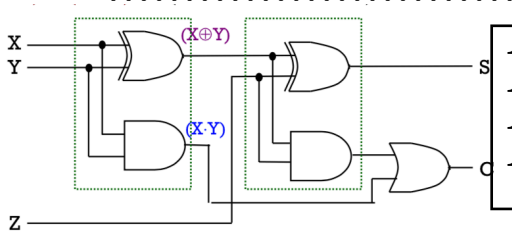
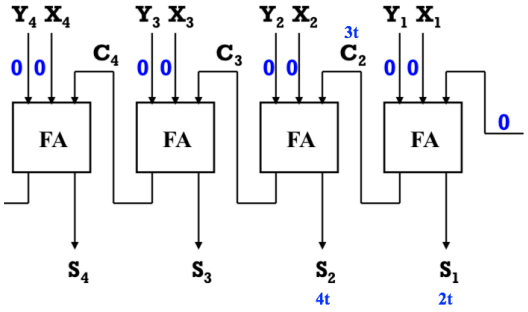
Precedence from highest to lowest: Not (') And (·) Or (+)
minterm - product term - SOP - find 1
maxterm - sum term - POS - find 0
Each minterm is the complement of the maxterm
SOP - 2-level AND-OR/NAND circuit
POS - 2-level OR-AND/NOR circuit
PLAs may not be able to implement a given mapping due to not having enough minterms.
Half Adder:
$C = X \cdot Y$, $S = X' \cdot Y + X \cdot Y' = X \oplus Y$
K-Maps -> SOP
Gate-Level (SSI) Design: Full-Adder, Code Converter
$C = X \cdot Y + (X \oplus Y) \cdot Z$, $S = X \oplus (Y \oplus Z)$
Block-Level Design: block: 4-bit parallel/ripple-carry adder: **$C_{i+1} = X_i \cdot Y_i + (X_i \oplus Y_i) \cdot C_i$, $S_i = X_i \oplus Y_i \oplus C_i$**



1 KB= 2^{10} bytes;
1 MB = 2^{20} bytes;
1 GB = 2^{30} bytes;
1 TB= 2^{40} bytes.

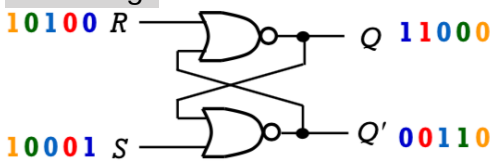
$X+X \cdot Y=X$ $\quad X \cdot Y+X \cdot Z+Y \cdot Z=X \cdot Y+X' \cdot Z$ $\quad Y' \cdot Z + Y \cdot Z'=Y \oplus Z$
$X+X' \cdot Y=X+Y$ $\quad Y' \cdot Z' + Y \cdot Z=(Y \oplus Z)'$ $\quad X+Y=(X \oplus Y)+X \cdot Y$

Given a logic gate with delay t. If inputs are stable at times $t_1, t_2, \ldots, t_n$, then the earliest time in which the output will be stable is: **$\max(t_1, t_2, \ldots, t_n) + t$**



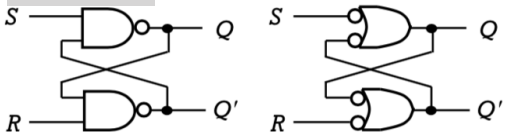
S-R Latch: Q(t+1)=S+R'•Q, S•R=0

Active-High



S	R	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	indeterminate	

Active-Low



Gated D Latch: Q(t+1)=D

EN	D	Q(t+1)	
1	0	0	Reset
1	1	1	Set
0	X	Q(t)	No change

combinational circuit: each output depends entirely on the immediate (present) inputs
sequential circuit: each output depends on both present inputs and state
- Synchronous: outputs change only at specific time
- Asynchronous: outputs change at any time

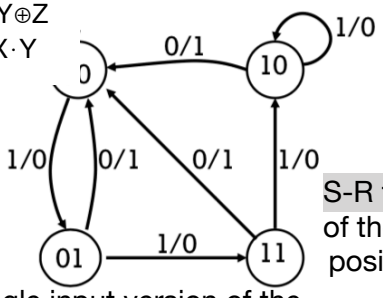
Multivibrator: a class of sequential circuits
§ Bistable (2 stable states)
- Latches and flip-flops.
- differ in the methods used for changing state.

