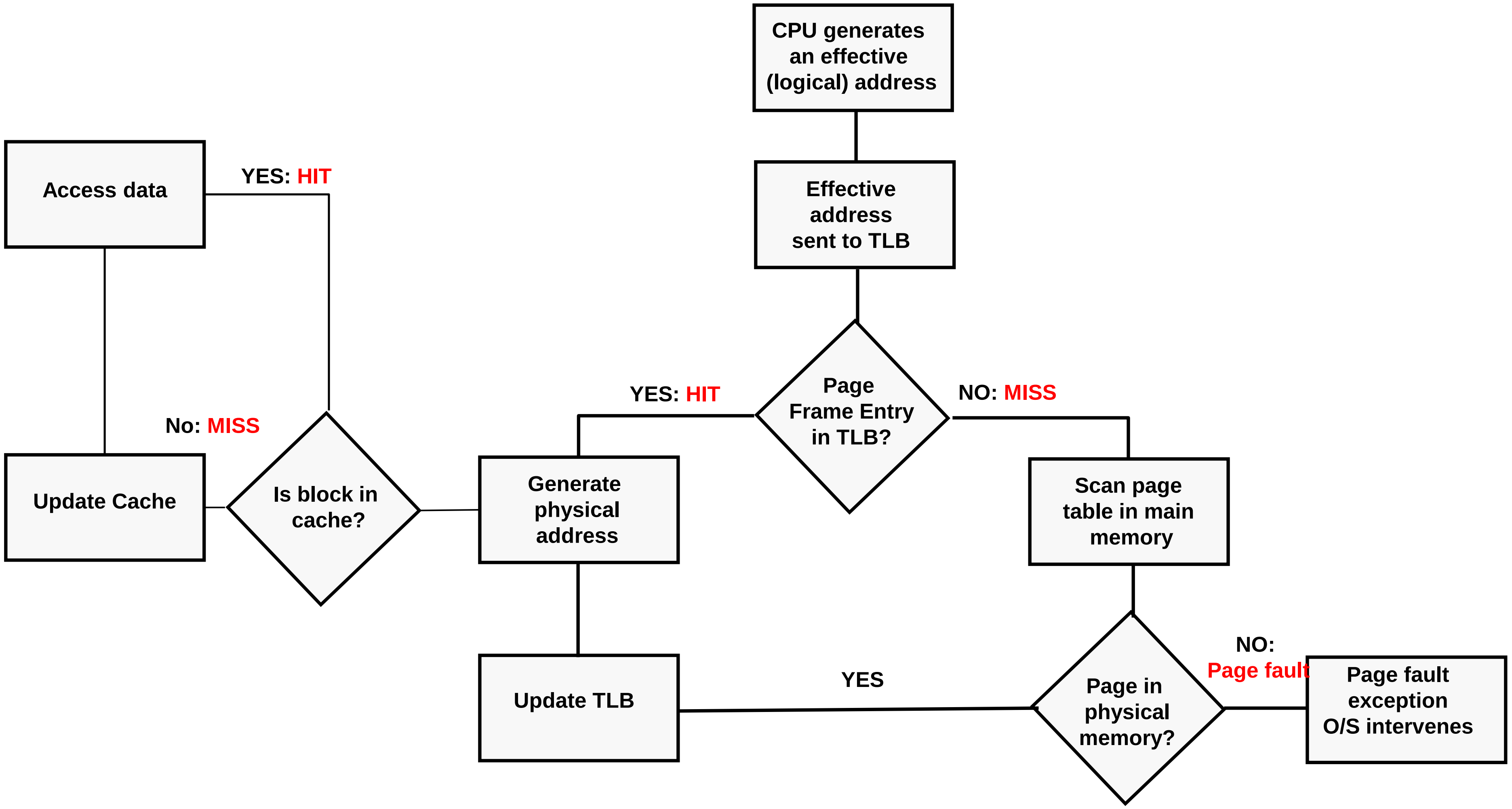
- Modern processors maintain a ***Translation Lookaside Buffer (TLB)*** as **part of the page map**
  - Holds the **same information as part of the page table**
    - ***cache for page table***
  - TLB cache algorithm holds **only most recently accessed pages**
    - Flushes ***Least Recently Used*** *(LRU)* entries from TLB
    - Holds **only mapping for valid pages**

Components of a Paging System



Put All Together

The TLB, Page Table, Main, and Cache Memory



CPU generates an effective (logical) address
Effective address sent to TLB
Page Frame Entry in TLB?
YES: HIT
NO: MISS
Generate physical address
Scan page table in main memory
Page in physical memory?
NO: Page fault
Page fault exception O/S intervenes
YES
Update TLB
Is block in cache?
YES: HIT
No: MISS
Access data
Update Cache