তৃতীয় অধ্যায়
সংখ্যা পদ্ধতি ও ডিজিটাল ডিভাইস
Number Systems and Digital Devices

আন্তর্জাতিক রবোটিক প্রতিযোগিতায় বাংলাদেশের স্কুলের শিক্ষার্থীদের অংশগ্রহণ :
ডিজিটাল ডিভাইস ব্যবহারের একটি উদাহরণ

মানব সভ্যতার ইতিহাসে বিজ্ঞান এবং প্রযুক্তি অনেক বড় ভূমিকা পালন করেছে। আমরা সবাই জানি আধুনিক সভ্যতার ইতিহাসে কম্পিউটার এবং তার সাথে সম্পর্কযুক্ত অন্যান্য ইলেকট্রনিক যন্ত্রপাতির অবদান সবচাইতে বেশি। একসময় যে কম্পিউটারটি বসানোর জন্য একটি পুরো বিল্ডিংয়ের প্রয়োজন হতো এখন তার চাইতেও শক্তিশালী একটি কম্পিউটার ব্যবহার করে তৈরি একটি মোবাইল ফোন আমরা আমাদের পকেটে নিয়ে ঘুরে বেড়াই। এই কম্পিউটার এবং তার সাথে আনুষাঙ্গিক যন্ত্রপাতি ইলেকট্রনিক্সের যে শাখার উপর নির্ভর করে গড়ে উঠেছে সেটি হচ্ছে ডিজিটাল ইলেকট্রনিক্স। এই অত্যন্ত গুরুত্বপূর্ণ শাখাটি দুই ভিত্তিক বাইনারি সংখ্যা এবং বুলিয়ান এলজেবরা নামে বিস্ময়করভাবে সহজ একটি গাণিতিক কাঠামো দিয়ে ব্যাখ্যা করা হয়। এই অধ্যায়ে শিক্ষার্থীদের সেই বিষয়গুলোর সাথে পরিচয় করিয়ে দেয়া হবে।

এ অধ্যায় পাঠ শেষে শিক্ষার্থীরা—

- সংখ্যা আবিষ্কারের ইতিহাস বর্ণনা করতে পারবে;
- সংখ্যা পদ্ধতির ধারণা ব্যাখ্যা করতে পারবে;
- সংখ্যা পদ্ধতির প্রকারভেদ বর্ণনা করতে পারবে;
- বিভিন্ন ধরনের সংখ্যা পদ্ধতির আন্তঃসম্পর্ক নির্ণয় করতে পারবে;
- বাইনারি যোগ-বিয়োগ সম্পন্ন করতে পারবে;
- চিহ্নযুক্ত সংখ্যার ধারণা ব্যাখ্যা করতে পারবে;
- ২ -এর পরিপূরক নির্ণয় করতে পারবে;
- কোডের ধারণা ব্যাখ্যা করতে পারবে;
- বিভিন্ন প্রকার কোডের তুলনা করতে পারবে;
- বুলিয়ান অ্যালজেবরার ধারণা ব্যাখ্যা করতে পারবে;
- বুলিয়ান উপপাদ্যসমূহ প্রমাণ করতে পারবে;
- লজিক অপারেটর ব্যবহার করে বুলিয়ান অ্যালজেবরার ব্যবহারিক প্রয়োগ করতে পারবে;
- বুলিয়ান অ্যালজেবরার সাথে সম্পর্কিত ডিজিটাল ডিভাইসসমূহের কর্মপদ্ধতি বিশ্লেষণ করতে পারবে।

৩.১ সংখ্যা পদ্ধতি আবিষ্কারের ইতিহাস (History of Inventing Numbers)

আমাদের দৈনন্দিন জীবনে আমরা প্রতিনিয়ত ভাষা এবং একই সাথে সংখ্যাকেও ব্যবহার করি। আমাদের প্রয়োজনের কারণে ভাষার সাথে সাথে আমরা সংখ্যা পদ্ধতি আবিষ্কার করেছি। সত্যি কথা বলতে কী অনেক প্রাণী এবং পাখিও অল্প কিছু গুণতে পারে। শুনে অবাক হয়ে যেতে হয় যে এখনো পৃথিবীর গহিন অরণ্যে এমন আদিবাসী মানুষ আছে যাদের জীবনে সংখ্যার বিশেষ প্রয়োজন হয় না বলে সেভাবে গুনতে পারে না। ব্রাজিলের পিরাহা নামের আদিবাসীরা এক এবং দুই থেকে বেশি গুনতে পারে না। এর চাইতে বেশি যে কোনো সংখ্যা হলেই তারা বলে 'অনেক'।

আদিম মানুষ যখন শিকারী হিসেবে বনে-জঙ্গলে ঘুরে বেড়াত তখন হিসেব রাখা বা গোনার সেরকম প্রয়োজন ছিল না। যখন তারা কৃষিকাজ করার জন্য স্থিতু হয়েছে, গবাদি পশু পালন করতে শুরু করেছে, শস্যক্ষেত্রে চাষাবাদ করেছে, গ্রাম, নগর-বন্দর গড়ে তুলেছে, রাজস্ব আদায় করা শুরু করেছে তখন থেকে গোনার প্রয়োজন শুরু হয়েছে। সেজন্য সংখ্যা পদ্ধতির ইতিহাস এবং সভ্যতার ইতিহাস খুবই ঘনিষ্ঠভাবে সম্পর্কিত। আমাদের প্রয়োজনের কারণে এখন আমরা অনেক বড় বড় সংখ্যা ব্যবহার করতে পারি, গণিতের সাহায্যে সেগুলো নানাভাবে প্রক্রিয়া করতে পারি।

আদিম কালে মানুষেরা গাছের ডাল বা হাড়ে দাগ কেটে কিংবা কড়ি, শামুক বা নুড়ি পাথর সংগ্রহ করে সংখ্যার হিসাব রেখেছে। তবে যখন আরো বড় সংখ্যা আরো বেশি স্থায়ীভাবে সংরক্ষণ করার প্রয়োজন হয়েছে তখন সংখ্যার একটি লিখিত রূপ বা চিহ্ন সৃষ্টি করে নিয়েছে। প্রায় পাঁচ হাজার বছর আগে মোটামুটি একই সময়ে সুমেরিয়ান-ব্যবলিয়ান এবং মিশরীয় সভ্যতার শুরু হয় এবং এই দুই জায়গাতেই সংখ্যার প্রথম লিখিত রূপ পাওয়া গেছে। সুমেরিয়ান-ব্যবলিয়ান সংখ্যা ছিল ষাটভিত্তিক এবং মিশরীয় সংখ্যা ছিল দশভিত্তিক। ব্যবলিয়ান সংখ্যা পদ্ধতির রেশ পৃথিবীতে এখনো রয়ে গেছে, আমরা মিনিট এবং ঘণ্টার হিসেব করি ষাট দিয়ে এবং কোণের পরিমাপ করি ষাটের গুণিতক দিয়ে। সুমেরিয়ান-ব্যবলিয়ান সংখ্যা পদ্ধতিতে স্থানীয় মান ছিল, মিশরীয় সংখ্যা পদ্ধতিতে ছিল না। দুই পদ্ধতিতেই কোনো কিছু না থাকলে সেটি বোঝানোর জন্য চিহ্ন ব্যবহার করা হতো কিন্তু সেটি মোটেও গাণিতিক সংখ্যা শূন্য ছিল না।

পরবর্তীকালে আরো তিনটি সভ্যতার সাথে সাথে সংখ্যা পদ্ধতি গড়ে উঠে, সেগুলো হচ্ছে মায়ান সভ্যতা, চীন সভ্যতা এবং ভারতীয় সভ্যতা। মায়ান সংখ্যা পদ্ধতি ছিল কুড়িভিত্তিক, চীন এবং ভারতীয় সংখ্যা পদ্ধতি ছিল দশভিত্তিক। (আমাদের দেশে যেসব মানুষ লেখাপড়ার সুযোগ পায়নি তারা কাজ চালানোর জন্য মৌখিকভাবে কুড়িভিত্তিক এক ধরনের সংখ্যা ব্যবহার করে থাকে।) মায়ান এবং ভারতীয় সংখ্যা পদ্ধতিতে স্থানীয় মান ব্যবহার করে। প্রয়োজনের কারণে সব সংখ্যা পদ্ধতিতেই শূন্যের জন্য একটি চিহ্ন থাকলেও প্রকৃত অর্থে শূন্যকে একটি সংখ্যা হিসেবে ধরে সেটিকে সংখ্যা পদ্ধতিতে নিয়ে এসে গণিতে ব্যবহার করে ভারতীয়রা এবং এই শূন্য আবিষ্কারকে আধুনিক গণিতের একটি অন্যতম যুগান্তকারী আবিষ্কার হিসেবে বিবেচনা করা হয়। মায়ান এবং চীন সংখ্যা পদ্ধতি মাত্র দুই-তিনটি (চিত্র 3.1) চিহ্ন ব্যবহার করে লেখা হতো। কিন্তু হাতে লেখার সময় পাশাপাশি অসংখ্য চিহ্ন বসানোর বিড়ম্বনা থেকে বাঁচার জন্য ভারতীয় সংখ্যা পদ্ধতিতে 1 থেকে 9 পর্যন্ত নয়টি এবং শূন্যের জন্য একটি চিহ্ন- এভাবে দশটি চিহ্ন ব্যবহার করতে শুরু করে। আমরা এই চিহ্নগুলোকে অঙ্ক বা Digit বলি।

২৫০০ বছর আগে গ্রিকরা ব্যবিলোনিয়ান এবং মিশরীয়দের সংখ্যা পদ্ধতির উপর ভিত্তি করে তাদের পূর্ণাঙ্গ ১০ ভিত্তিক সংখ্যা পদ্ধতি গড়ে তুলেছিল। রোমানরা গ্রিক সভ্যতার পতন ঘটানোর পর গণিতের অভূতপূর্ব বিকাশ থেমে যায়। রোমান সাম্রাজ্যে গণিতের সেরকম প্রয়োজন ছিল না। তাদের সংখ্যাগুলোতে আলাদা রূপ ছিল না এবং রোমান অক্ষর দিয়ে সেগুলো প্রকাশ করা হতো। অনাবশ্যকভাবে জটিল এবং অবৈজ্ঞানিক রোমান সংখ্যা এখনো বেঁচে আছে এবং ঘড়ির ডায়াল বা অন্যান্য জায়গায় মাঝে মাঝে আমরা তার ব্যবহার দেখতে পাই।

Hindu-Arabic	Roman	Greek	Egyptian	Babylonian	Chinese	Mayan
0					〇	
1	I	A	𓏺	𒁹	一	•
2	II	B	𓏺𓏺	𒁹𒁹	二	••
3	III	Γ	𓏺𓏺𓏺	𒁹𒁹𒁹	三	•••
4	IV	Δ	𓏺𓏺𓏺𓏺	𒁹𒁹𒁹𒁹	四	••••
5	V	E	𓐁	𒐈	五	—
6	VI	F	𓐁𓏺	𒐈𒁹	六	—•
7	VII	Z	𓐁𓏺𓏺	𒐈𒁹𒁹	七	—••
8	VIII	H	𓐁𓏺𓏺𓏺	𒐈𒁹𒁹𒁹	八	—•••
9	IX	Θ	𓐁𓏺𓏺𓏺𓏺	𒐈𒁹𒁹𒁹𒁹	九	—••••
10	X	I	∩	𒌋	十	=
50	L	N	∩∩∩	𒌋𒌋𒌋𒌋𒌋	五十	≡
100	C	P	𓍢	𒐐	一百	

চিত্র 3.1 : বিভিন্ন প্রাচীন সংখ্যা

ইসলামি সভ্যতার বিকাশ হওয়ার পর ভারতীয় সংখ্যা পদ্ধতি আরবদের মাধ্যমে ইউরোপে ছড়িয়ে পড়ে, যেটি আমাদের আধুনিক দশমিক সংখ্যা পদ্ধতি। এই সংখ্যা পদ্ধতিকে Hindu-Arabic সংখ্যা পদ্ধতি বলে। এখানে উল্লেখ্য যে শূন্য ব্যবহারের ফলে সংখ্যা পদ্ধতিতে বিস্ময়কর অগ্রগতি হলেও খ্রিষ্টীয় শাসকেরা শূন্যকে শয়তানের রূপ বিবেচনা করায় দীর্ঘদিন সেটাকে ঠেকিয়ে রাখার চেষ্টা করেছিল!

আমাদের হাতে দশ আঙুল থাকার কারণে দশভিত্তিক সংখ্যা গড়ে উঠলেও দুই, আট কিংবা ষোলোভিত্তিক সংখ্যাও আধুনিক প্রযুক্তিতে ব্যাপকভাবে ব্যবহার করা হয়।

৩.২ সংখ্যা পদ্ধতি (Number System)

সংখ্যাকে প্রকাশ করার এবং গণনা করার পদ্ধতিকে সংখ্যা পদ্ধতি বলে। সংখ্যাকে প্রকাশ করার জন্য বিভিন্ন প্রতীক বা চিহ্ন ব্যবহার করা হয়। এই প্রতীকগুলোকে দুটো ভিন্ন ভিন্ন পদ্ধতিতে ব্যবহার করা যায়।

৩.২.১ সংখ্যা পদ্ধতির প্রকারভেদ (Classification of Number System)

সংখ্যা পদ্ধতিকে নন-পজিশনাল এবং পজিশনাল এই দুটি মূল পদ্ধতিতে ভাগ করা যায় :

নন-পজিশনাল সংখ্যা পদ্ধতি : এই পদ্ধতিতে প্রতীক বা চিহ্নগুলো যেখানেই ব্যবহার করা হোক, তার মান একই থাকবে। রোমান সংখ্যা হচ্ছে নন-পজিশনাল (Non positional) সংখ্যার উদাহরণ। যেমন– রোমান সংখ্যায় 5 বোঝানোর জন্য V ব্যবহার করা হয়। V, VI কিংবা VII এই তিনটি উদাহরণে V তিনটি ভিন্ন জায়গায় বসেছে, কিন্তু প্রতি ক্ষেত্রেই V চিহ্নটি 5 বুঝিয়েছে। তথা পজিশনাল সংখ্যা পদ্ধতির ন্যায় V যতই ডান হতে বাম দিকে সরতে (স্থান পরিবর্তন) থাকুক না কেন তার স্থানীয় মানের (একক, দশক, শতক ইত্যাদির ন্যায়) কোন পরিবর্তন হয় না। এর কারণ হলো নন-পজিশনাল (অস্থানিক) সংখ্যা পদ্ধতিতে স্থানিক মানের অনুপস্থিতি। প্রাচীনকালে যখন সংখ্যাতত্ত্ব সেভাবে গড়ে উঠেনি তখন নন-পজিশনাল সংখ্যা পদ্ধতির প্রচলন ছিল।

পজিশনাল সংখ্যা পদ্ধতি : এই পদ্ধতিতে চিহ্ন বা প্রতীকটিকে কোন অবস্থানে ব্যবহার করা হচ্ছে তার উপর মানটি নির্ভর করে। আধুনিক সংখ্যাতত্ত্ব গড়ে উঠার পর পজিশনাল (Positional) সংখ্যা পদ্ধতির প্রচলন শুরু হয়েছে। আমাদের প্রচলিত দশমিক পদ্ধতি হচ্ছে পজিশনাল সংখ্যা পদ্ধতির উদাহরণ। কারণ 555 সংখ্যাকে ডান দিকের প্রথম অঙ্কটি 5 সংখ্যাকে বোঝালেও তার বামেরটি 50 এবং এর বামেরটি 500 সংখ্যাকে বোঝাচ্ছে। এটি 10 ভিত্তিক সংখ্যা এবং প্রত্যেকটি অবস্থানের একটি মান রয়েছে। ডান দিকের প্রথম অঙ্কটির মান 1, বামেরটি 10, এর বামেরটি 100 এভাবে আগের অবস্থান থেকে আগের অবস্থান সবসময়েই 10 গুণ বেশি। যদি এটি 8 ভিত্তিক সংখ্যা হতো তাহলে পরের অবস্থান আগের অবস্থান থেকে 8 গুণ বেশি হতো। 16 ভিত্তিক সংখ্যা হলে প্রতিটি অবস্থান আগের অবস্থান থেকে 16 গুণ বেশি হতো।

নিচে কয়েকটি পজিশনাল সংখ্যা পদ্ধতির উদাহরণ দেওয়া হলো।

বাইনারি সংখ্যা

আমরা সবাই দশভিত্তিক দশমিক সংখ্যার সাথে পরিচিত কিন্তু ডিজিটাল ইলেকট্রনিক্সের জন্য দশভিত্তিক সংখ্যা খুব কার্যকর নয়, দশটি চিহ্নের জন্য দশটি ভিন্ন ভিন্ন ভোল্টেজ ব্যবহার করে ইলেকট্রনিক যন্ত্রপাতি তৈরি করা বাস্তবসম্মত নয়। দুটি চিহ্নের জন্য দুটি ভোল্টেজ লেভেল তুলনামূলকভাবে অনেক সহজ। সেজন্য ডিজিটাল ইলেকট্রনিক্স আসলে 2 ভিত্তিক বা বাইনারি (Binary) সংখ্যার উপর ভিত্তি করে গড়ে উঠেছে।

দশমিক সংখ্যায় যেরকম 0, 1, 2, 3, 4, 5, 6, 7, 8 এবং 9- এই দশটি চিহ্ন বা অঙ্ক (Digit) ব্যবহার করে গড়ে উঠেছে, বাইনারি সংখ্যা ঠিক সেরকম 0 এবং 1 এই দুইটি অঙ্ক ব্যবহার করে গড়ে উঠেছে। তবে সে কারণে কোনো সংখ্যাকে প্রকাশ করার জন্য তুলনামূলকভাবে বেশি অঙ্ক ব্যবহার করা ছাড়া বাইনারি সিস্টেমে আর কোনো সীমাবদ্ধতা নেই। যে কোনো সংখ্যা এই বাইনারি সংখ্যা দিয়ে প্রকাশ করা যায় এবং যে কোনো গাণিতিক প্রক্রিয়া এই বাইনারি সংখ্যা দিয়ে করা সম্ভব।

বাইনারি সংখ্যাতেও প্রত্যেকটি অঙ্কের একটি স্থানীয় মান রয়েছে। দশমিক সংখ্যায় স্থানীয় মান 10^0, 10^1, 10^2 ... এভাবে বেড়েছে, বাইনারি সংখ্যাতে 2^0, 2^1, 2^2, 2^3 ... এভাবে বেড়েছে। ভগ্নাংশে প্রকাশ করার জন্য দশমিক বিন্দুর পর অঙ্কগুলো 10^{-1}, 10^{-2}, 10^{-3} ... এভাবে কমছে, ঠিক সেরকম বাইনারি সংখ্যায় বাইনারি বিন্দু (বা র‍্যাডিক্স বিন্দু)'র পর অঙ্কগুলো 2^{-1}, 2^{-2}, 2^{-3} ... এভাবে কমেছে। তুলনা করার জন্য নিচে দশমিক এবং বাইনারি সংখ্যার একটি উদাহরণ দেওয়া হলো :

দশমিক সংখ্যা									বাইনারি সংখ্যা								
10^4	10^3	10^2	10^1	10^0		10^{-1}	10^{-2}	10^{-3}	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}
↓	↓	↓	↓	↓		↓	↓	↓	↓	↓	↓	↓	↓		↓	↓	↓
2	**3**	**5**	**0**	**1**	**.**	**2**	**3**	**7**	**1**	**1**	**0**	**0**	**1**	**.**	**1**	**1**	**0**
↑				↑				↑	↑				↑				↑
MSD				দশমিক বিন্দু				LSD	MSB				বাইনারি বিন্দু				LSB