§ Monostable or one-shot (1 stable state)
§ Astable (no stable state)

Memory element: a device which can remember value indefinitely, or change value on command from its inputs.

Two types of triggering/activation
- Level/Pulse-triggered
§ Latches § ON = 1, OFF = 0
- Edge-triggered
§ Flip-flops
§ Positive edge-triggered (ON = 0 to 1; OFF = other time)
§ Negative edge-triggered (ON = 1 to 0; OFF = other time)

input/output = x/y



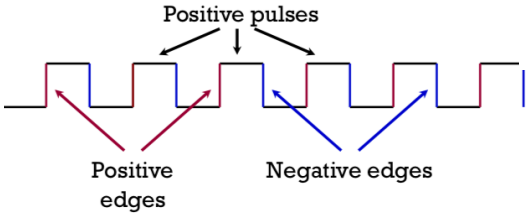
Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
AB	A^+B^+	A^+B^+	y	y
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0

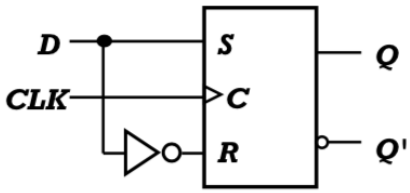
T flip-flop: Single input version of the J-K flip-flop Q(t+1)=T•Q'+T'•Q

T	CLK	Q(t+1)	Comments
0	↑	Q(t)	No change
1	↑	Q(t)'	Toggle

Flip-Flops: synchronous bistable
- Output changes state at a specified point on a triggering input called the clock.
- Change state either at the positive (rising) edge, or at the negative (falling) edge of the clock signal.

Note ">" symbol at the clock input.
> positive, o> negative edge-triggered





A positive edge-triggered D flip-flop formed with an S-R flip-flop.

S-R flip-flop: On the triggering **edge** of the clock pulse
positive edge-triggered *S-R* flip-flop

S	R	CLK	Q(t+1)	Comments
0	0	X	Q(t)	No change
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	?	Invalid

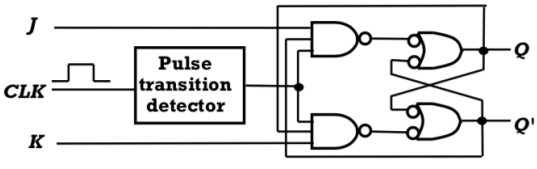
X = irrelevant ("don't care")
↑ = clock transition LOW to HIGH

D flip-flop: On the triggering **edge** of the clock pulse

D	CLK	Q(t+1)	Comments
1	↑	1	Set
0	↑	0	Reset

↑ = clock transition LOW to HIGH

J-K flip-flop: *Q* and *Q'* are fed back to the pulse-steering NAND gates.



J	K	CLK	Q(t+1)	Comments
0	0	↑	Q(t)	No change
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	Q(t)'	Toggle

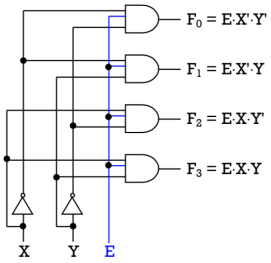
decoder: Convert binary information from *n* input lines to (a maximum of) 2^n output lines.

- Known as n-to-m-line decoder, or simply n:m or n x m decoder (m ≤ 2n).
- May be used to generate 2^n minterms of n input variables.

SOP = decoder to generate minterms + OR gate to form the sum

enable control signal: the device is only activated when the enable E = 1(one-enable) (MSI is mainly 0-enable)

X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

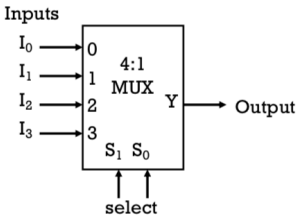


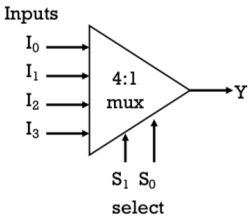
multiplexer: It steers one of 2^n inputs to a single output line, using *n* selection lines. Also known as a *data selector*.

- A number of input lines
- A number of selection lines
- One output line
- Output of multiplexer is "sum of the (product of *data lines* and *selection lines*)"
- Example: Output of a 4-to-1 multiplexer is: $Y = I_0 \cdot (S_1' \cdot S_0') + I_1 \cdot (S_1' \cdot S_0) + I_2 \cdot (S_1 \cdot S_0') + I_3 \cdot (S_1 \cdot S_0) = I_0 \cdot m_0 + I_1 \cdot m_1 + I_2 \cdot m_2 + I_3 \cdot m_3$

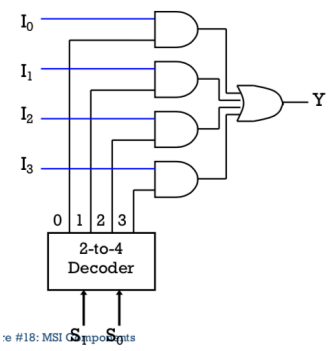
I_0	I_1	I_2	I_3	S_1	S_0	Y
d_0	d_1	d_2	d_3	0	0	d_0
d_0	d_1	d_2	d_3	0	1	d_1
d_0	d_1	d_2	d_3	1	0	d_2
d_0	d_1	d_2	d_3	1	1	d_3

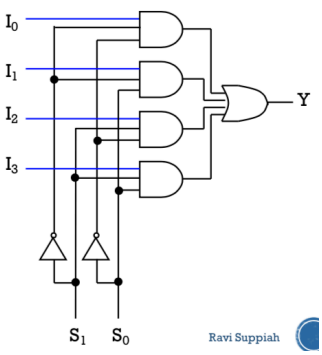
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



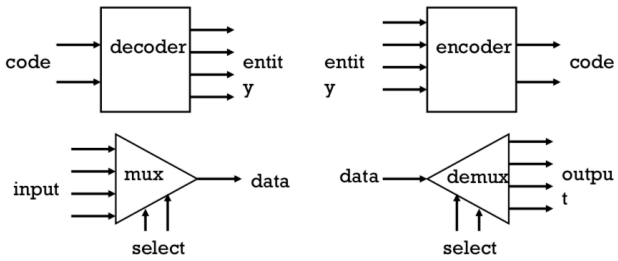


A 2^n-to-1-line multiplexer, or simply 2^n:1 MUX, is made from an n:2^n decoder by adding to it 2^n input lines, one to each AND gate.



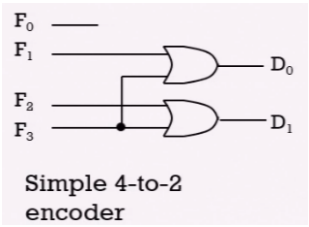


Ravi Suppiah



encoder: exactly one input line is high and the rest are low, $D_0 = F_1 + F_3$, $D_1 = F_2 + F_3$

F_0	F_1	F_2	F_3	D_1	D_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1
0	0	0	0	X	X
0	0	1	1	X	X
0	1	0	1	X	X
0	1	1	0	X	X
0	1	1	1	X	X
1	0	0	1	X	X
1	0	1	0	X	X
1	0	1	1	X	X
1	1	0	0	X	X
1	1	0	1	X	X
1	1	1	0	X	X
1	1	1	1	X	X



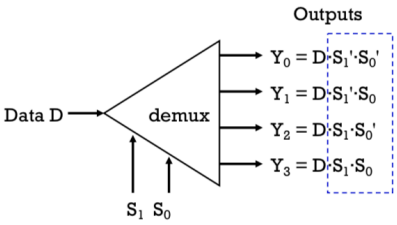
Simple 4-to-2 encoder

priority encoder:

- If two or more inputs or equal to 1, the input with the highest priority takes precedence.
- If all inputs are 0, this input combination is considered invalid.