এখানে MSD ও LSD বলতে বোঝানো হয় Most ও Least Significant Digit এবং MSB ও LSB বলতে বোঝানো হয় Most ও Least Significant Bit। দশমিক সংখ্যাটির মতো বাইনারি সংখ্যাটির মান বের করার জন্য আসলে বাইনারি সংখ্যার সাথে তার স্থানীয় মান গুণ দিয়ে সব যোগ করে নিতে হবে।

$$11001.110_2 = 1\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3}$$
$$= 16 + 8 + 4 + 0 + 0 + 1 + 0.5 + 0.25 + 0$$
$$= 25.75_{10}$$

এখানে বাইনারি সংখ্যার জন্য সাবস্ক্রিপ্টে যে 2 এবং দশমিক সংখ্যার জন্য 10 লেখা হয়েছে সেগুলো হচ্ছে তাদের ভিত্তি বা বেজ (Base)। কোনো সংখ্যাপদ্ধতিতে একটি সংখ্যা বোঝানোর জন্য সর্বমোট যতগুলো অঙ্ক ব্যবহার করতে হয়, সেটি হচ্ছে সংখ্যাটির ভিত্তি বা বেজ। দশমিক পদ্ধতির জন্য বেজ 10, বাইনারির জন্য বেজ 2, ঠিক সেরকম অক্টাল এবং হেক্সাডেসিমেল নামেও সংখ্যা পদ্ধতির ব্যবহার করা হয়, যাদের বেজ যথাক্রমে 8 এবং 16. সাধারণভাবে একটি সংখ্যা পদ্ধতির জন্য সবসময় তার বেজ লেখার প্রয়োজন হয় না তবে একই সাথে একাধিক সংখ্যা পদ্ধতি থাকলে সংখ্যাটির পাশে তার বেজ লেখা থাকলে বিভ্রান্তির সুযোগ থাকে না।

এই অধ্যায়ে আমরা একটি ডিজিটাল সিস্টেমের জন্য প্রয়োজনীয় সংখ্যা পদ্ধতি গড়ে তুলব যেখানে ভগ্নাংশের প্রয়োজন হবে না, কাজেই আমরা আমাদের সকল আলোচনা শুধু পূর্ণ সংখ্যার মাঝে সীমাবদ্ধ রাখব।

টেবিল: 3.1

স্থানীয় মান				দশমিক সংখ্যা
$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

3.1 টেবিলে বাইনারি সংখ্যা এবং দশমিক সংখ্যার পর্যায়ক্রম মানের একটা উদাহরণ দেয়া হলো।

অক্টাল সংখ্যা

অক্টাল সংখ্যার ভিত্তি বা বেজ হচ্ছে 8 এবং এই সংখ্যার জন্য যে আটটি অঙ্ক ব্যবহার করা হয় সেগুলো হচ্ছে 0, 1, 2, 3, 4, 5, 6 এবং 7। 3.2 টেবিলে 0 থেকে 16 পর্যন্ত অক্টাল সংখ্যা লিখে দেখানো হলো :

টেবিল : 3.2

দশমিক সংখ্যা	অক্টাল সংখ্যা	দশমিক সংখ্যা	অক্টাল সংখ্যা
0	0	8	10
1	1	9	11
2	2	10	12
3	3	11	13
4	4	12	14
5	5	13	15
6	6	14	16
7	7	15	17

হেক্সাডেসিমেল সংখ্যা

হেক্সাডেসিমেলের ভিত্তি হচ্ছে 16। কাজেই এটাকে প্রকাশ করার জন্য 16 টি অঙ্ক প্রয়োজন। ডেসিমেল দশটি সংখ্যা 0 থেকে 9 পর্যন্ত, এর পরের ৬টি অঙ্কের জন্য A, B, C, D, E এবং F এই ইংরেজি বর্ণকে ব্যবহার করা হয়। ৩.৩ টেবিলে দশমিক সংখ্যা এবং তার হেক্সাডেসিমেল রূপটি দেখানো হলো। একই টেবিলে হেক্সাডেসিমেল সংখ্যাগুলোর জন্য তার বাইনারি রূপটিও দেখানো হয়েছে। প্রতিটি হেক্সাডেসিমেল অংকের জন্য চারটি করে বাইনারি বিটের প্রয়োজন হয়। সে কারণে হেক্সাডেসিমেল 10 কে বাইনারি 10000 না লিখে 00010000 হিসেবে লেখা হয়েছে।

টেবিল: 3.3

দশমিক সংখ্যা	হেক্সাডেসিমেল সংখ্যা	বাইনারি সংখ্যা	অক্টাল সংখ্যা
0	0	0000	0
1	1	0001	1
2	2	0010	2
3	3	0011	3
4	4	0100	4
5	5	0101	5
6	6	0110	6
7	7	0111	7
8	8	1000	10
9	9	1001	11
10	A	1010	12
11	B	1011	13
12	C	1100	14
13	D	1101	15
14	E	1110	16
15	F	1111	17
16	10	00010000	20

৩.২.২ সংখ্যা পদ্ধতির রূপান্তর (Conversion of Numbers)

বাইনারি থেকে দশমিক

আমরা বাইনারি সংখ্যাকে দশমিক সংখ্যায় এবং দশমিক সংখ্যাকে বাইনারি সংখ্যায় রূপান্তর করতে পারি। নিচে বাইনারি সংখ্যাকে দশমিক সংখ্যয় রূপান্তর করার আরেকটি উদাহরণ দেয়া হলো।

$$101101_2 = 1\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$$
$$= 32 + 0 + 8 + 4 + 0 + 1$$
$$= 45_{10}$$

দশমিক থেকে বাইনারি

ঠিক একইভাবে একটি দশমিক সংখ্যাকে বাইনারি সংখ্যায় রূপান্তর করতে হলে দশমিক সংখ্যাটিকে প্রথমে 2 -এর পাওয়ারের যোগফল হিসেবে লিখতে হবে। যেরকম :

$$76 = 64 + 8 + 4 = 2^6 + 2^3 + 2^2$$

বাইনারি সংখ্যায় যেহেতু স্থানীয় মান রয়েছে তাই প্রত্যেকটি স্থানীয় মানকে দেখাতে হবে। যেগুলো নাই তার জন্য 0 ব্যবহার করতে হবে।

$$76_{10} = 2^6 + 0 + 0 + 2^3 + 2^2 + 0 + 0 = 1001100_2$$

তবে যে কোনো সংখ্যাকে 2-এর পাওয়ারের যোগফল হিসেবে বের করার একটি সহজ উপায় হচ্ছে ক্রমাগত 2 দিয়ে ভাগ করে যাওয়া। যতক্ষণ পর্যন্ত ভাগফল শূন্য না হবে ততক্ষণ পর্যন্ত 2 দিয়ে ভাগ করে যেতে হবে। ভাগশেষগুলো LSB থেকে শুরু করে ক্রমান্বয়ে MSB পর্যন্ত বাইনারি সংখ্যাগুলো বের করে দেবে। যেরকম 25 -এর জন্য :

25 কে 2 দিয়ে ভাগ দিতে হবে $\frac{25}{2}$ ভাগফল 12 এবং ভাগশেষ 1 (LSB)

ভাগফল 12 কে 2 দিয়ে ভাগ দিতে হবে $\frac{12}{2}$ ভাগফল 6 এবং ভাগশেষ 0

ভাগফল 6 কে 2 দিয়ে ভাগ দিতে হবে $\frac{6}{2}$ ভাগফল 3 এবং ভাগশেষ 0

ভাগফল 3 কে 2 দিয়ে ভাগ দিতে হবে $\frac{3}{2}$ ভাগফল 1 এবং ভাগশেষ 1

ভাগফল 1 কে 2 দিয়ে ভাগ দিতে হবে $\frac{1}{2}$ ভাগফল 0 এবং ভাগশেষ 1

বাইনারি সংখ্যা : (MSB) 1 1 0 0 1

পদ্ধতিটা বুঝে গেলে আমরা সেটাকে আরো সংক্ষেপে লিখতে পারি। যেরকম 37 -এর জন্য আমরা লিখব :

2	37	ভাগশেষ
2	18 → 1	↑ (LSB)
2	9 → 0	
2	4 → 1	
2	2 → 0	
2	1 → 0	
	0 → 1	(MSB)

বাইনারি সংখ্যা: 100101

এই পদ্ধতিটি আমরা দশমিক থেকে অন্য যে কোনো ভিত্তিক সংখ্যায় রূপান্তর করার জন্যও ব্যবহার করতে পারি। শুধু 2 -এর পরিবর্তে যে ভিত্তিক সংখ্যায় রূপান্তর করতে চাই সেই সংখ্যাটি দিয়ে ভাগ করতে হবে।

ভগ্নাংশের ক্ষেত্রে দশমিক হতে বাইনারিতে রূপান্তর :

দশমিক ভগ্নাংশকে ২ দ্বারা গুণ করতে হয় এবং গুণফলের পূর্ণ অংশটি সংরক্ষিত রেখে ভগ্নাংশকে পুনরায় ২ দ্বারা গুণ করতে হয়, এরপর পূর্ণ অংক হিসেবে প্রাপ্ত অঙ্কগুলো প্রাপ্তির ক্রমানুসারে পাশাপাশি লিখে দশমিক সংখ্যার সমকক্ষ বাইনারি সংখ্যা পাওয়া যায়।

উদাহরণ : $(0.46)_{10}$ কে বাইনারিতে রূপান্তর কর।

সমাধান :

প্রথম পদ্ধতি-

		.46
		× 2
MSB	0	.92
		× 2
	1	.84
		× 2
	1	.86
		× 2
	1	.36
		× 2
LSB	0	.72

$\therefore (0.46)_{10} = (0.01110...)_2$

দ্বিতীয় পদ্ধতি-

.46	.92	.84	.86	.36
× 2	× 2	× 2	× 2	× 2
0.92	1.84	1.86	1.36	0.72
↓	↓	↓	↓	↓
0	1	1	1	0
MSB				LSB

$(0.46)_{10} = (0.01110...)_2$

দশমিক থেকে অক্টাল

এখানে আমরা আগে দেখানো ডেসিমেল থেকে বাইনারি সংখ্যায় রূপান্তরের পদ্ধতিটি ব্যবহার করব, তবে অক্টাল সংখ্যার বেজ যেহেতু 8 তাই 2 দিয়ে ক্রমান্বয়ে ভাগ করার পরিবর্তে 8 দিয়ে ক্রমান্বয়ে ভাগ করা হবে। যেমন- 710 কে অক্টালে রূপান্তর করার জন্য লিখব :

8	710	ভাগশেষ
8	88 → 6	↑ (LSD)
8	11 → 0	
8	1 → 3	
	0 → 1	(MSD)

অক্টাল সংখ্যা : 1306

ভগ্নাংশের ক্ষেত্রে দশমিক হতে অক্টালে রূপান্তর :

দশমিক ভগ্নাংশকে ৮ দ্বারা গুণ করতে হবে এবং প্রাপ্ত গুণফলের পূর্ণ অংশটি সংরক্ষিত রেখে গুণফলের ভগ্নাংশকে পুনরায় ৮ দ্বারা গুণ করতে হবে এরপর পূর্ণ অংক হিসেবে প্রাপ্ত অংকগুলো প্রাপ্তির ক্রমানুসারে পাশাপাশি লিখে দশমিক সংখ্যাটির সমকক্ষ অক্টাল সংখ্যা পাওয়া যায়।

উদাহরণ: $(123.45)_{10}$ কে অক্টালে রূপান্তর কর।

সমাধান :

পূর্ণ অংশ- **অবশিষ্ট (Remainer)**

8	123		LSB
8	15	3	↑
8	1	7	↑
	0	1	MSB

$\therefore (123.45)_{10} = (173.34631.....)_8$

ভগ্নাংশ

MSB ↓ LSB

	.45
	× 8
3	.60
	× 8
4	.80
	× 8
6	.40
	× 8
3	.20
	× 8
1	.60

ভগ্নাংশের ক্ষেত্রে অক্টাল হতে দশমিকে রূপান্তর :

ভগ্নাংশের পর হতে অক্টাল বিন্দুর পর হতে -1, -2, -3 ইত্যাদি দ্বারা অবস্থান চিহ্নিত করে নিতে হয়। এর পর প্রতিটি ডিজিটকে 8^n দ্বারা গুণ করে গুণফলকে যোগ করে দশমিক সংখ্যা পাওয়া যায়। সেখানে n হলো -1, -2, -3 ইত্যাদি।

উদাহরণ : $(123.45)_8$ কে দশমিক সংখ্যায় রূপান্তর কর।

সমাধান :

2 1 0 -1 -2

1 2 3 . 4 5

$1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 + 4 \times 8^{-1} + 5 \times 8^{-2}$

$= 64 + 16 + 3 + 4/8 + 5/8^2$

$= 83 + 4/8 + 5/64$

$= 83 + 0.5 + 0.078125$

$= 83.578125$

$\therefore (123.45)_8 = (83.578125)_{10}$

নিজে কর: ফাঁকা ঘরগুলোতে দশমিক 71 থেকে 90 পর্যন্ত অক্টাল সংখ্যায় লিখ এবং অক্টাল 41 থেকে 60 পর্যন্ত দশমিক সংখ্যায় লিখ।

দশমিক	অক্টাল
71	107
72	110
73	
74	
75	

দশমিক	অক্টাল
76	
77	
78	
79	
80	

অক্টাল	দশমিক
41	
42	
43	
44	36
45	37

অক্টাল	দশমিক
46	
47	
50	
51	
52	

অক্টাল থেকে বাইনারি

অক্টাল সংখ্যার একটি বড় সুবিধা হচ্ছে যে, যেকোনো সংখ্যাকে খুব সহজে বাইনারিতে রূপান্তর করা যায়। অক্টাল সংখ্যার অঙ্কগুলো হচ্ছে 0, 1, 2, 3, 4, 5, 6 এবং 7 এবং এই প্রত্যেকটি সংখ্যাকে তিন বিট বাইনারি সংখ্যা হিসেবে প্রকাশ করা যায়।

Octal :	0	1	2	3	4	5	6	7
Binary :	000	001	010	011	100	101	110	111

এই রূপান্তরটি ব্যবহার করে যে কোনো অক্টাল সংখ্যাকে তার জন্য প্রযোজ্য তিনটি বাইনারি সংখ্যা দিয়ে প্রকাশ করলেই পুরো অক্টাল সংখ্যার বাইনারি রূপ বের হয়ে যাবে। যেমন :

$$\begin{array}{rccccc} & 4 & 1 & 2 & & \\ 412_8 = & 100 & 001 & 010 & = 100001010_2 & \\ & 1 & 4 & \cdot\ 5 & 3 & \\ 14.53_8 = & 001 & 100 & 101 & 011 & = 001100.101011_2 \end{array}$$

তবে নিচের উদাহরণে সর্ব বামে দুটি 0 রয়েছে এবং সেই দুটো লেখার প্রয়োজন নেই। তাই-

$$14.53_8 = 1100.101011_2$$

বাইনারি থেকে অক্টাল

একই পদ্ধতির বিপরীত প্রক্রিয়া করে আমরা খুব সহজে যে কোনো বাইনারি সংখ্যাকে অক্টাল সংখ্যায় রূপান্তর করতে পারব। প্রথমে বাইনারি সংখ্যার অঙ্কগুলো তিনটি তিনটি করে ভাগ করে নিতে হবে। সর্ববামে যদি তিনটির কম অঙ্ক থাকে তাহলে এক বা দুইটি শূন্য বসিয়ে তিন অঙ্ক করে নিতে হবে। তারপর প্রতি তিনটি বাইনারি অঙ্কের জন্য নির্ধারিত অক্টাল সংখ্যাগুলো বসিয়ে নিতে হবে। যেমন :

$$\begin{array}{rcccccc} 10100101011_2 = & 010 & 100 & 101 & 011 & = 2453_8 \\ & 2 & 4 & 5 & 3 & \end{array}$$

এখানে তিনটি করে মেলানোর জন্য সর্ব বামে একটি বাড়তি শূন্য বসানো হয়েছে।

হেক্সাডেসিমেল থেকে ডেসিমেল

হেক্সাডেসিমেল থেকে ডেসিমেলে রূপান্তর করার জন্য আমরা অঙ্কগুলোকে তাদের নির্দিষ্ট স্থানীয় মান দিয়ে গুণ করে একসাথে যোগ করে নেব। হেক্সাডেসিমেলের বেজ যেহেতু 16 তাই স্থানীয় মান হবে যথাক্রমে 16^0, 16^1, 16^2, 16^3 এরকম :

$$356_{16} = 3\times16^2 + 5\times16^1 + 6\times16^0 = 768 + 80 + 6 = 854_{10}$$
$$2AF_{16} = 2\times16^2 + 10\times16^1 + 15\times16^0 = 512 + 160 + 15 = 687_{10}$$

লক্ষ করতে হবে যে এখানে হেক্সাডেসিমেল A -এর পরিবর্তে 10 এবং F -এর পরিবর্তে 15 বসানো হয়েছে।

ভগ্নাংশের ক্ষেত্রে হেক্সাডেসিমেল হতে দশমিকে রূপান্তর :

ভগ্নাংশের ক্ষেত্রে হেক্সাডেসিমেল বিন্দুর পর হতে -1, -2, -3 ইত্যাদি দ্বারা অবস্থান চিহ্নিত করে নিতে হয়। এরপর প্রতিটি ডিজিটকে 16^n দ্বারা গুণ করে গুণফলকে যোগ করলে দশমিক সংখ্যা পাওয়া যায়। যেখানে n হচ্ছে -1, -2, -3 ইত্যাদি।

উদাহরণঃ $(AB.CD)_{16}$ কে দশমিকে রূপান্তর কর।

সমাধানঃ

$$A\,(10)\times16^1 + B\,(11)\times16^0 + C\,(12)\times16^{-1} + D\,(13) + 16^{-2}$$
$$= 160 + 11 + \frac{12}{16} + \frac{13}{16^2}$$
$$= 171 + \frac{3}{4} + \frac{13}{256}$$

= 171 + 0.75 + 0.0507

= 171.8007

$\therefore (AB.CD)_{16} = (171.8007)_{10}$

দশমিক থেকে হেক্সাডেসিমেল

এখানেও আমরা বাইনারি কিংবা অক্টাল সংখ্যার জন্য আগে দেখানো পদ্ধতিটি ব্যবহার করব। তবে বেজ যেহেতু 16 তাই 2 কিংবা 8 দিয়ে ক্রমান্বয়ে ভাগ করার পরিবর্তে 16 দিয়ে ক্রমান্বয়ে ভাগ করা হবে। ভাগশেষ যদি 10 কিংবা তার থেকে বেশি হয় তাহলে পরিচিত ডেসিমেল অংকের পরিবর্তে যথাক্রমে A, B, C, D, E এবং F লিখতে হবে। এই পদ্ধতিতে 7106 কে হেক্সাডেসিমেলে রূপান্তর করা হয়েছে। এখানে উল্লেখ্য, ভাগশেষ হিসেবে 12 সংখ্যার জন্য C এবং 11 সংখ্যার জন্য হেক্সাডেসিমেল প্রতীক B লেখা হয়েছে।

16	7106	
16	444 - 2	(LSD)
16	27 - 12	
16	1 - 11	
	0 - 1	(MSD)

হেক্সাডেসিমেল: $1BC2_{16}$

ভগ্নাংশের ক্ষেত্রে দশমিক হতে হেক্সাডেসিমেলে রূপান্তর :