Inputs				Outputs		
D_0	D_1	D_2	D_3	*f*	*g*	*V*
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

demultiplexers: Given an input line and a set of selection lines, a demultiplexer directs data from the input to one selected output line.
demultiplexer circuit is actually identical to a decoder with enable.



S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK Flip-flop

Q	Q^+	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

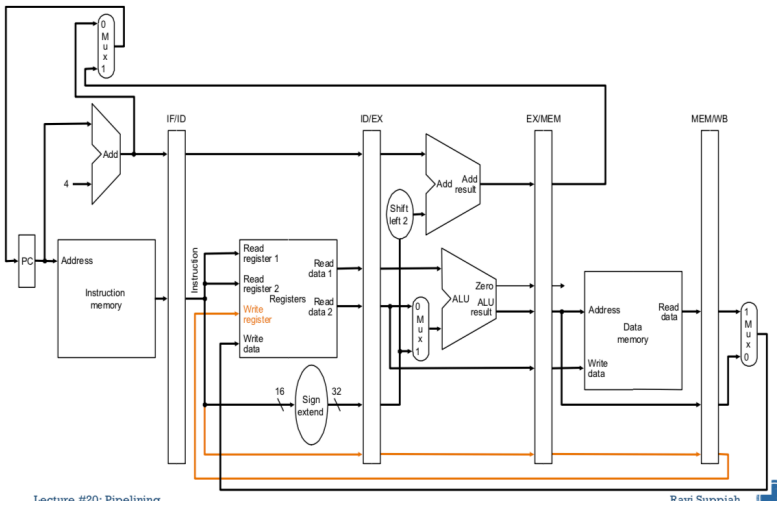
SR Flip-flop

Q	Q^+	D
0	0	0
0	1	1
1	0	0
1	1	1

D Flip-flop

Q	Q^+	T
0	0	0
0	1	1
1	0	1
1	1	0

T Flip-flop



§ **IF**: Instruction Fetch
§ **ID**: Instruction Decode and Register Read
§ **EX**: Execute an operation or calculate an address
§ **MEM**: Access an operand in data memory
§ **WB**: Write back the result into a register

- At the end of a cycle, **IF/ID** receives (stores):
  - Instruction read from InstructionMemory[PC]
  - PC + 4
- PC + 4
  - Also connected to one of the MUX's inputs (another coming later)

At the beginning of a cycle IF/ID register supplies:	**At the end of a cycle ID/EX receives:**
❖ Register numbers for reading two registers ❖ 16-bit offset to be sign-extended to 32-bit	❖ Data values read from register file ❖ 32-bit immediate value ❖ PC + 4
At the beginning of a cycle ID/EX register supplies:	**At the end of a cycle EX/MEM receives:**
❖ Data values read from register file ❖ 32-bit immediate value ❖ PC + 4	❖ (PC + 4) + (Immediate x 4) ❖ ALU result ❖ isZero? signal ❖ Data Read 2 from register file
At the beginning of a cycle EX/MEM register supplies:	**At the end of a cycle MEM/WB receives:**
❖ (PC + 4) + (Immediate x 4) ❖ ALU result ❖ isZero? signal ❖ Data Read 2 from register file	❖ ALU result ❖ Memory read data
At the beginning of a cycle MEM/WB register supplies:	**At the end of a cycle**
❖ ALU result ❖ Memory read data	❖ Result is written back to register file (if applicable) ❖ **There is a bug here.....**

	EX Stage				MEM Stage			WB Stage	
	RegDst	ALUSrc	ALUop		Mem Read	Mem Write	Branch	MemTo Reg	Reg Write
			op1	op0					
R-type	1	0	1	0	0	0	0	0	1
lw	0	1	0	0	1	0	0	1	1
sw	X	1	0	0	0	1	0	X	0
beq	X	0	0	1	0	0	1	X	0

Single Cycle
§ Cycle time:
§ $CT_{seq} = \sum_{k=1}^{N} T_k$
§ T_k = Time for operation in stage k
§ N = Number of stages
§ Total Execution Time for **I** instructions:

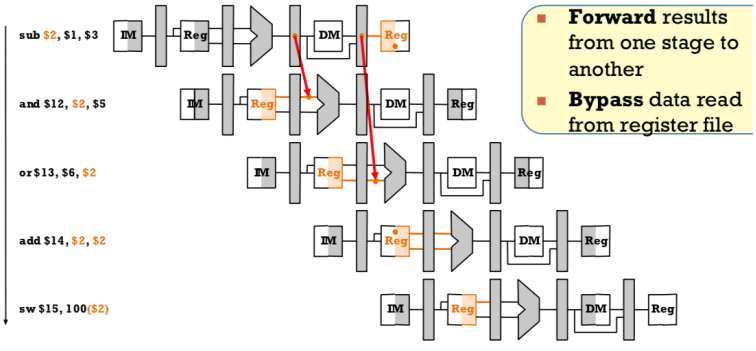
Multi Cycle
§ Cycle time:
§ $CT_{multi} = \max(T_k)$
§ max(Tk) = longest stage duration among the N stages
§ Total Execution Time for **I** instructions:
§ $Time_{multi}$ = Cycles × CycleTime = I×Average

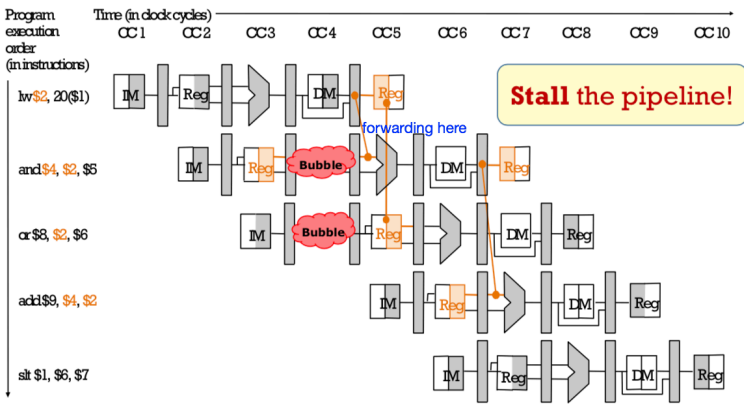
CPI×CT_{multi}
§ Average CPI is needed because each instruction takes different number of cvcles to finish

Pipeline Cycle
§ Cycle time:
§ $CT_{pipeline} = \max(T_k) + T_d$
§ max(Tk) = longest time among the N stages
§ T_d = Overhead for pipelining, e.g. pipeline register
§ Cycles needed for **I** instructions:
§ I + N -1
§ N - 1 is the cycles wasted in filling up the pipeline
§ Total Execution Time for **I** instructions:
§ $Time_{pipeline}$ = Cycle × $CT_{pipeline}$ = (I + N -1) × ($\max(T_k)$ +

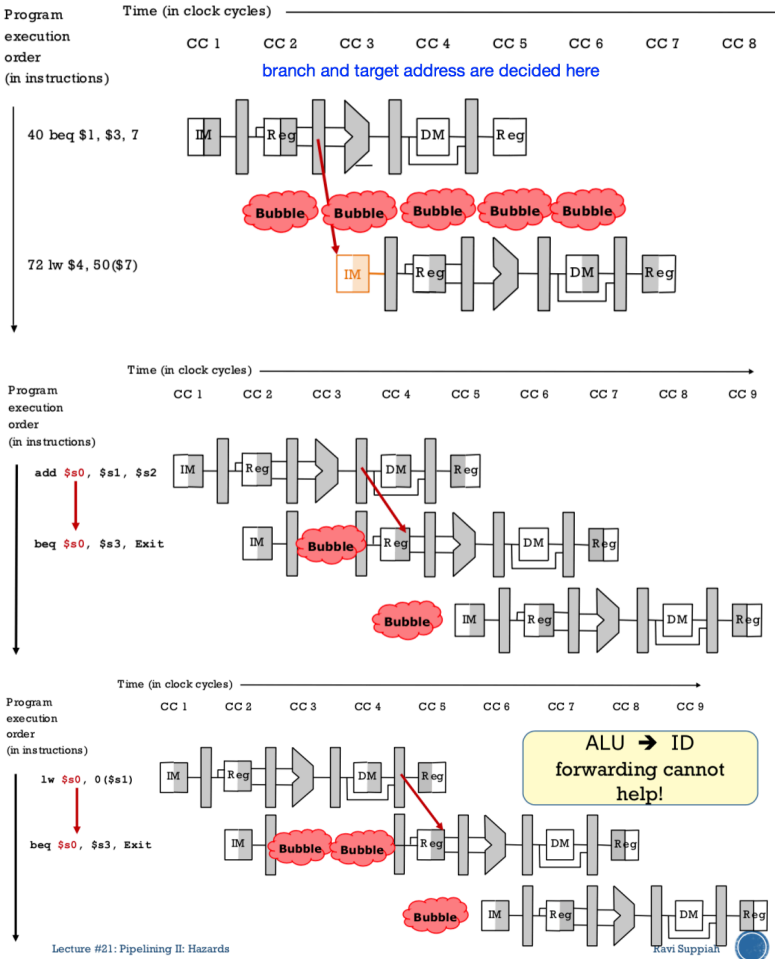
Ideal Speedup
- Every stage takes the same amount of time
- No pipeline overhead T_d=0
- Number of instructions I is much larger than number of stages N

$Speedup_{pipeline} = Time_{seq} / Time_{pipeline}$
Pipeline processor can gain **N** times speedup, where **N** is the number of pipeline stages

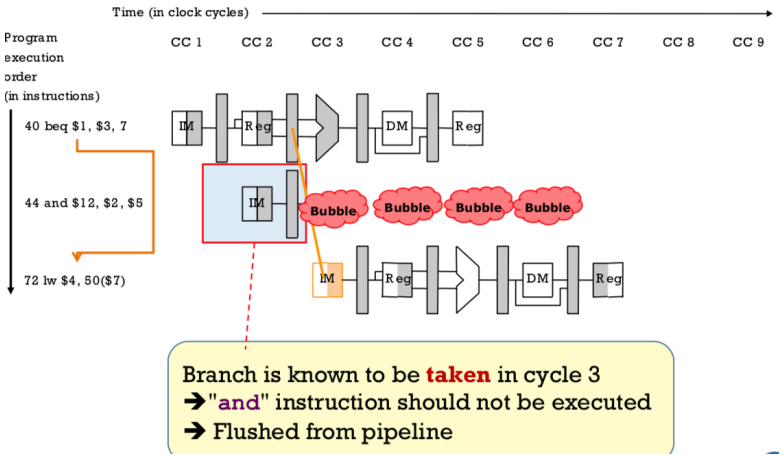


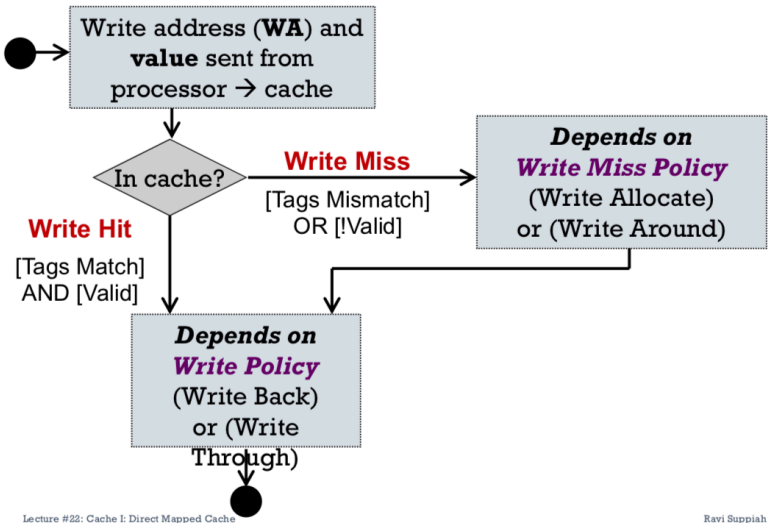


6.1 REDUCE STALLS: EARLY BRANCH (3/3)



6.2 BRANCH PREDICTION: WRONG PREDICTION

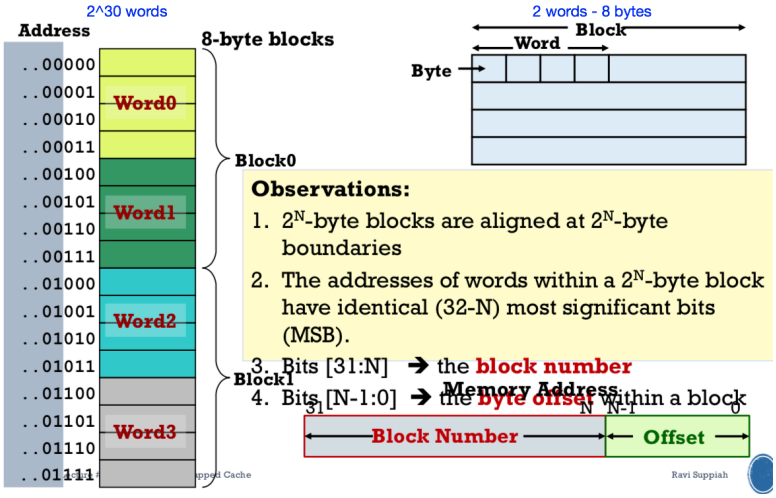




- **Hit**: Data is in cache (e.g., **X**)
- Hit rate: Fraction of memory accesses that hit
- Hit time: Time to access cache
- **Miss**: Data is not in cache (e.g., **Y**)
- Miss rate = 1 – Hit rate