দশমিক ভগ্নাংশকে ১৬ দ্বারা গুণ করতে হবে এবং প্রাপ্ত গুণফলের পূর্ণ অঙ্কটি সংরক্ষিত রেখে গুণফলের ভগ্নাংশকে পুনরায় ১৬ দ্বারা গুণ করতে হবে তবে প্রাপ্ত ভগ্নাংশগুলো ৯ এর বেশি হলে প্রতিটি সংখ্যার সমকক্ষ হেক্সাডেসিমেল মান লিখতে হবে। এরপর পূর্ণ অঙ্ক হিসেবে প্রাপ্ত অঙ্কগুলো প্রাপ্তির ক্রমানুসারে পাশাপাশি লিখতে উক্ত দশমিক সংখ্যাটির সমকক্ষ হেক্সাডেসিমেল সংখ্যা পাওয়া যায়।

উদাহরণ : $(0.71)_{10}$ কে হেক্সাডেসিমেলে রূপান্তর কর।

সমাধান :

প্রথম পদ্ধতি-

	.71
	×16
B (11)	.36
	×16
5	.76
	×16
C (12)	.16

$\therefore (0.71)_{10} = (0.B5C...)_{16}$

বিকল্প পদ্ধতি-

.71	.36	.76
× 16	× 16	× 16
11.36	5.76	12.16
↓	↓	↓
B	5	C

$\therefore (0.71)_{10} = (0.B5C...)_{16}$

হেক্সাডেসিমেল থেকে বাইনারি

অক্টাল সংখ্যার বেলায় আমরা প্রত্যেকটি অক্টাল অঙ্কের জন্য তিন বিট বাইনারি সংখ্যা ব্যবহার করেছিলাম। হেক্সাডেসিমেলের জন্য প্রতিটি হেক্সাডেসিমেল অঙ্কের জন্য চার বিট বাইনারি সংখ্যা ব্যবহার করা হবে।

	9	F	2	3	
$9F23_{16}$ =	1001	1111	0010	0011	$= 1001111100100011_2$

সর্ববামে 0 থাকলে সেগুলোকে রাখার প্রয়োজন নেই।

বাইনারি থেকে হেক্সাডেসিমেল

এখানেও আগের মতো বাইনারি সংখ্যাগুলোকে চারটির সমন্বয় করে ভাগ করে নিতে হবে। সর্ববামে যদি চারটির কম বাইনারি অঙ্ক থাকে তাহলে সেখানে প্রয়োজনীয় সংখ্যক 0 বসিয়ে চারটির গ্রুপ করে নিতে হবে। তারপর প্রতি চারটি বাইনারি সংখ্যার জন্য নির্ধারিত হেক্সাডেসিমেল সংখ্যাটি বসিয়ে দিতে হবে।

যেরকম :

10110111000011_2 = 0010 1101 1100 0011 = $2DC3_{16}$
2 D C 3

হেক্সাডেসিমেলে যেহেতু চারটি বাইনারি অঙ্ক একটি হেক্সাডেসিমেল অঙ্ক দিয়ে প্রতিস্থাপন হয় তাই অনেক বড় বাইনারি সংখ্যা লেখার জন্য হেক্সা অথবা অক্টাল সংখ্যা ব্যবহার করা হয়।

সমস্যা : হেক্সাডেসিমেল সংখ্যা 38 থেকে শুরু করে পরবর্তী 25টি সংখ্যা লিখ। হেক্সাডেসিমেল 38-এর দশমিক মান কত?

হেক্সাডেসিমেল থেকে অক্টাল কিংবা অক্টাল থেকে হেক্সাডেসিমেলে রূপান্তর করার সবচেয়ে সহজ নিয়ম হচ্ছে, প্রথমে বাইনারিতে রূপান্তর করে নেয়া। তারপর হেক্সাডেসিমেলের জন্য চারটি করে এবং অক্টালের জন্য তিনটি করে বাইনারি অঙ্ক নিয়ে তাদের জন্য নির্ধারিত হেক্সাডেসিমেল অথবা অক্টাল সংখ্যাগুলো বেছে নেয়া। যেমন :

$B2F_{16}$ = 1011 0010 1111_2 = 101 100 101 111_2 = 5457_8
B 2 F 5 4 5 7

এখানে $B2F_{16}$ কে অক্টালে রূপান্তর করার জন্য প্রথমে সংখ্যাটির তিনটি হেক্সাডেসিমেল অঙ্কের জন্য নির্ধারিত চারটি করে বাইনারি অঙ্ক ব্যবহার করে মোট 12টি বাইনারি অঙ্কে রূপান্তর করা হয়েছে। তারপর এই 12টি বাইনারি অঙ্ককে তিনটি করে মোট 4 টি গ্রুপে ভাগ করা হয়েছে। এবারে প্রতি গ্রুপের জন্য নির্ধারিত অক্টাল অঙ্কগুলো বসিয়ে 5457_8 পাওয়া গেছে। এভাবে তিনটি অঙ্কের গ্রুপ করার সময় প্রয়োজন হলে সর্ব বামের গ্রুপটিতে একটি বা দুইটি বাড়তি 0 বসানো যেতে পারে।

৩.৩ বাইনারি যোগ বিয়োগ (Addition and Subtraction in Binary System)

বাইনারি সংখ্যা আমাদের পরিচিত দশমিক সংখ্যার মতোই একটি সংখ্যা পদ্ধতি। পার্থক্যটুকু হচ্ছে যে দশমিক সংখ্যা পদ্ধতিতে ভিত্তি 10 এবং বাইনারিতে ভিত্তি 2। কাজেই দশমিক সংখ্যা পদ্ধতিতে আমরা যেভাবে যোগ এবং বিয়োগ করতে পারি দশমিক পদ্ধতিতেও হুবহু সেভাবে যোগ এবং বিয়োগ করতে পারব। যেমন :

বাইনারি যোগ	বাইনারি বিয়োগ
101 100 101	101 100 101
11 001 001	11 001 001
1 000 101 110	10 011 100

তবে যেহেতু বাইনারি সংখ্যার সবচেয়ে বড় ব্যবহার ডিজিটাল ইলেকট্রনিক্সে তাই বাইনারি যোগ এবং বিয়োগের প্রয়োগের জন্য আলাদা কিছু পদ্ধতি ব্যবহার করা হয়। সাধারণ সংখ্যা যোগ-বিয়োগের বেলায় আমাদের কখনোই আমরা কত অঙ্কের সংখ্যা যোগ কিংবা বিয়োগ করছি সেটি আগে থেকে জানার প্রয়োজন হয় না কিন্তু ইলেকট্রনিক সার্কিট ব্যবহার করে বাইনারি যোগ-বিয়োগ করার সময় কত অঙ্কের

সংখ্যা যোগ করছি আগে থেকে জানতে হয়। কারণ সার্কিটটি যতগুলো বিট ধারণ করতে পারবে সংখ্যাটিতে তার থেকে বেশি সংখ্যক অঙ্ক থাকলে সেটি ব্যবহার করা যায় না। শুধু তাই নয় যোগ করার পর বিটের নির্ধারিত সংখ্যা থেকে বিটের সংখ্যা বেড়ে গেলে সেটিও সঠিকভাবে ফলাফল দেবে না। ডিজিটাল ইলেকট্রনিক্সে যেহেতু দুটি ভিন্ন ভিন্ন ভোল্টেজ দিয়ে বাইনারি 0 এবং 1 অঙ্ক দুটি দিয়ে প্রকাশ করা হয় তাই যাবতীয় গাণিতিক অঙ্ককও এই অঙ্ক দুটো দিয়েই প্রকাশ করতে হবে।

অনেকে মনে করতে পারে ডিজিটাল ইলেকট্রনিক্স করার জন্য বাইনারি সংখ্যা দিয়ে যোগ, বিয়োগ, গুণ এবং ভাগ এই প্রত্যেকটি প্রক্রিয়াই করার ব্যবস্থা থাকতে হয়। আসলে একটি সংখ্যাকে নেগেটিভ করা এবং যোগ করার সার্কিট থাকলেই অন্য সব গাণিতিক প্রক্রিয়া করা যায়। কোনো একটি সংখ্যা বিয়োগ করতে হলে সংখ্যাটিকে নিগেটিভ করে যোগ করতে হবে। সংখ্যা দিয়ে গুণ করার পরিবর্তে সেই নির্দিষ্ট সংখ্যক বার যোগ করলেই হয়। বার বার বিয়োগ করে ভাগের কাজ চালিয়ে নেয়া যায়। তাই আমরা দেখব একটি সংখ্যাকে নেগেটিভ করার একটি সুনির্দিষ্ট পদ্ধতি জানা থাকলে শুধু যোগ করার সার্কিট দিয়ে আমরা বিয়োগ, গুণ, এবং ভাগও করতে পারব।

৩.৪ চিহ্নযুক্ত সংখ্যা (Signed Numbers)

একটি বাইনারি সংখ্যাকে পজেটিভ বা নেগেটিভ হিসেবে দেখানোর একটি সহজ উপায় হচ্ছে MSB টিকে সাইনের জন্য নির্ধারিত করে রাখা। যদি সেটি 0 হয় তাহলে বুঝতে হবে সংখ্যাটি পজেটিভ আর যদি সেটি 1 হয় তাহলে বুঝতে হবে সংখ্যাটি নেগেটিভ। কাজেই 8 (আট) বিটের একটি সংখ্যার জন্যে 7টি বিট দিয়ে সংখ্যার মান প্রকাশ করা হবে এবং অষ্টম বিটটি সংখ্যার সাইন প্রকাশ করার জন্য আলাদাভাবে সংরক্ষিত থাকবে। এভাবে সংখ্যা প্রকাশ করার সময় আরো একটি বিষয় সবসময় মেনে চলতে হয়। সংখ্যাগুলোর বিট সংখ্যা সবচেয়ে পরিপূর্ণ রাখতে হবে -এর মাঝে ফাঁকা অংশ থাকতে পারবে না। আট বিটের সংখ্যায় +1 লেখার সময় 01 লেখা যাবে না, 00000001 লিখতে হবে। প্রথম 0টি বোঝাচ্ছে সংখ্যাটি পজেটিভ, পরের সাত বিট দিয়ে 1 লেখা হয়েছে। একইভাবে -1 লিখতে হলে 11 লেখা যাবে না 10000001 লিখতে হবে। প্রথম 1টি বোঝাচ্ছে সংখ্যাটি নেগেটিভ পরের সাতটি বিট দিয়ে সংখ্যার মান (1) প্রকাশ করা হয়েছে। এই পদ্ধতিতে কিছু পজিটিভ এবং নিগেটিভ সংখ্যা লিখে দেখানো হলো :

চার বিটের সংখ্যা :

দশমিক +2 = 0 010 (↑ সংখ্যার মান, সংখ্যার সাইন)

দশমিক –2 = 1 010 (↑ সংখ্যার মান, সংখ্যার সাইন)

আট বিটের সংখ্যা :

দশমিক +53 = 0 0110101 (↑ সংখ্যার মান, সংখ্যার সাইন)

দশমিক –77 = 1 1001101 (↑ সংখ্যার মান, সংখ্যার সাইন)

এই পদ্ধতিতে সংখ্যাকে পজেটিভ এবং নেগেটিভ হিসেবে প্রকাশ করায় একটি গুরুতর সমস্যা আছে। সমস্যাটি বোঝার জন্য আমরা নিচে চার বিটের দুটি সংখ্যা লিখছি, এক বিট সাইনের জন্য, বাকি তিন বিট মূল সংখ্যাটির মান বোঝানোর জন্য :

0000 এবং 1000

বোঝাই যাচ্ছে প্রথম সংখ্যাটি +0 এবং দ্বিতীয়টি -0 কিন্তু আমরা সবাই জানি, শূন্য (0) সংখ্যাটির পজেটিভ এবং নেগেটিভ হয় না- কিন্তু এই পদ্ধতিতে +0 এবং -0 মেনে নেয়া ছাড়া কোনো উপায় নেই। +0 এবং -0 এর অস্তিত্বটি কম্পিউটারে জটিল হিসেবে অনেক বড় সমস্যার সৃষ্টি করতে পারে।

৩.৫ ২ -এর পরিপূরক (2's Complement)

সাইন বিট দিয়ে সংখ্যার পজেটিভ এবং নেগেটিভ প্রকাশ করার জটিলতা থেকে রক্ষা পাওয়ার একটি চমৎকার পদ্ধতি রয়েছে। সেটি হচ্ছে 2 -এর পরিপূরক (2's complement) বিষয়টি বোঝার আগে আমরা নেগেটিভ সংখ্যা বলতে কী বোঝাই সেটি বুঝে নেই। একটি সংখ্যার সাথে যে সংখ্যাটি যোগ করলে যোগফল শূন্য হবে সেটিই হচ্ছে তার নেগেটিভ সংখ্যা। কাজেই আমাদেরকে কোনো একটি বাইনারি সংখ্যা দেওয়া হলে আমরা এমন আরেকটি বাইনারি সংখ্যা খুঁজে বের করব, যেটি যোগ করলে যোগফল হবে শূন্য।

আমরা আট বিটের একটি বাইনারি সংখ্যা দিয়ে শুরু করি। ধরা যাক সংখ্যাটি : 10110011। এবারে আমরা সংখ্যাটির 1 -এর পরিপূরক (1's complement) নিই অর্থাৎ প্রত্যেকটি 1 কে 0 দিয়ে এবং 0 কে 1 দিয়ে পরিবর্তন করে নিই :

মূল সংখ্যা	10110011
1 -এর পরিপূরক	01001100
সংখ্যা দুটির যোগফল	11111111

এই বাইনারি সংখ্যাটি হচ্ছে আট বিটের সর্বোচ্চ সংখ্যা। এর সাথে 1 যোগ করা হলে সংখ্যাটি আর আট বিটে সীমাবদ্ধ থাকবে না, এটি হবে 9 বিটের একটি সংখ্যা।

```
  11111111
         1
-----------
 1 00000000
```

আমরা যেহেতু 8 (আট) বিটের সংখ্যার মাঝে সীমাবদ্ধ থাকতে চাই, তাই নবম বিটকে উপেক্ষা করে আমরা বলতে পারি সংখ্যাটি 00000000 বা শূন্য। যেহেতু একটা সংখ্যার সাথে শুধু তার নেগেটিভ সংখ্যা যোগ করা হলেই যোগফল হিসেবে আমরা শূন্য পাই, তাই আমরা বলতে পারি যে কোনো বাইনারি সংখ্যার 1 কে 0 এবং 0 কে 1 দিয়ে পরিবর্তন করে (বা 1 এর পরিপূরক নিয়ে) যে সংখ্যা পাব তার সাথে 1 যোগ করে নেয়া হলে সেটি মূল বাইনারি সংখ্যার নেগেটিভ হিসেবে কাজ করবে। এই ধরণের সংখ্যাকে বলা হয় মূল সংখ্যাটির 2 -এর পরিপূরক।

আমরা এখন 10110011 -এর নিগেটিভ অথবা 2 -এর পরিপূরক বের করতে পারি :

মূল সংখ্যা	10110011
1 -এর পরিপূরক	01001100
1 যোগ	1
2 -এর পরিপূরক	01001101

কাজেই আমরা বলতে পারি, আট বিটের একটি সংখ্যা হিসেবে 01001101 হচ্ছে 10110011 এর নেগেটিভ। একটি সংখ্যাকে একবার নেগেটিভ করে আবার সেটিকে নেগেটিভ করা হয় তাহলে আমরা আগের সংখ্যাটি ফিরে পাব। আমরা আমাদের এই উদাহরণটিতে সেটি পরীক্ষা করে দেখতে পারি। 01001101কে আবার 2 -এর পরিপূরক করা হলে আমরা পাব :

মূল সংখ্যা	01001101
1 -এর পরিপূরক	10110010
1 যোগ	1
2 -এর পরিপূরক	10110011

আমরা সত্যি সত্যি মূল সংখ্যাটি ফিরে পেয়েছি, অর্থাৎ 01001101 এবং 10110011 হচ্ছে একটি আরেকটির নেগেটিভ।

এবারে একটা খুবই গুরুত্বপূর্ণ বিষয় আমাদের বিবেচনা করতে হবে। আমরা 2 -এর পরিপূরক বের করে যে কোনো বাইনারি সংখ্যাকে তার নেগেটিভ করতে পারব, কিন্তু মূল বাইনারি সংখ্যাটি শুরুতে কত ছিল সেটি কি আমরা জানি? যেমন ধরা যাক 1001 একটি চার বিটের বাইনারি সংখ্যা (যার দশমিক মান হচ্ছে 9), খুব সহজেই আমরা দেখাতে পারি 0111 হচ্ছে এর 2 -এর পরিপূরক (যার দশমিক মান হচ্ছে 7)। অর্থাৎ এই সংখ্যা দুটি একে অপরের 2 -এর পরিপূরক :

মূল সংখ্যা	1001	মূল সংখ্যা	0111
1 -এর পরিপূরক	0110	1 -এর পরিপূরক	1001
1 যোগ	1	1 যোগ	1
2 -এর পরিপূরক	0111	2 -এর পরিপূরক	1001

তাহলে আমরা প্রশ্ন করতে পারি, চার বিটের একটি সংখ্যা হিসেবে আমরা কি 1001 কে +9 ধরে নিয়ে এর 2 -এর পরিপূরক হিসেবে 0111কে -9 ধরে নেব? নাকি 0111কে +7 ধরে নিয়ে 2এর পরিপূরক হিসেবে 1001কে -7 ধরে নেব? এই বিভ্রান্তি থেকে মুক্তি পাবার জন্য একটি নিয়ম মেনে চলা হয়। নিয়মটি হচ্ছে MSB যদি 0 হয় শুধু তাহলেই সংখ্যাটি পজেটিভ হবে এবং বাইনারি সংখ্যাটি প্রকৃত মান দেখাবে। MSB যদি 1 হয় তাহলে সংখ্যাটি নেগেটিভ এবং শুধু 2 -এর পরিপূরক নিয়ে তার প্রকৃত পজেটিভ মান বের করা যাবে।

এই পদ্ধতিতে কিছু সংখ্যার নেগেটিভ রূপ বের করে দেখানো হলো :

চার বিটের উদাহরণ :		আট বিটের উদাহরণ :	
$+6_{10}$ =	0110	$+83_{10}$ =	01010011
1 -এর পরিপূরক	1001	1 -এর পরিপূরক	10101100
1 যোগ	1	1 যোগ	1
2 -এর পরিপূরক -6_{10}	1010_2	2 -এর পরিপূরক -83_{10}	10101101

উদাহরণ : 50_{10} থেকে 25_{10} সংখ্যাটি 2 -এর পরিপূরক পদ্ধতি ব্যবহার করে বিয়োগ দাও।

উত্তর :

$+25_{10}$ =	$0001\ 1001_2$
1 এর পরিপূরক	1110 0110
1 যোগ	1
2 এর পরিপূরক -25_{10}	1110 0111

$+50_{10}$ =	$0011\ 0010_2$
-25_{10} =	$1110\ 0111_2$
যোগফল	$1\ 0001\ 1001_2$

যোগফলে নবম বিটে 1 অঙ্কটি ওভারফ্লো হিসেবে চলে এসেছে, সেটিকে বিবেচনা করার প্রয়োজন নেই।

বাকি আট বিটের সংখ্যার MSB এর মান 0, যার অর্থ সংখ্যাটি পজেটিভ এবং আমরা জানি :
$0001\ 1001_2 = +25_{10}$ কাজেই উত্তরটি সঠিক।

উদাহরণ : 25_{10} থেকে 50_{10} সংখ্যাটি 2 -এর পরিপূরক পদ্ধতি ব্যবহার করে বিয়োগ দাও।

উত্তর :

$+50_{10} =$	$0011\ 0010_2$
1 -এর পরিপূরক	1100 1101
1 যোগ	1
2 -এর পরিপূরক -50_{10}	1100 1110

$+25_{10} =$	$0001\ 1001_2$
$-50_{10} =$	$1100\ 1110_2$
যোগফল	$1110\ 0111_2$

যোগফলে আট বিটের সংখ্যার MSB এর মান 1, যার অর্থ সংখ্যাটি নেগেটিভ। কাজেই 2 -এর পরিপূরক পদ্ধতি ব্যবহার করে সংখ্যাটিকে আবার নেগেটিভ করে তার পজেটিভ মান বের করতে হবে।

যোগফল	1110 0111
1 -এর পরিপূরক	0001 1000
1 যোগ	1
2 -এর পরিপূরক	0001 1001

আমরা জানি $00011001_2 = 25_{10}$ কাজেই প্রকৃত যোগফল -25_{10}, অর্থাৎ উত্তরটি সঠিক।

৩.৬ কোড (Code)

৩.৬.১ কোডের ধারণা (Concept of Code)

আমরা আগেই বলেছি কম্পিউটারের ভেতর ডিজিটাল প্রক্রিয়া চালানোর জন্য দুইটি ভিন্ন ভিন্ন ভোল্টেজ দিয়ে যাবতীয় ইলেকট্রনিক্স কাজকর্ম করা হয়। এই দুইটি ভোল্টেজের একটিকে 0 অন্যটিকে 1 হিসেবে বিবেচনা করে বাইনারি সংখ্যা হিসেবে যে কোনো সংখ্যাকে প্রক্রিয়া করা সম্ভব হয়। কিন্তু আমরা সবাই জানি কম্পিউটারে শুধু সংখ্যা প্রবেশ করিয়ে সেগুলোকে নানা ধরনের প্রক্রিয়া করলেই হয় না সেখানে নানা ধরনের বর্ণ, শব্দ, চিহ্ন এগুলোকে প্রক্রিয়া করতে হয়। কম্পিউটার যেহেতু অভ্যন্তরীণ ইলেকট্রনিক সার্কিটে 0 এবং 1 ছাড়া অন্য অভ্যন্তরীণ ইলেকট্রনিক সার্কিটে কোনো কিছু প্রক্রিয়া করতে পারে না, তাই শব্দ চিহ্ন বর্ণ তাদের সবকিছুকেই প্রথমে এই 0 এবং 1 এ রূপান্তরিত করে নিতে হয়। বর্ণ, অক্ষর, শব্দ বা চিহ্নকে এভাবে বাইনারিতে রূপান্তর করার প্রক্রিয়াকে কোডিং করা বলা হয়ে থাকে। নিচে এই ধরনের প্রচলিত কয়েকটি কোডের উদাহরণ দেওয়া হলো।

৩.৬.২ কোডের উদাহরণ (Examples of Code)

বিসিডি (BCD)

আমরা আমাদের দৈনন্দিন হিসাব নিকাশ সবসময়ই দশমিক সংখ্যা দিয়ে করে থাকি। এই সংখ্যাকে কম্পিউটারে কিংবা ইলেকট্রনিক সার্কিট দিয়ে ডিজিটাল প্রক্রিয়া করার জন্য সেগুলোকে বাইনারিতে রূপান্তর করে নিতে হয়। কিন্তু দশমিক সংখ্যার বহুল ব্যবহারের জন্য এর দশমিক রূপটি যতটুকু সম্ভব অক্ষুণ্ন রেখে বাইনারি সংখ্যায় রূপান্তর করার জন্য বিসিডি (BCD: Binary Coded Decimal) কোডিং পদ্ধতি গ্রহণ করা হয়েছে।

দশমিক	0	1	2	3	4	5	6	7	8	9
বিসিডি	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

এই পদ্ধতিতে একটি দশমিক সংখ্যার প্রত্যেকটি অঙ্ককে আলাদাভাবে চারটি বাইনারি বিট দিয়ে প্রকাশ করা হয়। যদিও চার বিটে 0 থেকে 15 এই 16টি সংখ্যা প্রকাশ করা সম্ভব, কিন্তু BCD কোডে 10 থেকে 15 পর্যন্ত এই বাড়তি ছয়টি সংখ্যা কখনোই ব্যবহার করা হয় না। দশমিক 10কে বাইনারিতে 1010 হিসেবে চার বিটে লেখা যায় কিন্তু বিসিডিতে 0001 0000 এই আট বিটের প্রয়োজন। নিচে BCD কোডের একটি উদাহরণ দেওয়া হলো :

4578_{10}	4	5	7	8
বিসিডি	0100	0101	0111	1000

উদাহরণ : 100100100110 বিসিডি কোডে লেখা একটি দশমিক সংখ্যা, সংখ্যাটি কত?

উত্তর : 100100100110 বিটগুলোকে চারটি করে বিটে ভাগ করে প্রতি চার বিটের জন্য নির্ধারিত দশমিক অঙ্কটি বসাতে হবে।

বিসিডি	1001	0010	0110
দশমিক	9	2	6

আলফানিউমেরিক কোড (Alphanumeric Code)

কম্পিউটারে সংখ্যার সাথে সাথে নানা বর্ণ, যতিচিহ্ন, গাণিতিক চিহ্ন ইত্যাদি ব্যবহার করতে হয়। যে কোডিংয়ে সংখ্যার সাথে সাথে অক্ষর, যতিচিহ্ন, গাণিতিক চিহ্ন ইত্যাদি ব্যবহার করা যায় সেগুলোতে আলফা নিউমেরিক কোড ব্যবহার করা হয়। নিচে কয়েকটি আলফা নিউমেরিক কোডের উদাহরণ দেওয়া হলো।

ই বি সি ডি আই সি (EBCDIC)

EBCDIC (Extended Binary Coded Decimal Interchange Code) একটি আট বিটের কোডিং। যেহেতু এটি আট বিটের কোড, কাজেই এখানে সব মিলিয়ে 256টি ভিন্ন ভিন্ন চিহ্ন প্রকাশ করা সম্ভব। আই বি এম নামের একটি কম্পিউটার কোম্পানি তাদের কম্পিউটারে সংখ্যার সাথে সাথে অক্ষর যতিচিহ্ন ইত্যাদি ব্যবহার করার জন্য BCD -এর সঙ্গে মিল রেখে এই কোডটি তৈরি করেছিল। 1963 এবং 1964 সালে কম্পিউটারে ইনপুট দেওয়ার পদ্ধতিটি ছিল- অনেক প্রাচীন কাগজের কার্ডে গর্ত করে ইনপুট দিতে হতো। কাজেই EBCDIC তৈরি করার সময় কাগজে গর্ত করার বিষয়টিও বিবেচনা করা হয়েছিল। সেই সময়ের কম্পিউটারে ইনপুট দেওয়ার জটিলতা এখন আর নেই, কাজেই EBCDIC কোডটিরও কোনো গুরুত্ব নেই।

অ্যাসকি (ASCII)

ASCII হচ্ছে American Standard Code for Information Interchange কথাটির সংক্ষিপ্ত রূপ। এটি সাত বিটের একটি আলফানিউমেরিক কোড। এটি প্রাথমিকভাবে টেলিপ্রিন্টারে ব্যবহার করার জন্য তৈরি করা হয়েছিল এবং পরবর্তীকালে কম্পিউটারে এটি সমন্বয় করা হয়। সাত বিটের কোড হওয়ার কারণে এখানে সব মিলিয়ে 128টি চিহ্ন প্রকাশ করা যায়। এর প্রথম 32টি কোড যান্ত্রিক নিয়ন্ত্রণের জন্য ব্যবহার করা হয়, বাকি 96টি কোড ছোট হাতের, বড় হাতের ইংরেজি অক্ষর, সংখ্যা, যতিচিহ্ন, গাণিতিক চিহ্ন ইত্যাদির জন্য ব্যবহার করা হয়। টেবিলে অ্যাসকি কোডটি দেখানো হলো। ইদানীং 16, 32 কিংবা 64 বিট কম্পিউটারের প্রচলনের জন্য সাত বিটের ASCII- তে সীমাবদ্ধ থাকার প্রয়োজন নেই বলে অষ্টম বিট যুক্ত করে Extended ASCII-তে আরো 128টি চিহ্ন নানাভাবে ব্যবহার হলেও প্রকৃত ASCII বলতে এখনো মূল 128টি চিহ্নকেই বোঝানো

হয়। টেবিলে অ্যাসকি কোডের প্রথম 32টি যান্ত্রিক নিয়ন্ত্রণের কোড (0 - 31) ছাড়া পরবর্তী 96টি (32 - 127) প্রতীক দেখানো হয়েছে।

টেবিল 3.4: অ্যাসকি টেবিল

সংখ্যা	প্রতীক	সংখ্যা	প্রতীক	সংখ্যা	প্রতীক	সংখ্যা	প্রতীক	সংখ্যা	প্রতীক	সংখ্যা	প্রতীক
32	Sp	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	Del

ইউনিকোড (Unicode)

ইউনিকোড হলো প্রাচীন মিশরীয় হায়ারোগ্লিফিক্স ভাষা থেকে শুরু করে বর্তমান সময়ের অক্ষর, বর্ণ, চিহ্ন, ইমোজি ইত্যাদির এনকোডিং পদ্ধতি। বর্তমানে পূর্বের এনকোডিং পদ্ধতি যেমন ASCII ও EBCDIC-কেও ইউনিকোডের আওতায় আনা হয়েছে। তথা পৃথিবীর প্রায় সব ভাষার লেখালেখির মাধ্যমগুলোকে ইউনিকোড পদ্ধতিতে সমন্বিত করা হয়েছে। ইউনিকোড কনসোর্টিয়াম নামক একটি সংস্থা ১৯৯১ সালে ২৪টি ভাষা নিয়ে প্রথম সংস্করণ 1.0.0 চালু করেন। ২০২০ সাথে ১৫৪টি ভাষা নিয়ে এর ১৩তম সংস্করণ চালু হয়েছে। ইউনিকোডের ৩টি বহুল প্রচলিত ফরমেট/স্ট্যান্ডার্ড রয়েছে। যথা-

১. UTF-8: এটি 8 বিটের (byte) একক। এখানে একটি অক্ষরকে 1 থেকে 4 বাইটের মধ্যে উপস্থাপন করা হয়। তথা এ ফরমেট অনুযায়ী প্রতিটি বর্ণের জন্য 0000_{16} থেকে $10FFFF_{16}$ এর মধ্যে একটি সংখ্যা নির্দিষ্ট করে দেয়া আছে। যেমন 0041_{16} হচ্ছে ইংরেজি 'A' অক্ষর এবং 0995_{16} হচ্ছে বাংলা 'ক' অক্ষর যা UTF-8 রেঞ্জের মধ্যে অবস্থিত। UTF-8 ইমেইল ও ইন্টারনেটে বহুল ব্যবহৃত এনকোডিং পদ্ধতি।

২. UTF-16: এটি 16 বিটের (shorts) একক। এখানে একটি অক্ষরকে 1 থেকে 2 বাইটের মধ্যে উপস্থাপন করা হয়। এর সাহায্যে মূলত ডেটা সংরক্ষণ ও টেক্সট প্রক্রিয়াকরণের কাজে ব্যবহৃত হয়।

৩. UTF-32: এটি 32 বিটের (longs) একক। এখানে একটি অক্ষরকে নির্ধারিত 4 বাইটের মধ্যে উপস্থাপন করা হয়। এখানে দক্ষতার সাথে অক্ষরকে ব্যবহার করা হয়।

উল্লেখ থাকে যে, UTF-8 এবং UTF-16 হচ্ছে সবচেয়ে প্রচলিত পদ্ধতি। এর মাঝে ওয়েবসাইটে ব্যবহার করার জন্য UTF-8 অধিকতর স্ট্যান্ডার্ড হয়ে দাঁড়িয়েছে। কারণ এ ক্ষেত্রে প্রতিটি বর্ণের জন্য 4 বাইট স্থান সংরক্ষণ করা থাকলেও ব্যবহারের ক্ষেত্রে UTF-8 শুধুমাত্র যতগুলো বিট প্রয়োজন হয় ততটুকু ব্যবহার করে থাকে।

টেবিল 3.5: বাংলা ইউনিকোড

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+098x		ঁ	ং	ঃ		অ	আ	ই	ঈ	উ	ঊ	ঋ	ঌ			এ
U+099x	ঐ			ও	ঔ	ক	খ	গ	ঘ	ঙ	চ	ছ	জ	ঝ	ঞ	ট
U+09Ax	ঠ	ড	ঢ	ণ	ত	থ	দ	ধ	ন		প	ফ	ব	ভ	ম	য
U+09Bx	র		ল				শ	ষ	স	হ			়	ঽ	া	ি
U+09Cx	ী	ু	ূ	ৃ	ৄ			ে	ৈ			ো	ৌ	্	ৎ	
U+09Dx								ৗ					ড়	ঢ়		য়
U+09Ex	ৠ	ৡ	ৢ	ৣ			০	১	২	৩	৪	৫	৬	৭	৮	৯
U+09Fx	ৰ	ৱ	৲	৳	৴	৵	৶	৷	৸	৹	৺	৻				

৩.৭ বুলিয়ান এলজেবরা ও ডিজিটাল ডিভাইস (Boolean Algebra and Digital Devices)

৩.৭.১ বুলিয়ান এলজেবরা (Boolean Algebra)

আমরা সবাই কম-বেশি এলজেবরার সাথে পরিচিত। বুলিয়ান এলজেবরা একটি ভিন্ন ধরনের এলজেবরা যেখানে শুধু 0 এবং 1 এর সেট {0, 1} নিয়ে কাজ করা হয়। প্রথমে দেখে মনে হতে পারে যে এলজেবরার প্রক্রিয়ার এবং তার ফলাফলে 0 কিংবা 1 -এর বাইরে কিছুই হতে পারবে না, সেটি আমাদের কী কাজে লাগবে? কিন্তু বিস্ময়ের ব্যাপার হচ্ছে ডিজিটাল ইলেক্ট্রনিক্সের পুরো জগৎটি বুলিয়ান এলজেবরাকে ভিত্তি করে গড়ে উঠেছে।

বুলিয়ান এলজেবরায় মাত্র তিনটি প্রক্রিয়া (operation) করা হয়। সেগুলো হচ্ছে পূরক (Complement), গুণ (Multiply) এবং যোগ (Add)। যেহেতু সকল প্রক্রিয়া করা হবে 0 এবং 1 দিয়ে কাজেই, এই তিনটি প্রক্রিয়াও খুবই সহজ। সেগুলো এরকম :

বুলিয়ান পূরক : 0 এর পূরক 1 এবং 1 -এর পূরক 0 লেখা হয় এভাবে : $\bar{0} = 1$ এবং $\bar{1} = 0$

বুলিয়ান গুণ : 0.0 = 0, 1.0 = 0, 0.1= 0, 1.1 = 1

বুলিয়ান যোগ : 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1 এবং 1 + 1 = 1

আমরা দেখতে পাচ্ছি উপরে দেখানো এলজেবরার নিয়মগুলোর ভেতর শুধু 1 + 1 = 1 এই যোগটি আমাদের প্রচলিত ধারণার সাথে মিলে না (কিন্তু যেহেতু আমরা শুধু {0, 1} সেট নিয়ে কাজ করছি এখানে অন্য কিছু বসানোরও সুযোগ নেই।) শুধু তাই নয় বুলিয়ান এলজেবরার প্রক্রিয়াগুলো লেখার সময় আমরা যদিও 0 এবং 1 এই দুটি সংখ্যা লিখছি কিন্তু মনে রাখতে হবে এই দুটি আসলে সংখ্যা নয়, এই দুটি হচ্ছে দুটি ভিন্ন অবস্থা। যেরকম 0 এবং 1 ইলেক্ট্রনিক সার্কিটে দুটি ভিন্ন ভিন্ন ভোল্টেজ (0 v এবং 5 v) হতে পারে, অপটিকেল ফাইবারে আলোহীন এবং আলোযুক্ত অবস্থা হতে পারে কিংবা লজিকের মিথ্যা (False বা F) এবং সত্য (True কিংবা T) হতে পারে।

বুলিয়ান এলজেবরা করার সময় সবার প্রথম পূরক তারপর গুণ এবং সবশেষে যোগ করতে হয়। তবে পাশাপাশি অসংখ্য প্রক্রিয়া থাকলে ব্র্যাকেট ব্যবহার করে বিভ্রান্তি কমিয়ে রাখা ভালো। কোনো বিভ্রান্তির সুযোগ না থাকলে x.y কে xy হিসেবে লেখা যায়।

উদাহরণ : $1.0 + \overline{(0+1)} = ?$
উত্তর : $1.0 + \overline{(0+1)} = 0 + \bar{1} = 0 + 0 = 0$

৩.৭.২ বুলিয়ান উপপাদ্য (Boolean Theorem)

আমাদের প্রচলিত এলজেবরার মতোই বুলিয়ান এলজেবরার বেশ কিছু উপপাদ্য রয়েছে। এর মাঝে গুরুত্বপূর্ণ কয়েকটি নিচে দেখানো হলো। বুলিয়ান এলজেবরা যেহেতু {0, 1} সেট দিয়ে তৈরি তাই চলকের (Variable) মান একবার 0 এবং আরেকবার 1 বসিয়ে এই উপপাদ্যগুলো খুবই সহজেই প্রমাণ করা যায়।

টেবিল 3.6: বুলিয়ান উপপাদ্য

দ্বৈত পরিপূরক (Double Complement)	$\bar{\bar{x}} = x$
অপরিবর্তনীয় উপপাদ্য (Idempotent)	$x + x = x \qquad x.x = x$
পরিচিতি উপপাদ্য (Identity)	$x + 0 = x \qquad x.1 = x$
কর্তৃত্ব উপপাদ্য (Domination)	$x + 1 = 1 \qquad x.0 = 0$
বিনিময় উপপাদ্য (Commutative)	$x + y = y + x \qquad xy = yx$
অনুষঙ্গ উপপাদ্য (Associative)	$x + (y + z) = (x + y) + z$ $x(yz) = (xy)z$
বিভাজন উপপাদ্য (Distributive)	$x + yz = (x + y)(x + z)$ $x(y + z) = xy + xz$
ডি মরগান উপপাদ্য (De Morgan)	$\overline{x.y} = \bar{x} + \bar{y}$ $\overline{x + y} = \bar{x}.\bar{y}$
সহায়ক উপপাদ্য (Absorption)	$x + xy = x$ $x(x + y) = x$

এখানে বেশ কিছু উপপাদ্য আমাদের পরিচিত এলজেবরার সাথে সাদৃশ্যপূর্ণ আবার বেশ কিছু উপপাদ্যের পরিচিত উপপাদ্যের সাথে মিল নেই।

$$x.\bar{x} = 0$$

উদাহরণ : বিভাজন উপপাদ্য $x + yz = (x + y).(x + z)$ **টি প্রমাণ কর।**

উত্তর : ডানদিক $(x + y).(x + z)$

$= xx + xz + yx + yz$

$= x + xz + yx + yz$ $\quad Idempotent\ x.x = x$

$= x(1 + z) + yx + yz$

$= x + yx + yz$ $\quad Domination\ 1 + z = 1$

$= x(1 + y) + yz$

$= x + yz$ $\quad Domination\ 1 + y = 1$

= বাম দিক (প্রমাণিত)

উদাহরণ : ডি মরগানের উপপাদ্য দুটি প্রতি ক্ষেত্রের জন্য মান বসিয়ে প্রমাণ কর।

উত্তর : এখানে যেহেতু x এবং y দুটি চলক রয়েছে, দুটিরই মান হওয়া সম্ভব 0 এবং 1 কাজেই সর্বমোট 2^2 বা চারটি ভিন্ন মান হওয়া সম্ভব। প্রত্যেকটির জন্য আলাদাভাবে লেখা যেতে পারে।

x	y	$x.y$	$\overline{x.y}$	$\bar{x}$	$\bar{y}$	$\bar{x} + \bar{y}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