- Miss penalty: Time to replace cache block + hit time
- Hit time < Miss penalty

Average Access Time = Hit rate x Hit Time + (1-Hit rate) x Miss penalty

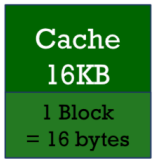
3. MEMORY TO CACHE MAPPING (2/2)



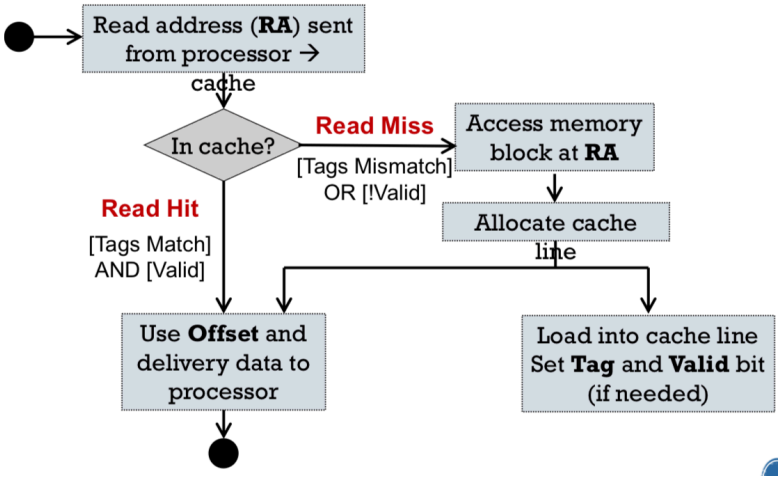
Cache Block size = 2^N bytes

Number of cache blocks = 2^M

Offset, N = 4 bits
Block Number = 32 – 4 = **28 bits**
Check: Number of Blocks = 2^{28}



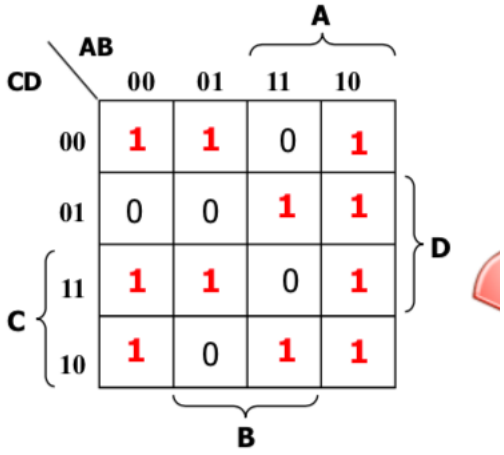
Number of Cache Blocks
= 16KB / 16bytes = 1024 = 2^{10}
Cache Index, M = 10bits
Cache Tag = 32 – 10 – 4 = **18 bits**



7PIs, 4EPIs

How mar

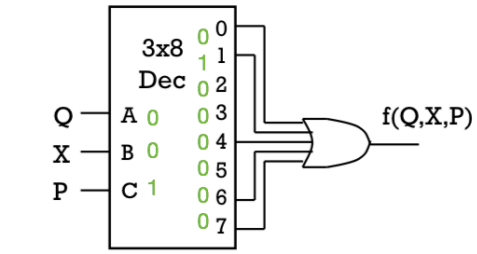
How mar



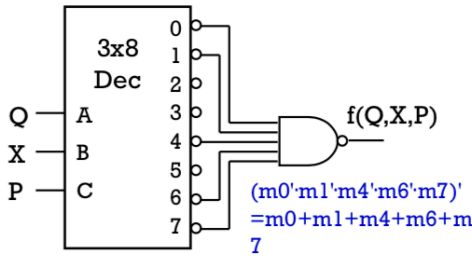
- Exercise: What modifications should be made to provide an ENABLE input for the 3×8 decoder and the 4×16 decoder created in the previous two examples?
- Exercise: How to construct a 4×16 decoder using five 2×4 decoders with enable?

2. DECODERS: IMPLEMENTING FUNCTIONS REVISIT (2/2)

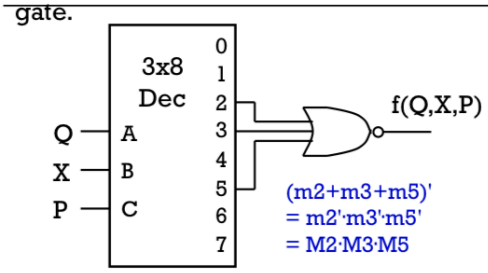
$f(Q,X,P) = \Sigma m(0,1,4,6,7) = \prod M(2,3,5)$



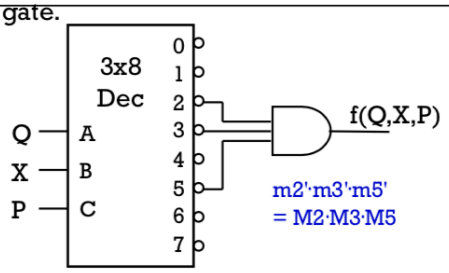
(a) Active-high decoder with OR gate.



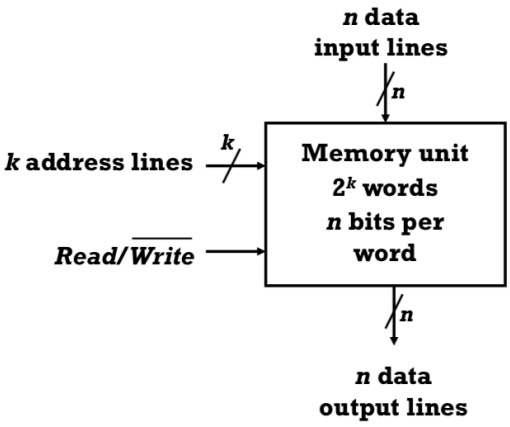
(b) Active-low decoder with NAND gate.



(c) Active-high decoder with NOR gate.



(d) Active-low decoder with AND gate.



- **Write** operation:
  - Transfers the address of the desired word to the address lines.
  - Transfers the data bits (the word) to be stored in memory to the data input lines.
  - Activates the *Write* control line (set *Read/Write* to 0).
- **Read** operation:
  - Transfers the address of the desired word to the address lines.
  - Activates the *Read* control line (set *Read/Write* to 1).

Memory Enable	*Read/Write*	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

- Two types of RAM
- Static RAMs use flip-flops as the memory cells.
- Dynamic RAMs use capacitor charges to represent data. Though simpler in circuitry, they have to be constantly refreshed.