$\overline{x.y} = \bar{x} + \bar{y}$ **(প্রমাণিত)**

x	y	$x + y$	$\overline{x + y}$	$\bar{x}$	$\bar{y}$	$\overline{x.y}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

$\overline{x + y} = \bar{x}.\bar{y}$ **(প্রমাণিত)**

নিজে কর : বুলিয়ান এলজেবরার ভেতর কোন কোন উপপাদ্যগুলো আমাদের পরিচিত এলজেবরার উপপাদ্য থেকে ভিন্ন। (Hint : চলক x, y, z -এর জন্য 0 এবং 1 -এর বাইরে কোনো মান বসানো হলে যেগুলো কাজ করে না সেগুলো পরিচিত এলজেবরার উপপাদ্য থেকে ভিন্ন।)

আমাদের পরিচিত সাধারণ এলজেবরায় আমরা যেরকম বেশ কিছু চলক ব্যবহার করে অন্য আরেকটি বড় এক্সপ্রেশন তৈরি করতে পারি, বুলিয়ান এলজেবরার বেলাতেও সেটা সত্যি। সাধারণ এলজেবরার মতো বুলিয়ান এলজেবরাতেও আমরা বুলিয়ান উপপাদ্যগুলো ব্যবহার করে সেগুলো অনেক সরল করে ফেলতে পারি। যেমন ধরা যাক x, y এবং z এই তিনটি চলক ব্যবহার করে নিচের এক্সপ্রেশনটি লেখা হয়েছে :

$$xyz + xy + x$$

এটাকে আমরা এভাবে সরল রূপ দিতে পারি :

$$xyz + xy + x = xy(z + 1) + x = xy + x = x(y + 1) = x$$

এটাকে সরল করার জন্য আমরা domination উপপাদ্য $z + 1 = 1$ এবং $y + 1 = 1$ ব্যবহার করেছি।

উদাহরণ : $xyz + x\bar{y}z + \bar{x}yz + \bar{x}\bar{y}z$ এক্সপ্রেশনটিকে সরল কর।

উত্তর : $xyz + x\bar{y}z + \bar{x}yz + \bar{x}\bar{y}z$

$= xz(y + \bar{y}) + \bar{x}z(y + \bar{y})$

$= xz + \bar{x}z$ যেহেতু $(y + \bar{y}) = 1$

$= z(x + \bar{x})$

$= z$ যেহেতু $(x + \bar{x}) = 1$

আমরা যখন ডিজিটাল ইলেকট্রনিক্সের শুরুতে নানা ধরনের গেট নিয়ে আলোচনা করব তখন দেখব বুলিয়ান এলজেবরার এভাবে একটি বড় এবং জটিল এক্সপ্রেশনকে সরল করতে পারলে একটি জটিল সার্কিটকে অনেক ছোট করে ফেলা যায়।

৩.৭.৩ ডি-মরগানের উপপাদ্য (De Morgan's Theorem)

1নং টেবিলে বেশ কিছু উপপাদ্য রয়েছে, এদের ভেতর থেকে ডি মরগান উপপাদ্যটিকে আলাদাভাবে বিবেচনা করা দরকার। বুলিয়ান এলজেবরার শুরুতে বলা হয়েছিল যে এখানে তিনটি প্রক্রিয়া করা হয়, পরিপূরক, গুণ এবং যোগ। আমরা ডি মরগান সূত্রটিতে দেখতে পাই দুটি চলকের যোগকে পরিপূরক করা হলে সেটি পূরক চলকের গুণ হিসেবে লেখা যায়। অর্থাৎ যোগকে গুণ দিয়ে প্রকাশ করা যায়।

$$\overline{x + y} = \bar{x}.\bar{y}$$

এই উপপাদ্যের একটি সুদূরপ্রসারী প্রভাব রয়েছে। যেহেতু পরিপূরক প্রক্রিয়া প্রয়োগ করে যেকোনো যোগকে গুণ হিসেবে প্রকাশ করা যায় তাই আমরা ইচ্ছে করলেই বলতে পারি, বুলিয়ান এলজেবরাতে মৌলিক প্রক্রিয়া তিনটি নয়- দুইটি। পরিপূরক এবং গুণ।

আবার আমরা যদি দ্বিতীয় ডি মরগান সূত্রটি ব্যবহার করি তাহলে পরিপূরক যেকোনো গুণকে আমরা যোগ দিয়ে পাল্টে দিতে পারব। অর্থাৎ

$$\overline{x.y} = \bar{x} + \bar{y}$$

কাজেই একইভাবে আমরা বলতে পারি বুলিয়ান এলজেবরাতে প্রক্রিয়া তিনটি নয়, প্রক্রিয়া দুটি অর্থাৎ পরিপূরক এবং যোগ। অর্থাৎ আমরা দেখতে পাচ্ছি বুলিয়ান এলজেবরাতে মৌলিক প্রক্রিয়া দুইটি, পরিপূরক ও গুণ কিংবা পরিপূরক ও যোগ।

উদাহরণ : Domination উপপাদ্য $x + 1 = 1$ কে গুণ দিয়ে প্রকাশ করা।

উত্তর : $x + 1 = 1$

দুইপাশে পরিপূরক করে আমরা লিখতে পারি, $\overline{x + 1} = \bar{1}$

ডি মরগান উপপাদ্য ব্যবহার করে : $\bar{x}.\bar{1} = \bar{1}$ কিংবা $\bar{x}.0 = 0$ (যেহেতু $\bar{1} = 0$)

$\bar{x}$ কে যদি আমরা অন্য একটি চলক y দিয়ে প্রতিস্থাপন করি :

$y.0 = 0$ যেটি Domination উপপাদ্যের দ্বিতীয় সূত্রটি।

উদাহরণ : Domination উপপাদ্য $x.0 = 0$ যোগ দিয়ে প্রকাশ কর।
উত্তর : দুই পাশে পরিপূরক নিয়ে : $\overline{x.0} = \bar{0}$
ডি মরগান উপপাদ্য ব্যবহার করে : $\bar{x} + \bar{0} = \bar{0}$
$\bar{x} + 1 = 1$ (যেহেতু $\bar{0} = 1$)
যদি $\bar{x}$ কে আমরা অন্য একটি চলক y দিয়ে প্রতিস্থাপন করি :
$y + 1 = 1$ যেটি Domination উপপাদ্যের প্রথম সূত্রটি।

দুইয়ের অধিক চলকের জন্য ডি মরগান উপপাদ্য

যদিও ডি মরগান উপপাদ্যটি x ও y দুটি চলকের জন্য দেখানো হয়েছিল কিন্তু এটি আসলে দুইয়ের অধিক যে কোনো সংখ্যক চলকের জন্য সত্যি। অর্থাৎ ডি মরগান সূত্রের ব্যাপক রূপ দুইটি হচ্ছে :

$$\overline{x_1 + x_2 + x_3 \ldots x_n} = \bar{x_1}.\bar{x_2}.\bar{x_3} \ldots \bar{x_n}$$

$$\overline{x_1.x_2.x_3 \ldots x_n} = \bar{x_1} + \bar{x_2} + \bar{x_3} \ldots \bar{x_n}$$

নিজে কর : $\overline{x_1 + x_2} = \bar{x_1}.\bar{x_2}$ হলে প্রমাণ কর $\overline{x_1 + x_2 + x_3 \ldots x_n} = \bar{x_1}.\bar{x_2}.\bar{x_3} \ldots \bar{x_n}$
সাহায্য : $\overline{x_1 + x_2 + x_3 \ldots x_n} = \overline{x_1 + (x_2 + x_3 \ldots x_n)} = \bar{x_1}.\overline{(x_2 + x_3 \ldots x_n)} = \cdots$

নিজে কর : $\overline{x_1.x_2} = \bar{x_1} + \bar{x_2}$ হলে প্রমাণ কর $\overline{x_1.x_2.x_3 \ldots x_n} = \bar{x_1} + \bar{x_2} + \bar{x_3} \ldots \bar{x_n}$

৩.৭.৪ সত্যক সারণী (Truth Table)

বুলিয়ান এলজেবরার পরিপূরক, যোগ এবং গুণ, এই তিনটি প্রক্রিয়াকে আমরা তিনটি সারণী বা টেবিল আকারেও লিখতে পারি। x এবং y যদি দুটি বুলিয়ান চলক হয় যেগুলো শুধু 0 এবং 1 এই দুটি মান পেতে পারে তাহলে কোন মানের জন্য কোন প্রক্রিয়ায় কোন ফলাফল পাওয়া যাবে সেটি আমরা এভাবে লিখতে পারি।

x	$\bar{x}$
0	1
1	0

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x.y$
0	0	0
0	1	0
1	0	0
1	1	1

একটি বিশেষ প্রক্রিয়ায় কোন ইনপুটের জন্য কোন আউটপুট পাওয়া যায় সেটি যদি একটি সারণী বা টেবিল দিয়ে পুরোপুরিভাবে প্রকাশ করা হয় সেটাকে সত্যক সারণী বা ট্রুথ টেবিল বলা হয়। উপরের সত্যক সারণী থেকে আমরা দেখতে পাচ্ছি যদি একটি চলক (x) থাকে তাহলে সত্যক সারণী দুটি ভিন্ন ভিন্ন ইনপুট থাকে। চলকের সংখ্যা যদি দুটি হয় তাহলে ইনপুটের সংখ্যা হয় $2^2 = 4$টি। চলকের সংখ্যা যদি হয় n তাহলে ইনপুটের সংখ্যা হয় 2^n টি।

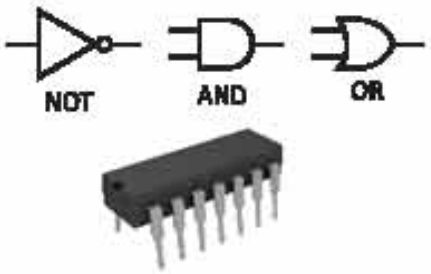
উদাহরণ : $x.\overline{(y + z)}$ বুলিয়ান ফাংশনটির সত্যক সারণী লিখ।

উত্তর : নিচে দেখানো হলো।

x	y	z	$(y+z)$	$\overline{(y+z)}$	$x.\overline{(y+z)}$
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

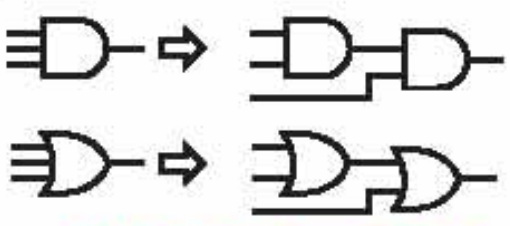
৩.৭.৫ মৌলিক গেট (AND, OR, NOT Gate)

এই অধ্যায়ের শুরুতে বলা হয়েছিল যে বুলিয়ান এলজেবরা হচ্ছে ডিজিটাল ইলেকট্রনিক্সের ভিত্তি- বিষয়টি কীভাবে ঘটে সেটি এখানে আলোচনা করা হবে। বুলিয়ান এলজেবরায় যে প্রক্রিয়াগুলোর কথা বলা হয়েছিল (পরিপূরক, গুণ এবং যোগ) সেগুলো বাস্তবায়ন করার জন্য ইলেকট্রনিক গেট তৈরি করা হয়। অর্থাৎ যে ইলেকট্রনিক ডিভাইস দিয়ে লজিক বাস্তবায়ন করা যায় সেগুলোকে গেট বলে। বুলিয়ান এলজেবরায় ইনপুট এবং আউটপুট দুটি সংখ্যা {0,1} দিয়ে প্রকাশ করা হয়েছিল। ডিজিটাল ইলেকট্রনিক্সে সেগুলো দুটি ভোল্টেজ দিয়ে বাস্তবায়ন করা হয়। ব্যবহারের প্রয়োজনের উপর নির্ভর করে নানা ধরনের কাজের জন্য নানা ধরনের ভোল্টেজ নির্ধারণ করে দেওয়া আছে।

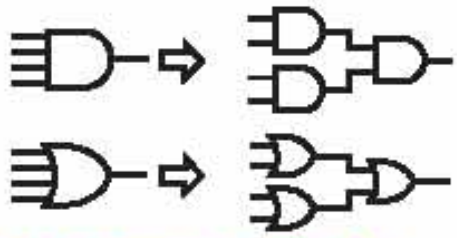


চিত্র 3.2 : NOT, AND এবং OR গেট এর প্রতীক এবং একটি ডিজিটাল ইন্টিগ্রেটেড সার্কিট (IC)

বুলিয়ান এলজেবার তিনটি প্রক্রিয়াকে বাস্তবায়ন করার জন্য যে তিনটি ইলেকট্রনিক গেট বা লজিক গেট ব্যবহার করা হয় তা 3.2 চিত্রে দেখানো হলো। এখানে পরিপূরক প্রক্রিয়াটির জন্য NOT গেট, গুণ করার জন্য AND এবং যোগ করার জন্য OR গেট। আমরা ছবিতে পরিপূরক, গুণ এবং যোগ করার জন্য যে সত্যক সারণী তৈরি করেছিলাম সেগুলোর দিকে তাকালেই এই নতুন নামকরণের যৌক্তিকতা বুঝতে পারব। NOT গেইটি একটি ইনপুটের বিপরীত অবস্থান তৈরি করে। AND গেটের আউটপুট 1 হওয়ার জন্য প্রথম এবং দ্বিতীয় দুটি ইনপুটকেই 1 হতে হয়। OR গেটের আউটপুট 1 হওয়ার জন্য প্রথম অথবা দ্বিতীয় যে কোনোটি অথবা দুটিই 1 হতে হয়। আমরা এই গেটগুলোকে মৌলিক গেট বলি কারণ এই তিনটি গেট ব্যবহার করে আমরা যে কোনো জটিল ডিজিটাল ইলেকট্রনিক্স গড়ে তুলতে পারব।



চিত্র 3.3 : তিন ইনপুটের AND এবং OR গেট

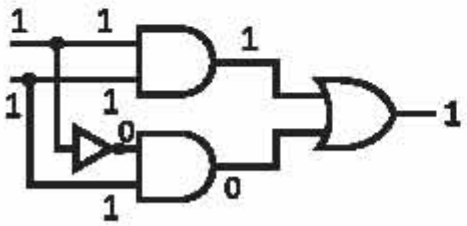


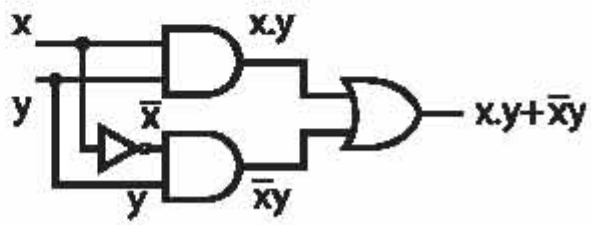
চিত্র 3.4 : চার ইনপুটের AND এবং OR গেট

আমরা যদিও দুই ইনপুটের AND এবং OR গেটের কথা বলেছি কিন্তু দুই থেকে বেশি ইনপুটের AND এবং OR গেট রয়েছে। শুধু তাই নয়, ইচ্ছে করলে আমরা দুই গেটের লজিক গেট ব্যবহার করেই দুই থেকে বেশি ইনপুটের লজিক গেট তৈরি করতে পারব।

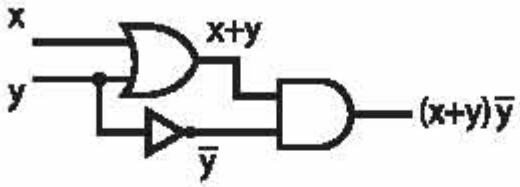
এবারে আমরা NOT, AND ও OR গেটগুলো ব্যবহার করে নানা ধরনের সার্কিট তৈরি করে এর ব্যবহারটি শিখে নেব।

উদাহরণ : নিচে দেখানো সার্কিটের ইনপুট দুটি যদি 1 হয় তাহলে আউটপুট কী হবে? একই সার্কিটে আমরা যদি নির্দিষ্ট মান না দিয়ে ইনপুট দুটিকে x এবং y বলি তাহলে আউটপুট কী?
উত্তর : নিচের ছবিতে দেখানো হলো।

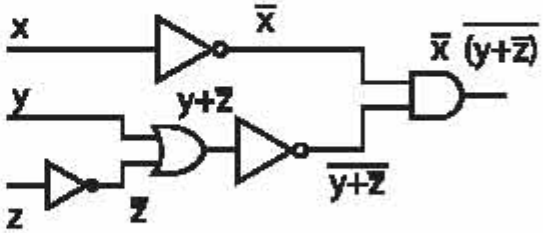




উদাহরণ : $(x + y)\bar{y}$ সার্কিটটি আঁকো।
উত্তর : পাশের ছবিতে দেখানো হলো।
x = 1, y = 0 হলে আউটপুট কী?
আউটপুট : $(x + y)\bar{y} = (1 + 0)\bar{0} = 1.1 = 1$



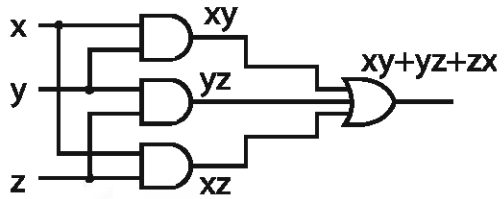
উদাহরণ : $\bar{x}\overline{(y + \bar{z})}$ সার্কিটটি আঁকো।
x = 1, y = 0 z = 1 হলে আউটপুট কী?
উত্তর : পাশের ছবিতে দেখানো হলো।
আউটপুট :
$\bar{x}\overline{(y + \bar{z})} = \bar{1}\overline{(0 + \bar{1})} = 0\overline{(0 + 0)} = 0$



উদাহরণ : $\bar{x}\overline{(y + \bar{z})}$ সার্কিটটির সত্যক সারণী তৈরি কর।
উত্তর : নিচের টেবিলে দেখানো হলো।

x	y	z	$\bar{x}$	$\bar{z}$	$(y+\bar{z})$	$\overline{(y+\bar{z})}$	$\bar{x}.\overline{(y+\bar{z})}$
0	0	0	1	1	1	0	0
0	0	1	1	0	0	1	1
0	1	0	1	1	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	1	0	0	0	1	0
1	1	0	0	1	1	0	0
1	1	1	0	0	1	0	0

উদাহরণ : তিনজনের ভিতর কমপক্ষে দুইজন "হ্যাঁ" ভোট দিলে ভোটে বিজয়ী বিবেচনা করা হবে এরকম একটি সার্কিট তৈরি কর।
উত্তর : পাশের ছবিতে দেখানো হলো।

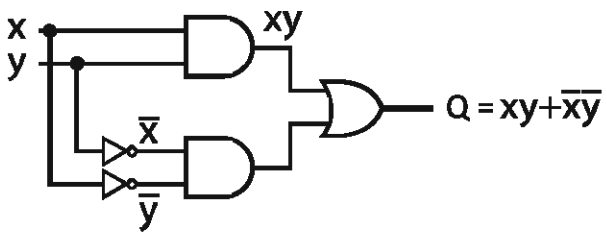


পরীক্ষা করে দেখ সত্যি সত্যি তিনটির ভেতর কমপক্ষে দুটো যদি 1 হয় তাহলে আউটপুট 1.

উদাহরণ : ধরা যাক তুমি একটি ঘরের আলো দুটি ভিন্ন ভিন্ন সুইচ দিয়ে নিয়ন্ত্রণ করতে চাও। অর্থাৎ আলো জ্বালানো থাকলে যে কোনো একটি সুইচ দিয়ে আলোটা নেভাতে পারবে আবার আলো নেভানো থাকলে যে কোনো একটি সুইচ দিয়ে সেটি দিয়ে জ্বালাতে পারবে।

উত্তর : মনে করি সুইচ দুটি হচ্ছে একটা সার্কিটের দুটি ইনপুট x এবং y, যখন x কিংবা y এর মান 1 তখন সুইচটি অন অবস্থায় আছে এবং যখন মান 0 তখন অফ অবস্থায় আছে। যেহেতু মাত্র দুইটি সুইচ কাজেই আমাদের মাত্র চারটি অবস্থানের জন্য আউটপুট Q বের করতে হবে। আলোটি আমরা Q আউটপুট দিয়ে প্রকাশ করতে পারি অর্থাৎ যখন Q এর মান 1 তখন আলোটি জ্বলবে যখন Q এর মান 0 তখন আলেটি নিভে যাবে। যখন দুটি সুইচই অফ, ধরা যাক তখন আলোটি জ্বলছে, অর্থাৎ x = 0, y = 0 এবং Q = 1 এটি হবে সত্যক সারণির প্রথম অবস্থান। এখান থেকে শুরু করে আমরা অন্য অবস্থাগুলো বের করতে পারব। এই অবস্থান থেকে যদি যে কোনো একটি সুইচ পরিবর্তন করতে চাই তাহলে সেটা হওয়া সম্ভব : x = 0, y = 1 কিংবা x = 1, y = 0 এবং তখন Q = 0 হতে হবে (অর্থাৎ আলোটি নিভে যেতে হবে।) আমরা সত্যক সারণির আরো দুইটি তথ্য পেয়ে গেছি। সত্যক সারণির শেষ অবস্থান x = 1, y = 1, এই অবস্থানে পৌঁছাতে হলে যেহেতু x = 0, y = 1 কিংবা x = 1, y = 0 অবস্থানের একটি সুইচের পরিবর্তন করতে হবে, কাজেই Q-এর মানও 0 থেকে 1 হতে হবে। 3.5 নং চিত্রে এই লাইট কন্ট্রোল সিস্টেমের ট্রুথ টেবিল এবং নিচের ছবিতে এটি বাস্তবায়ন করার জন্য প্রয়োজনীয় সার্কিটটি দেখানো হয়।

x	y	Q
0	0	1
0	1	0
1	0	0
1	1	1

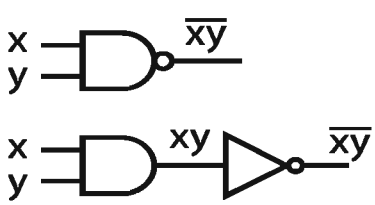


চিত্র 3.5 : লাইট কন্ট্রোল সিস্টেমের সত্যক সারণী এবং তার সার্কিট

৩.৭.৬ সর্বজনীন গেট (Universal Gate)

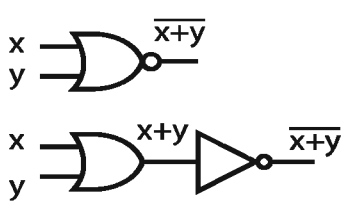
সর্বজনীন গেট আলোচনা করার আগে আমাদের NAND এবং NOR গেটের সাথে পরিচিত হতে হবে। এই গেট দুটির নাম থেকেই বোঝা যাচ্ছে যে NAND গেট হচ্ছে NOT-AND বা AND গেটের আউটপুটের NOT। অর্থাৎ একটি AND গেটের আউটপুটটি একটি NOT গেট দিয়ে রূপান্তরিত করে নিলে NAND গেটের আউটপুট পাওয়া যায়। 3.6 চিত্রে NAND গেটের সত্যক সারণী, প্রতীক এবং লজিকেল রূপটি দেখানো হলো।

x	y	$\overline{x.y}$
0	0	1
0	1	1
1	0	1
1	1	0



চিত্র 3.6 : NAND গেটের সত্যক সারণী, প্রতীক এবং লজিকেল রূপ

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0



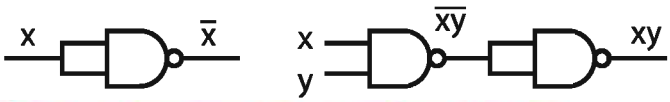
চিত্র 3.7 : NOR গেটের সত্যক সারণী, প্রতীক এবং লজিকেল রূপ

একইভাবে NOR গেট হচ্ছে OR গেটের আউটপুটকে NOT গেট দিয়ে পরিবর্তিত করা রূপ। তার সত্যক সারণী প্রতীক এবং লজিক গেটের রূপটি 3.7 চিত্রে দেখানো হলো।

বুলিয়ান এলজেবরার পরিপূরক, যোগ ও গুণ এই তিনটি প্রক্রিয়া রয়েছে। ডি মরগান সূত্র ব্যবহার করে দেখানো হয়েছিল যে পরিপূরক ও যোগ কিংবা পরিপূরক ও গুণ এরকম দুটি প্রক্রিয়া দিয়েই বুলিয়ান এলজেবরার যে কোনো প্রক্রিয়া করা সম্ভব। কাজেই আমরা বলতে পারি ডিজিটাল ইলেকট্রনিক্সের যেকোনো সার্কিট তিনটি ভিন্ন ভিন্ন লজিক গেটের পরিবর্তে দুটি গেট দিয়ে বাস্তবায়ন সম্ভব। সেই দুটি গেট হচ্ছে NOT এবং AND অথবা NOT এবং OR যেহেতু শুধু NAND গেট দিয়ে NOT এবং AND দুটি গেইট তৈরি করা সম্ভব আবার শুধু NOR গেট দিয়েই NOT এবং OR গেট তৈরি করা সম্ভব তাই আমরা NAND এবং NOR গেটকে সর্বজনীন (Universal) গেট বলে থাকি।

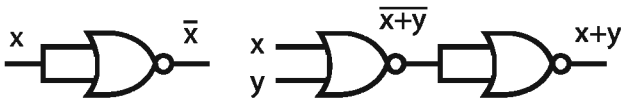
পাশের ছবিতে শুধু NAND গেট ব্যবহার করে NOT গেইট এবং AND গেট তৈরি করা এবং শুধু NOR গেট ব্যবহার করে NOT গেট এবং OR গেট তৈরি করার পদ্ধতি দেখানো হলো।

আমরা NAND গেট দিয়ে AND গেট এবং NOR গেট দিয়ে OR গেট তৈরি করা দেখিয়েছি।

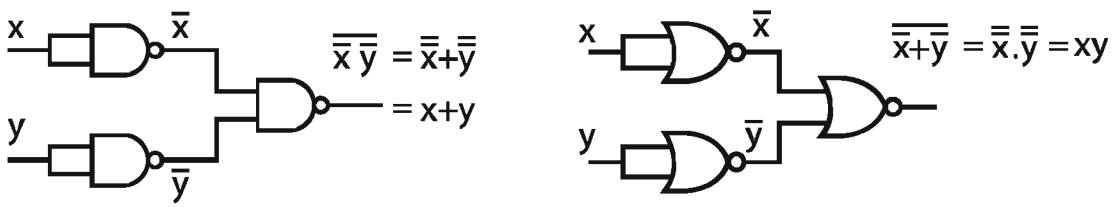


চিত্র 3.8 : লজিকেল NOT গেট এবং লজিকেল AND গেট

এখন আমরা উল্টোটা দেখাব, অর্থাৎ NAND গেট দিয়ে OR গেট এবং NOR গেট দিয়ে AND গেট তৈরি করা দেখাব।



চিত্র 3.9 : লজিকেল NOT গেট এবং লজিকেল OR গেট

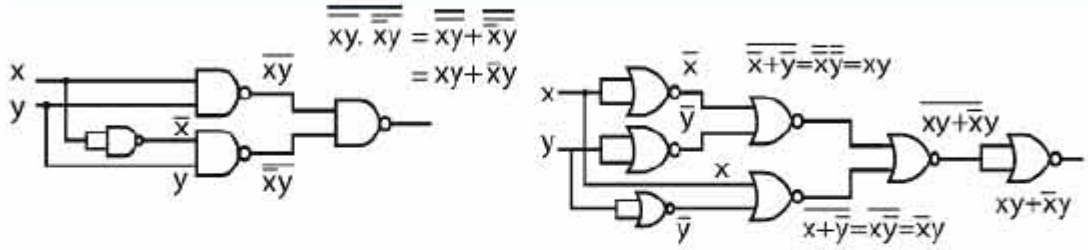


চিত্র 3.10 : NAND গেট দিয়ে OR গেট বাস্তবায়ন এবং NOR গেট দিয়ে AND গেট বাস্তবায়ন

এবারে আমরা শুধু NAND অথবা শুধু NOR গেট দিয়ে যে কোনো একটি সার্কিট তৈরি করে সর্বজনীন গেটের গুরুত্বটি দেখাব।

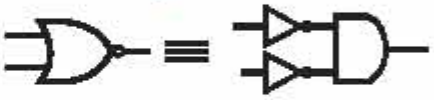
উদাহরণ : $x.y + \bar{x}y$ সার্কিটটি শুধু NAND গেট এবং শুধু NOR গেট দিয়ে তৈরি কর।

উত্তর : NAND ও NOR গেট দিয়ে তৈরি সার্কিট দুটির দিকে তাকিয়ে বুঝতে পারছ যে একই সার্কিট ভিন্ন ভিন্নভাবে তৈরি করা সম্ভব। কোনো সার্কিটে হয়তো বেশি গেটের প্রয়োজন হয় আবার কোনো সার্কিটে কম গেটের প্রয়োজন হয়। যত্ন করে সার্কিট তৈরি করার সময় সব সময় চেষ্টা করে অল্প গেট ব্যবহার করে বুদ্ধিসম্মত সার্কিট তৈরি করা।



চিত্র 3.11 : শুধু NAND গেট এবং শুধু NOR গেট দিয়ে তৈরি পূর্ণাঙ্গ সার্কিট

নিজে কর : পাশের ছবিটি কোন বুলিয়ান উপপাদ্য?



৩.৭.৭ বিশেষ গেট (XOR, XNOR Gate)

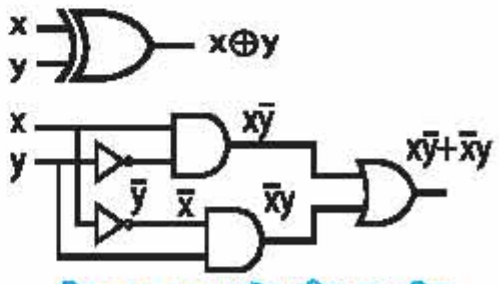
ডিজিটাল ইলেকট্রনিক্সের নানা ধরনের সার্কিটে অনেক সময়েই আমাদের বাইনারি সংখ্যা যোগ-বিয়োগ করতে হয়। এক বিটের বাইনারি যোগ এরকম :

0	0	1	1
+0	+1	+0	+1
0	1	1	10

1-এর সঙ্গে 1-এর যোগফলে দুটি বিট এসেছে, এখানে ডানপাশের বিটটিকে আমরা যোগফল এবং বামপাশের বিটটিকে বলতে পার ক্যারি। ক্যারি বিটটি নিয়ে আমরা আপাতত মাথা না ঘামিয়ে শুধু যোগফলের বিটটি নিয়ে আলোচনা করি। আমরা দেখেছি বুলিয়ানের যোগটিতে 1+1 করে আমরা 0 পাই না, 1 পাই। কাজেই বুলিয়ানের যোগ করার লজিক গেট AND কে আমরা বাইনারি যোগে ব্যবহার করতে পারি না। বাইনারির যোগে ব্যবহার করার জন্য Exclusive OR বা সংক্ষেপে XOR নামে আরেকটি লজিক গেট ব্যবহার করা হয়। এই গেটের সত্যক সারণী এবং প্রতীক 3.12 চিত্রে দেখানো হলো। সহজভাবে বলা যায় XOR গেটে ইনপুট দুটি ভিন্ন হলে আউটপুট 1, তা না হলে আউটপুট 0। XOR গেটের লজিক $x\bar{y} + \bar{x}y$, তোমরা এটা পরীক্ষা করে নিশ্চিত হয়ে নাও।

XOR গেটের সত্যক সারণী

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0



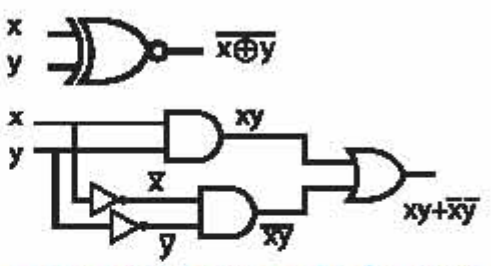
চিত্র 3.12 : XOR গেটের প্রতীক এবং লজিক

ডিজিটাল ইলেকট্রনিক্সে ব্যবহার করার জন্য XOR গেট আলাদাভাবে পাওয়া যায়। তবে আমরা ইচ্ছে করলে মৌলিক গেটগুলো ব্যবহার করেও XOR -এর লজিক বাস্তবায়ন করতে পারি।

প্রয়োজনীয় কোনো গেট তৈরি করা হলে সাধারণত তার NOT গেটটিও তৈরি করা হয়। সেই হিসেবে XNOR গেটটি বহুল ব্যবহৃত। XOR গেটের আউটপুটটির পর একটি NOT গেট বসিয়ে XNOR তৈরি করা সম্ভব হলেও গেটের সংখ্যা কমানোর জন্য পাশের ছবিতে দেখানো উপায়ে এই লজিকটি পাওয়া সম্ভব।

x	y	$\overline{x \oplus y}$
0	0	1
0	1	0
1	0	0
1	1	1

যেহেতু NAND এবং NOR গেট সর্বজনীন গেট কাজেই মৌলিক গেট ব্যবহার না করে শুধু NAND অথবা শুধু NOR গেট ব্যবহার করে XOR অথবা XNOR-এর লজিক বাস্তবায়ন করা সম্ভব। সর্বজনীন গেট ব্যবহার করে AND অথবা OR গেট বাস্তবায়নের সময় পদ্ধতিটি না দেখিয়ে সরাসরি উত্তরটি দেখানো হয়েছিল। এবারে আমরা NAND এবং NOR গেট ব্যবহারের পদ্ধতিটি দেখিয়ে তার জন্য প্রয়োজনীয় সার্কিট তৈরি করব।



চিত্র 3.13 : XNOR গেটের সত্যক সারণী, প্রতীক এবং লজিক

উদাহরণ : শুধু NAND এবং NOR গেট ব্যবহার করে XOR তৈরি কর।

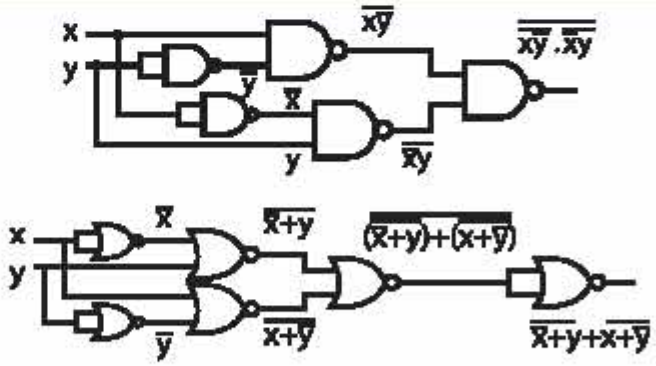
উত্তর : আমরা জানি XOR গেটের লজিক $x\bar{y} + \bar{x}y$ শুধু NAND গেট দিয়ে এই লজিক তৈরি করতে হলে ডি মরগান সূত্র ব্যবহার করে বুলিয়ান যোগ (+) কে বুলিয়ান গুণে (.) পাল্টে নিতে হবে। যেহেতু দুইবার পরিপূরক করা হলে লজিকের পরিবর্তন হয় না তাই আমরা লিখতে পারি :

$$x\bar{y} + \bar{x}y = \overline{(\overline{x\bar{y} + \bar{x}y})}$$

দ্বৈত পরিপূরক ডি মরগান সূত্র ব্যবহার করে যোগকে গুণ দিয়ে প্রতিস্থাপন করা হলে সেটি হবে:

$= \overline{\overline{x\bar{y}}.\overline{\bar{x}y}}$ ডি মরগান সূত্র

এবারে আমরা সার্কিটটি এঁকে ফেলি। (চিত্র 3.14)



চিত্র 3.14 : শুধু NAND এবং NOR গেট ব্যবহার করে তৈরি XOR কর

একইভাবে শুধু NOR ব্যবহার করে XOR তৈরি করতে হলে $x\bar{y}$ এবং $\bar{x}y$ -এর ভেতরকার বুলিয়ান গুণকে ডি মরগান সূত্র ব্যবহার করে যোগে রূপান্তর করতে হবে।

$x\bar{y} + \bar{x}y = \overline{(\overline{x\bar{y}})} + \overline{(\overline{\bar{x}y})}$ দ্বৈত পরিপূরক

$= \overline{\bar{x} + \bar{\bar{y}}} + \overline{\bar{\bar{x}} + \bar{y}}$ ডি মরগান সূত্র

$= \overline{\bar{x} + y} + \overline{x + \bar{y}}$

এবারে সার্কিটটি এঁকে ফেলা যাবে। (চিত্র 3.14)

উদাহরণ : শুধু NAND এবং NOR ব্যবহার করে XNOR তৈরি কর।

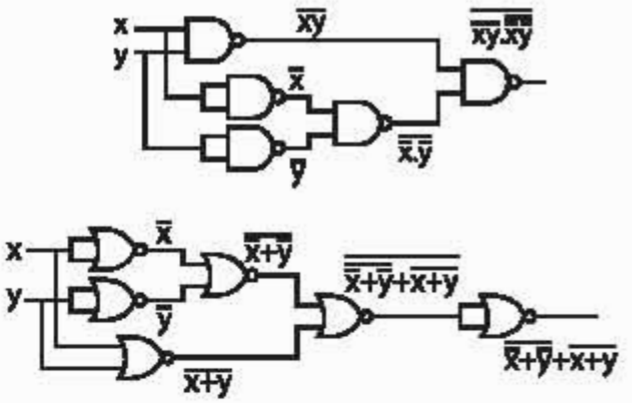
উত্তর : আমরা আগের উদাহরণের প্রক্রিয়ায় শুধু NAND ব্যবহার করে XNOR তৈরি করতে পারি। XNOR এর লজিক হচ্ছে : $xy + \bar{x}\bar{y}$ লজিক অপরিবর্তিত রেখে দ্বৈত পরিপূরক করা হলে আমরা পাই :

$xy + \bar{x}\bar{y} = \overline{\overline{xy + \bar{x}\bar{y}}}$ দ্বৈত পরিপূরক

এবারে ডি মরগান সূত্র ব্যবহার করে যোগকে গুণে রূপান্তর করতে হবে।

$= \overline{\overline{xy}.\overline{\bar{x}\bar{y}}}$ ডি মরগান সূত্র

এখন সার্কিটটা এঁকে ফেলা যাবে। (চিত্র 3.15)



চিত্র 3.15 : শুধু NAND এবং NOR গেট ব্যবহার করে তৈরি XNOR গেট

একইভাবে শুধু NOR ব্যবহার করে XNOR তৈরি করতে হলে xy এবং $\bar{x}\bar{y}$ -এর ভেতরকার বুলিয়ান গুণকে ডি মরগান সূত্র ব্যবহার করে যোগে রূপান্তর করতে হবে। XNOR এর লজিক $xy + \bar{x}\bar{y}$ অপরিবর্তিত রেখে দ্বৈত পরিপূরক করা হলে আমরা পাই :

$xy + \bar{x}\bar{y} = \overline{\overline{xy}} + \overline{\overline{\bar{x}\bar{y}}}$ দ্বৈত পরিপূরক এবারে ডি মরগান সূত্র ব্যবহার করে যোগকে গুণে রূপান্তর করতে হবে।

$= \overline{\bar{x} + \bar{y}} + \overline{\overline{\bar{x}} + \overline{\bar{y}}}$ ডি মরগান সূত্র

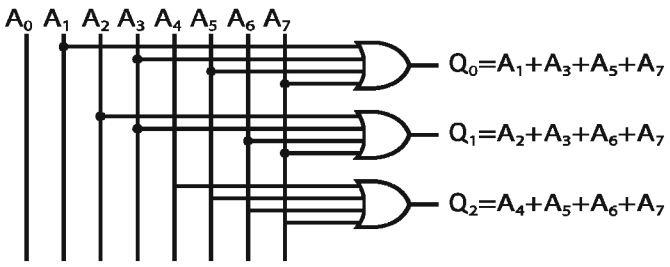
দ্বৈত পরিপূরক করে আরো সহজে লেখা যায় :

$= \overline{\bar{x} + \bar{y}} + \overline{x + y}$ দ্বৈত পরিপূরক। এবারে সার্কিটটি এঁকে ফেলা যাবে (চিত্র 3.15)।

৩.৭.৮ এনকোডার (Encoder)

বুলিয়ান এলজেব্রা ব্যবহার করে ডিজিটাল ইলেকট্রনিক্স আলোচনা করতে গিয়ে এখন পর্যন্ত নানা ধরনের গেট আলোচনা করা হয়েছে। এখন আমরা একাধিক গেট ব্যবহার করে তৈরি করা নানা ধরনের প্রয়োজনীয় ডিজিটাল সার্কিট সম্পর্কে আলোচনা করব। উল্লেখ্য যে আমাদের দৈনন্দিন প্রয়োজনে আলাদাভাবে গেট ব্যবহার করে এই সার্কিট তৈরি করতে হয় না, কারণ প্রায় সবগুলোই কোনো না কোনোভাবে ইন্টিগ্রেটেড সার্কিট হিসেবে পাওয়া যায়।

ইনপুট								আউটপুট		
A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	Q_2	Q_1	Q_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1



চিত্র 3.16 : এনকোডারের সত্যক সারণী এবং এই সত্যক সারণি বাস্তবায়নের জন্য প্রয়োজনীয় সার্কিট

এনকোডার ও ডিকোডার এরকম দুটো ডিজিটাল সার্কিট। এনকোডারে ইনপুট হিসেবে থাকে বেশ কয়েকটি ইনপুট লাইন এবং এই ইনপুট লাইনগুলোর যে কোনো একটিকে সিগন্যাল দিয়ে উজ্জীবিত করা হয় (অর্থাৎ শুধু সেই লাইনটির মান 1 অন্য সবগুলোর 0)। কত নম্বর লাইনটিকে উজ্জীবিত করা হয়েছে সেই সংখ্যাটি এনকোডারে বাইনারি সংখ্যা হিসেবে আউটপুটে দেখানো হয়। ধরা যাক ইনপুটে আটটি লাইন আছে, (A_0 থেকে A_7) এই আটটি লাইনের যে কোনো একটিতে ইনপুট দেওয়া হবে। কত নম্বর লাইনে (0 থেকে 7) ইনপুট দেওয়া হয়েছে সেটি জানানোর জন্য আউটপুটে তিনটি লাইনের প্রয়োজন (Q_0, Q_1 এবং Q_2)। আমরা প্রথমেই এই আটটি ইনপুট এবং তিনটি আউটপুটের এনকোডারের সত্যক সারণী বা ট্রুথ টেবিলটি তৈরি করে নেই (চিত্র 3.16)। যেমন : A_2 ইনপুট লাইনে সিগনাল দেওয়া হলে আমরা আউটপুটে বাইনারি 010 বা 2 সংখ্যাটি পাই কিংবা A_5 ইনপুট লাইনে সিগনাল দেওয়া হলে আমরা আউটপুটে বাইনারি 101 বা 5 সংখ্যাটি পাই ।

সত্যক সারণীটি যদি ঠিকভাবে লেখা হয়ে থাকে তাহলে তার জন্য সার্কিট তৈরি করা মোটেও কঠিন নয়। সত্যক সারণীটির দিকে তাকালেই আমরা দেখতে পাব, Q_0 লাইনে 1 পেতে হবে যখন A_1, A_3, A_5 এবং A_7 লাইনগুলোতে ইনপুট 1 দেয়া হয়েছে। কাজেই আমরা বলতে পারব, A_1, A_3, A_5 এবং A_7 লাইন চারটি একটি OR গেটের ইনপুটের সাথে সংযুক্ত করতে হবে এবং তার আউটপুট হবে Q_0। একইভাবে বলতে পারি Q_1 লাইনটি 1 দেবে যখন A_2, A_3, A_6 এবং A_7 লাইনগুলোতে ইনপুট 1 দেয়া হয়েছে। কাজেই একটা OR গেটের ইনপুট হিসেবে A_2, A_3, A_6 এবং A_7 হিসেবে সংযুক্ত করা হলে তার আউটপুট হবে Q_1। একইভাবে A_4, A_5, A_6 এবং A_7 একটি OR গেটের ইনপুটের সাথে সংযুক্ত করলে তার আউটপুট হবে Q_2। (চিত্র 3.16)

কাজেই এবারে আমরা খুব সহজেই 8 (আট) ইনপুট ও 3 আউটপুটের এনকোডারের সার্কিটটি তৈরি করতে পারি। তোমরা ইচ্ছা করলেই পরীক্ষা করে দেখতে পার। A_2 লাইনে ইনপুট 1 দেয়া হলে আউটপুটে বাইনারি 2 সংখ্যা পাবে কিংবা A_7 লাইনে ইনপুট 1 দেয়া হলে বাইনারি 7 সংখ্যা পাবে।

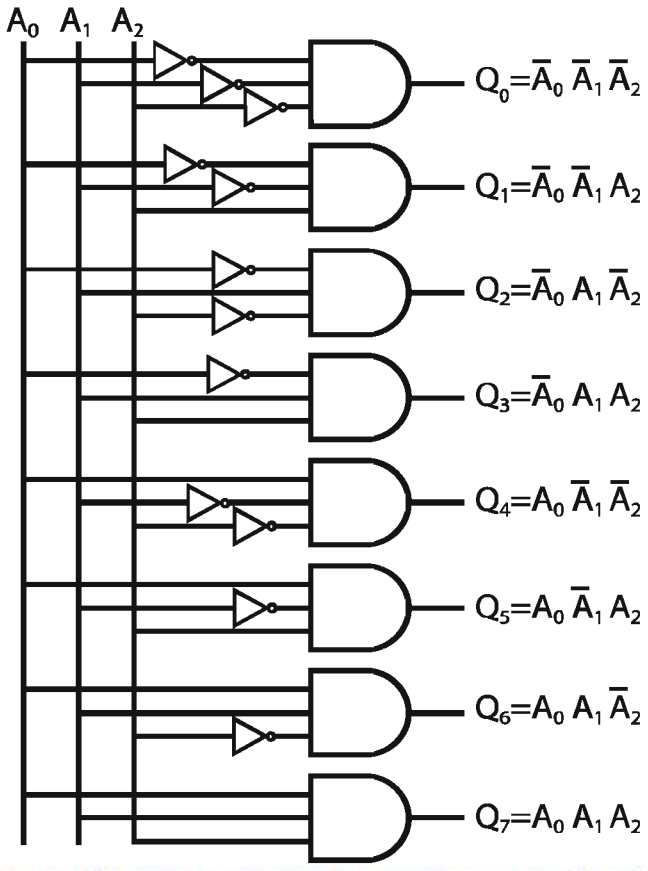
সমস্যা : আমরা সার্কিটে দেখতে পাচ্ছি A_0 ইনপুট লাইনটি ব্যবহার না করেই সার্কিটটি তৈরি করেছি। এটি কীভাবে সম্ভব?

৩.৭.৯ ডিকোডার (Decoder)

ডিকোডার সার্কিট এনকোডারের ঠিক বিপরীত কাজটুকু করে। এনকোডারে আলাদা আলাদা লাইনের সিগন্যালকে এনকোড করে আউটপুটে বাইনারি সংখ্যা হিসেবে প্রদান করেছে। ডিকোডার ইনপুটে বাইনারি কোনো সংখ্যা দেয়া হলে আউটপুটে সেই সংখ্যার লাইনটিতে একটি সিগন্যাল 1 দেওয়া হয়, অন্যগুলো 0 থেকে যায়। বিষয়টি বোঝার জন্য আমরা প্রথমেই ডিকোডারের সত্যক সারণী বা ট্রুথ টেবিলটি প্রস্তুত করি। বোঝাই যাচ্ছে, এনকোডারে যেগুলো ছিল আউটপুট লাইন, ডিকোডারে সেটা হবে ইনপুট লাইন এবং এনকোডারে যেগুলো ছিল ইনপুট লাইন ডিকোডারে সেগুলো হবে আউটপুট লাইন। ট্রুথ টেবিলে প্রথমে তিনটি ইনপুট লাইন (A_0, A_1, A_2) এবং তার পরে আটটি আউটপুট লাইন (Q_0, Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7) দেখানো হয়েছে (চিত্র 3.17)।

ডিকোডারের সত্যক সারণি

ইনপুট			আউটপুট							
A_2	A_1	A_0	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
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



চিত্র 3.17 : ডিকোডারের সত্যক সারণি বাস্তবায়নের জন্য প্রয়োজনীয় সার্কিট

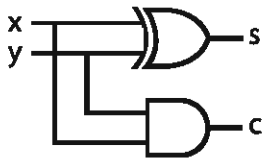
এটাকে কার্যকর করানোর জন্য ঠিক কী ধরনের সার্কিট ব্যবহার করতে হবে সেটা আমরা সত্যক সারণীটির দিকে তাকালেই বুঝতে পারব। যেহেতু এখানে Q_0 থেকে Q_7 এই আটটি আউটপুট রয়েছে কাজেই এই আটটি আউটপুটের জন্য আটটি তিন ইনপুটের AND গেট ব্যবহার করতে হবে। 3.17 চিত্রে সার্কিটটা দেখানো হয়েছে, প্রত্যেকটি তিন ইনপুটের AND গেটে A_0, A_1, A_2 এর সিগন্যাল দেয়া হয়েছে, কোনো কোনো সময় সরাসরি, কোনো কোনো সময় NOT গেট ব্যবহার করে পরিবর্তন করে। যেমন- প্রথম আউটপুটের জন্য A_0, A_1, A_2 সিগন্যাল (0,0,0) সরাসরি AND গেটে দেয়া হলে সেটি Q_0 আউটপুটে 1 দেবে না। আমরা জানি AND গেটের আউটপুটে 1 পেতে হলে ইনপুটের সব 1 হতে হয়। কাজেই A_0, A_1, A_2 এর সিগন্যাল (0,0,0) এর তিনটিকেই NOT করে দেয়া হলেই সেটি আউটপুটে 1 দেবে, সার্কিটে সেটা করা হয়েছে। ঠিক সেভাবে Q_3 তে পজেটিভ সিগন্যাল পেতে হলে তার জন্য নির্দিষ্ট AND গেটের ইনপুটে প্রথমটি (A_0) NOT করে অন্য দুটি সরাসরি দিতে হবে। একইভাবে বলা যায় Q_7 এর জন্য নির্ধারিত AND গেটটিতে A_0, A_1, A_2 এর সিগন্যাল যেহেতু সবই 1, তাই কোনোটিই NOT করার প্রয়োজন নেই, সরাসরি দেওয়া হলেই আমরা 1 আউটপুট পাব। সার্কিটে সেটা করা হয়েছে, তোমরা সত্যক সারণীর সাথে মিলিয়ে সার্কিটটি দেখে নাও।

৩.৭.১০ অ্যাডার (Adder)

আমরা এবারে লজিক গেট দিয়ে তৈরি করা আরো একটি ডিজিটাল সার্কিটের কথা বলব যেটি বাইনারি সংখ্যা যোগ করতে পারে। আমরা ইতোমধ্যে জেনে গেছি যে সঠিকভাবে বাইনারি সংখ্যা যোগ করতে পারলেই প্রয়োজনে সেই একই সার্কিট ব্যবহার করে বিয়োগ, গুণ এবং ভাগ করতে পারব।

XOR লজিক গেটটি আলোচনা করার সময় আমরা বাইনারি যোগ 1 + 1 = 10 সংখ্যাটিতে বলেছিলাম এর মাঝে ডানপাশের বিটটি যোগফল এবং বাম পাশের (হাতে থাকা) বিটটি ক্যারি (carry)। যোগফলের বিটটি XOR গেট দিয়ে পাওয়া যায় কিন্তু ক্যারি বিটটি কীভাবে পাওয়া যায় সেটি তখন আলোচনা করা হয়নি। সেটি খুবই সহজ একটি AND গেট দিয়ে পাওয়া যেতে পারে। কাজেই আমরা একটি বিটের সাথে অন্য একটি বিটের বাইনারি যোগ নিচের সার্কিট দিয়ে পেতে পারি (চিত্র 3.18) :

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



চিত্র 3.18 : x এবং y এই দুইটি বিট যোগ করার জন্য হাফ অ্যাডারের সত্যক সারণি এবং এই সত্যক সারণি বাস্তবায়নের জন্য প্রয়োজনীয় সার্কিট

এই ধরনের সার্কিটের নাম হচ্ছে হাফ অ্যাডার, কারণ এটি পূর্ণাঙ্গ বাইনারি যোগের সার্কিট নয়, এটি আংশিকভাবে যোগ করতে পারে। আগের ধাপ থেকে ক্যারি বিট হিসেবে 1 চলে এলে তখন যোগ করতে পারে না। প্রকৃত বাইনারি যোগে দুটি বিট যোগ করতে হলেও মাঝে মাঝেই এর আগের দুটি বিটের যোগ থেকে ক্যারি বিট চলে আসে, তখন দুইটি নয়, তিনটি বিট যোগ করার প্রয়োজন হতে পারে। নিচে দুটি বাইনারি সংখ্যার যোগফল দেখানো হয়েছে।

```
  ↓↓ ↓
 1001101
 1011001
---------
10100110
```

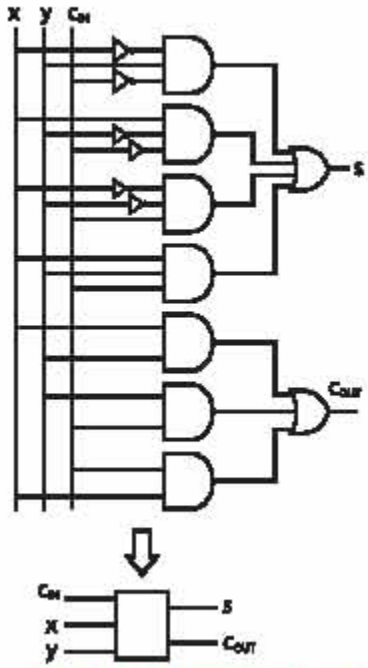
নিজে কর : মৌলিক গেট দিয়ে হাফ অ্যাডার তৈরি কর।

যদিও দুটি করে বিট যোগ করা হয়েছে কিন্তু তীর চিহ্ন দিয়ে দেখানো বিট দুটির বেলায় আগের ধাপ থেকে 1 বিটটি এসেছে বলে আসলে তিনটি বিট যোগ করা হয়েছে। আমরা অন্যভাবেও বলতে পারি, প্রতিবারই আমরা তিনটি বিট যোগ করেছি, কিন্তু অন্য ধাপগুলোতে ক্যারি বিটের মান ছিল 0। কাজেই এবারে আমরা x, y এবং C_{IN}, এই তিনটি ইনপুটের জন্য ট্রুথ টেবিলটি লিখে ফেলতে পারি। (টেবিল 3.6) এখানে x, y হচ্ছে বাইনারি যোগের প্রদত্ত সংখ্যার বিট এবং C_{IN} হচ্ছে আগের ধাপ থেকে আসরা ক্যারি বিটের মান। ট্রুথ টেবিলে আউটপুট দুটি, S এবং C_{OUT}। S হচ্ছে দুটি বিটের যোগফল, C_{OUT} হচ্ছে ক্যারি বিট যেটি পরের ধাপে C_{IN} হিসেবে যুক্ত হয়।

টেবিল 3.6

ইনপুট			আউটপুট	
x	y	C_{IN}	S	C_{OUT}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

ট্রুথ টেবিলের দিকে তাকিয়ে আমরা দেখতে পাচ্ছি x, y এবং C_{IN} -এর সম্ভাব্য আটটি ভিন্ন ভিন্ন ইনপুটের ভেতর চারটি ক্ষেত্রে যোগফল (S) এবং চারটি ক্ষেত্রে ক্যারি (C_{OUT}) আউটপুটের মান 1 হতে হবে। ডিকোডারের বেলায় আমরা যেভাবে AND গেটের আউটপুট 1 পাওয়ার জন্য NOT গেট দিয়ে ইনপুট পরিবর্তন করেছিলাম, এখানেও আমরা হুবহু একই পদ্ধতি গ্রহণ করতে পারি। 3.19 চিত্রে সেভাবে সার্কিটটি এঁকে দেখানো হলো। আমাদের আউটপুট দুটি (S এবং C_{OUT}) পাওয়ার জন্য AND গেটগুলোর OR গেট দিয়ে একত্র করে নেয়া হয়েছে। তবে তোমরা একটু অবাক হয়ে ভাবতে পার, S এবং C_{OUT} দুটির লজিক একই ধরনের থাকার পরও C_{OUT} -এর জন্য সার্কিটটি



চিত্র 3.19 : টেবিলে দেখানো সত্যক সারণি বাস্তবায়নের জন্য ফুল অ্যাডারের সার্কিট ও ব্লক ডায়াগ্রাম

তুলনামূলকভাবে সহজ কেন? মাত্র তিনটি দুই ইনপুট AND গেট দিয়ে কীভাবে আমরা সঠিক আউটপুট পেয়ে গেলাম?

S এর বেলায় 1 আউটপুটের জন্য INPUT এর মান হতে হবে এরকম :

$$S = \bar{x}y\overline{C_{IN}} + x\bar{y}\overline{C_{IN}} + \bar{x}\bar{y}C_{IN} + xyC_{IN}$$

একইভাবে ক্যারি আউটের জন্য C_{OUT} এর মান হতে হবে এরকম :

$$C_{OUT} = xy\overline{C_{IN}} + \bar{x}yC_{IN} + x\bar{y}C_{IN} + xyC_{IN}$$

কিন্তু এটাকে সহজ করে এভাবে লেখা সম্ভব। কীভাবে সম্ভব তার উত্তরটি নিচের উদাহরণে ব্যাখ্যা করা হয়েছে।

$$C_{OUT} = xy + yC_{IN} + xC_{IN}$$

উদাহরণ : S এর জন্য আউটপুটটি যৌক্তিক গেট দিয়ে আরও সরল করা সম্ভব না। তবে C_{OUT} -এর সমীকরণটি আরও সরল করা সম্ভব। তোমরা কি আরও সরল করে দেখাতে পারবে?

উত্তর : যেহেতু $A + A = A$, তাই আমরা সর্বশেষ টার্ম xyC_{IN}টি অন্য তিনটি টার্মের প্রত্যেকটার সাথে যোগ করতে পারি :

$$C_{OUT} = (xy\overline{C_{IN}} + xyC_{IN}) + (\bar{x}yC_{IN} + xyC_{IN}) + (x\bar{y}C_{IN} + xyC_{IN})$$

এখন আমরা এভাবে সাজাতে পারি

$$C_{OUT} = xy(\overline{C_{IN}} + C_{IN}) + yC_{IN}(\bar{x} + x) + xC_{IN}(\bar{y} + y)$$

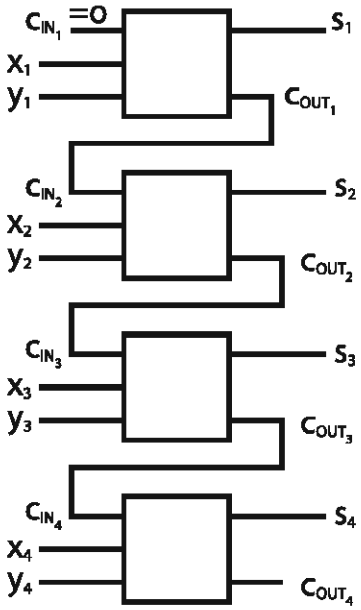
যেহেতু $A + \bar{A} = 1$, আমরা লিখতে পারি : $C_{OUT} = xy + yC_{IN} + xC_{IN}$

দেখতে পাচ্ছ পুরো সার্কিটটি অনেক সরল হয়ে গেছে, কিন্তু এটি সঠিক আউটপুট দেবে, ইচ্ছা করলে সেটি পরীক্ষা করে দেখতে পার।

নিজে কর : দুইটি অর্ধযোগ (হাফ অ্যাডার) বর্তনী দিয়ে একটি পূর্ণযোগ (ফুল অ্যাডার) বর্তনী বানানো সম্ভব কী? উত্তরের সাপেক্ষে যুক্তি দেখাও।

দুটি বিট যোগ করার এই সার্কিটটিকে ফুল এডার বলে। যেকোনো সত্যকার কাজের সার্কিটে অনেক বিট যোগ করতে হয়, কিন্তু প্রত্যেকটি বিটের জন্য যেন এই পুরো সর্কিটটি আঁকতে না হয় সেজন্য আমরা পুরো সার্কিটটিকে একটা ব্লক ডায়াগ্রাম দিয়ে দেখিয়েছি, এখানে শুধু ইনপুট এবং আউটপুট লাইনগুলো দেখানো হয়েছে। চার বিটের একটি বাইনারি যোগের জন্য কীভাবে চারটি ফুল এডার সার্কিট যোগ করতে হবে সেটি ব্লক ডায়াগ্রাম গুলো যুক্ত করে দেখানো হলো। (চিত্র 3.20)

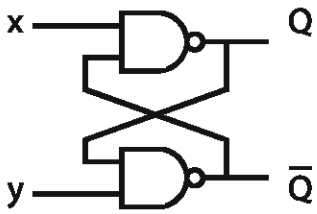
লক্ষ কর, প্রথম ব্লক ডায়াগ্রামে $C_{IN1} = 0$ কারণ প্রথম দুটি বিট যোগ করার সময় আগের কোনো ধাপ থেকে কিছু C_{IN} আসা সম্ভব নয়। উল্লেখ্য যে, চার বিট যোগ করতে হলে যোগফল সঠিকভাবে দেখাতে হলে কিন্তু সর্বশেষ C_{OUT} -এর জন্য পঞ্চম বিট প্রয়োজন হয়।



চিত্র 3.20: চার বিট যোগ করার প্রয়োজনীয় সার্কিটের জন্য ব্লক ডায়াগ্রাম

৩.৭.১১ রেজিস্টার (Register)

আমরা এতক্ষণ পর্যন্ত যে কয়টি সার্কিট তৈরি করতে শিখেছি তার প্রত্যেকটিরই একটি বিশেষত্ব রয়েছে, সেটি হচ্ছে যতক্ষণ ইনপুটে সঠিক সিগন্যাল দেওয়া হবে ততক্ষণ আউটপুটে সঠিক সিগন্যাল পাব। ইনপুটে সঠিক সিগন্যাল না থাকলে আউটপুটে কোনো বিশ্বাসযোগ্য মান থাকবে না।



x	y	Q	$\bar{Q}$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	0	1
		1	0

চিত্র 3.21 : একটি ফ্লিপফ্লপের সার্কিট এবং তার বিচিত্র সত্যক সারণি

কিন্তু আমাদের অনেক সময়েই একটি সার্কিটে কোনো একটি মান সংরক্ষণ করতে হয়, আমরা সেটাকে মেমোরি বলে থাকি। এখন আমরা এ ধরনের একটি সার্কিটের কথা বলব যেখানে একটি ইনপুট দিয়ে সেই ইনপুটের মানটিকে সংরক্ষণ করা সম্ভব। এই ধরনের সার্কিটকে বলে ফ্লিপফ্লপ। 3.21 চিত্রে একটি ফ্লিপফ্লপের সার্কিট দেখানো হলো। এখানে Q একটি আউটপুট এবং $\bar{Q}$ তার পূরক।

এবারে আমরা এই ফ্লিপ ফ্লপের সত্যক সারণী বা ট্রুথ টেবিলটি লেখার চেষ্টা করি। NAND গেটের জন্য যেকোনো একটি ইনপুট 0 হলে আউটপুট 1 হয়। তাই ইনপুট x এবং y দুটোই যদি 0 হয় (অন্য ইনপুটের মান যাই হোক না কেন) দুটো NAND গেটের আউটপুট Q এবং $\bar{Q}$ দুটোর মানই হবে 1। কিন্তু আমরা যেহেতু একটিকে Q অন্যটিকে $\bar{Q}$ হিসেবে অভিহিত করছি, অর্থাৎ একটি 1 হলে অন্যটিকে অবশ্যই 0 হতে হবে, কাজেই দুটোই 1 হওয়া সঠিক নয়। তাই আমরা ধরে নেব ইনপুট x এবং y দুটোই কখনো একসাথে 0 করা

হবে না, অর্থাৎ এটি গ্রহণযোগ্য ইনপুট নয়। তবে x = 0 এবং y = 1 হলে Q এবং $\bar{Q}$ আউটপুট দুটি যুক্তিসঙ্গতভাবে যথাক্রমে 1 এবং 0 হবে। আবার ইনপুট x = 1 এবং y = 0 হলে এর বিপরীত ব্যাপারটি ঘটে, অর্থাৎ তখন Q = 0 এবং $\bar{Q} = 1$ পাওয়া যায়। তোমরা অবশ্যই এটি পরীক্ষা করে নিশ্চিত হয়ে নাও।

তবে দুটোই 1 হলে সবচেয়ে চমকপ্রদ বিষয়টি ঘটে। তোমরা নিজেরাই পরীক্ষা করে দেখতে পারো যে তাহলে Q এবং $\bar{Q}$ এর আউটপুট যথাক্রমে 1 এবং 0 অথবা 0 এবং 1 এই দুটোই হতে পারে। এটি গাণিতিক কোনো ব্যাপার নয়, পুরোপুরি বাস্তব একটি সার্কিট, আমরা তাহলে কোন আউটপুটটি পাব?

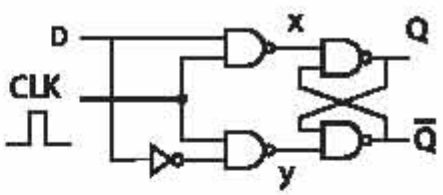
উত্তরটি কিন্তু বেশ সহজ। এটি নির্ভর করে x = 1 এবং y = 1 অবস্থাটির আগের অবস্থা কী। যদি ঠিক আগের অবস্থা x = 0 এবং y = 1 হয়ে থাকে তাহলে Q হবে 1 (এবং $\bar{Q}$ হবে তার বিপরীত অর্থাৎ 0) এবং যদি আগের অবস্থা x = 1 এবং y = 0 হয়ে থাকে তাহলে Q হবে 0 (এবং $\bar{Q}$ হবে তার বিপরীত অর্থাৎ 1) ট্রুথ টেবিলে সেটা এভাবে দেখানো যেতে পারে :

	x	y	Q	$\bar{Q}$
↓	1	0	0	1
	1	1	0	1

	x	y	Q	$\bar{Q}$
↓	0	1	1	0
	1	1	1	0

আমরা ইচ্ছা করলে এভাবেও বলতে পারি, x এবং y দুটোকেই 1 করে দিয়ে আমরা x এর মান $\bar{Q}$ এ এবং y এর মান Q এর মাঝে সংরক্ষণ করে রেখেছি। কাজেই এই ফ্লিপফ্লপ ব্যবহার করার সাধারণ নিয়ম হচ্ছে x এবং y দুটোকে সবসময়েই 1 হিসেবে রাখা এবং প্রয়োজন অনুযায়ী শুধু ক্ষুদ্র একটি সময়ের জন্য x অথবা y কে 0 করা। x কে 0 করা হলে Q হবে 1 এবং y কে 0 করা হলে Q হবে 0 (এবং $\bar{Q}$ হবে Q এর বিপরীত)।

এটি সাধারণত কীভাবে করা হয় সেটি 3.22 চিত্রের সার্কিটে দেখানো হলো। D ইনপুটটি x এবং y-এর মাঝে সরাসরি না দিয়ে দুটি বাড়তি NAND গেট দিয়ে দেয়া হচ্ছে। নিচের NAND গেটের আগে একটি ইনভার্টার দেওয়ার কারণে সবসময়েই x এবং y একটি 1 অন্যটি তার বিপরীত 0 সিগন্যাল পেয়ে থাকে। তবে যতক্ষণ CLK ইনপুটটি 0 থাকবে ততক্ষণ D ইনপুটের মান এই বাড়তি NAND গেটের ভেতর দিয়ে x এবং y পর্যন্ত পৌঁছাতে পারে না। D ইনপুটের মান যাই থাকুক না কেন, CLK ইনপুটটি 0 হলে x এবং y ইনপুটের মান সবসময় 1 থাকবে। D ইনপুটের মান ফ্লিপফ্লপে লোড করতে হলে অল্প সময়ের জন্য CLK ইনপুটটির মান 1 করতে হয়। মানটি লোড করার পর সেটি আবার 0 করে ফেলা হয়।

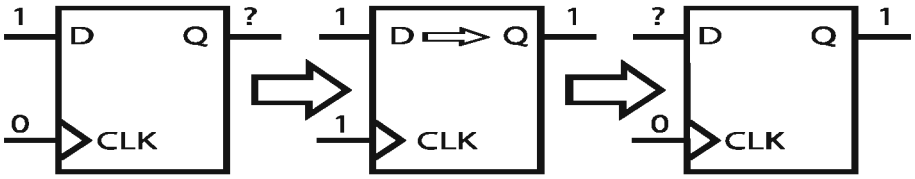


চিত্র 3.22 : DQ ফ্লিপফ্লপ-এর আভ্যন্তরীণ গঠন

ধরা যাক D এর মান 1 করে একটি ক্ষুদ্র সময়ের জন্য CLK এর মান 1 করা হলো (ইলেক্ট্রনিক্সের ভাষায় "একটি CLK পালস দেওয়া হলো")। তাহলে সেই পালসের সময়টুকুতে x হবে 0, y হবে 1 কাজেই Q হবে 1 (স্বাভাবিকভাবে $\bar{Q}$-এর মান হবে Q-এর বিপরীত, অর্থাৎ 0) পালসটুকু শেষ হওয়ার পর যেহেতু x এবং y দুটোর মানই আবার 1 হয়ে যাবে, তাই ফ্লিপফ্লপের নিয়ম অনুযায়ী Q-এর মান 1 হিসেবে সংরক্ষিত থেকে যাবে। অর্থাৎ মনে হবে D তে যে 1 মান দেওয়া হয়েছে সেটি CLK পালস দিয়ে Q তে লোড করা হয়েছে। ঠিক একইভাবে D তে 0 দিয়ে একটি CLK পালস দেওয়া হলে Q হবে 0 এবং মনে হবে D-এর 0 সিগন্যালটি Q তে লোড করা হয়েছে।

এই ধরনের সার্কিটে নাম DQ ফ্লিপফ্লপ। 3.23 চিত্রে বিষয়টি ব্যাখ্যা করা হয়েছে, সহজ করার জন্য $\bar{Q}$ দেখানো হয়নি। এক কথায় বলা যায়, D এর মানটি একটি পালস দিয়ে Q-এ নিয়ে আসা হয়, মানটি সেখানে

সংরক্ষিত থাকে, D-এর মান পরিবর্তন করা হলেও Q-এর মানের পরিবর্তন হয় না। শুধু আরেকটি CLK পালস দিয়ে D-এর নতুন মান Q তে লোড করা যাবে। এই ধারণাটি নিয়ে তোমাদের মাঝে যেন কোনো

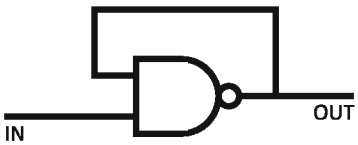


চিত্র 3.23 : এখানে D থেকে Q তে 1 লোড করার পদ্ধতিটি দেখানো হচ্ছে। শুরুতে D তে 1 দেওয়ার পরও Q-এর মানের কোনো পরিবর্তন নেই। পরের ধাপে যখন CLK-এ একটি পালস (1) দেওয়া হলো তখন D-এর মানটি Q তে চলে গেল। শেষ ধাপে CLK-এর মান আবার 0 করার পর D তে যে মানই দেয়া হোক Q-এর মানের কোনো পরিবর্তন হবে না

বিভ্রান্তি না থাকে কারণ এর পরের সব কয়টি সার্কিটে আমরা DQ ফ্লিপফ্লপ ব্যবহার করব।

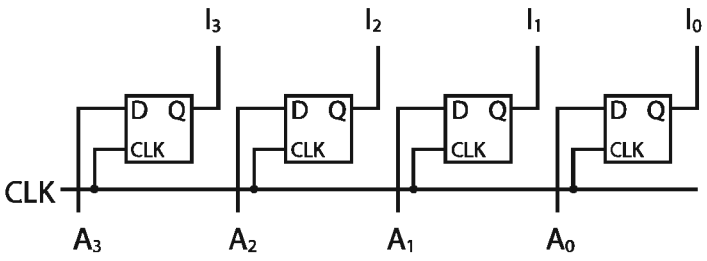
নিজে কর : দুটি NAND গেট ব্যবহার না করে দুটি NOR গেট ব্যবহার করে একটি ফ্লিপফ্লপ তৈরি করা হলে তার ট্রুথ টেবিল কেমন হবে?

নিজে কর : পাশের ছবিতে দেখানো গেটটির ইনপুট 0 হলে আউটপুট কী হবে? ইনপুট 1 হলে আউটপুট কী হবে? (উল্লেখ্য একটি গেটের ইনপুটে সিগন্যাল দেওয়ার সাথে সাথে আউটপুটে মান পাওয়া যায় না, আউটপুটে মান আসতে প্রায় 10ns-এর মতো সময় দরকার হয়। এই পদ্ধতিতে একাধিক গেট ব্যবহার করে খুব সহজে CLK তৈরি করা যায়।)



প্যারালাল লোড রেজিস্টার

3.24 চিত্রে চারটি DQ ফ্লিপফ্লপ পাশাপাশি বসিয়ে একটি সার্কিট তৈরি করা হয়েছে। যেহেতু একই সাথে চারটি ফ্লিপফ্লপে CLK পালস দেওয়া হয়, তাই এই চারটি ফ্লিপফ্লপ একই সাথে চার বিট তথ্য সংরক্ষণ করতে পারে। যদি A_0, A_1, A_2 এবং A_3 তে চার বিট তথ্য দেওয়া হয় তাহলে সেই চার বিট তথ্য CLK পালস দেওয়ার সাথে সাথে I_0, I_1, I_2 এবং I_3 তে সংরক্ষিত হয়ে যাবে। তখন A_0, A_1, A_2 এবং A_3-এর বিটগুলো পরিবর্তিত হলেও I_0, I_1, I_2 এবং I_3 তে সংরক্ষিত তথ্যের কোনো পরিবর্তন হবে না। শুধু নতুন একটি CLK পালস দেওয়া হলেই পরিবর্তিত A_0, A_1, A_2 এবং A_3 এর মান I_0, I_1, I_2 এবং I_3 তে লোড হবে। ফ্লিপফ্লপের সংখ্যা বাড়িয়ে পুরো এক বাইট কিংবা কয়েক বাইট তথ্য একসাথে রাখা সম্ভব।



চিত্র 3.24 : প্যারালাল লোড রেজিস্টার

এই ধরনের সার্কিটকে প্যারালাল লোড রেজিস্টার বলে।

শিফট রেজিস্টার

প্যারালাল লোড রেজিস্টারে DQ ফ্লিপফ্লপগুলোতে সিগন্যাল একই সাথে লোড করা হয়। ভিন্ন আরেক ধরনের ফ্লিপফ্লপ আছে যেখানে ফ্লিপফ্লপগুলোর আউটপুট Q অন্যটির ইনপুট D-এর সাথে সংযুক্ত করে প্রতি ক্লক পালসে এক ফ্লিপফ্লপের সিগন্যাল পরের ফ্লিপফ্লপে পাঠানো যায়। এই ধরনের রেজিস্টারকে শিফট রেজিস্টার বলে। শিফট রেজিস্টারের ইনপুটে সিরিয়াল ডেটা দিয়ে আউটপুটে প্যারালাল ডাটা পাওয়া যায়। 3.25 চিত্রে একটি শিফট রেজিস্টারের সার্কিট দেখানো হলো।

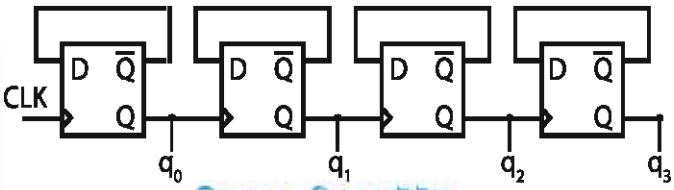
চিত্র 3.25 : শিফট রেজিস্টার

৩.৭.১২ কাউন্টার (Counter)

কাউন্টার এক ধরনের ডিজিটাল সার্কিট যেটি গণনা করতে পারে। আমরা DQ ফ্লিপফ্লপ দিয়ে খুব সহজে কাউন্টার তৈরি করতে পারি। DQ ফ্লিপফ্লপের আউটপুট Q এবং $\bar{Q}$ দুটোই থাকে তবে যেহেতু রেজিস্টার তৈরি করার সার্কিটগুলোতে $\bar{Q}$ ব্যবহার করার দরকার হয়নি, তাই সার্কিটে ইচ্ছা করে $\bar{Q}$ টি দেখানো হয়নি। কাউন্টার তৈরি করার সময় Q এবং $\bar{Q}$ দুটো আউটপুটেরই প্রয়োজন হবে, তাই 3.26 চিত্রে দুটোই দেখানো হয়েছে। তবে সার্কিটটি সহজে আঁকার জন্য $\bar{Q}$ টি উপরে এবং Q টি নিচে আঁকা হলো। একটি খুবই সহজ কাউন্টারের সার্কিট 3.26 চিত্রে দেখানো হয়েছে।

টেবিল 3.7

CLK পালসের সংখ্যা	q_3	q_2	q_1	q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1



চিত্র 3.26 : রিপল কাউন্টার

এখানে যেটা সবচেয়ে গুরুত্বপূর্ণ সেটা হচ্ছে সব ফ্লিপফ্লপে কিন্তু একই CLK পালস দেওয়া হচ্ছে না। প্রথম ফ্লিপফ্লপটি আসল CLK পালস পেলেও অন্য ফ্লিপফ্লপগুলো তার আগের ফ্লিপফ্লপের আউটপুট Q-এর সিগন্যালকে তার CLK পালস হিসেবে ব্যবহার করছে।

সার্কিটে দেখানো না হলেও প্রথমে সবগুলো ফ্লিপফ্লপ রিসেট করে নিতে হবে যেন সব Q-এর মান হয় 0 (কাজেই সবগুলো $\bar{Q}$-এর মান হয় 1)। এবারে প্রতি CLK পালসে প্রথম ফ্লিপফ্লপের $\bar{Q}$-এর মান D-এর মধ্য দিয়ে Q তে লোড হবে।

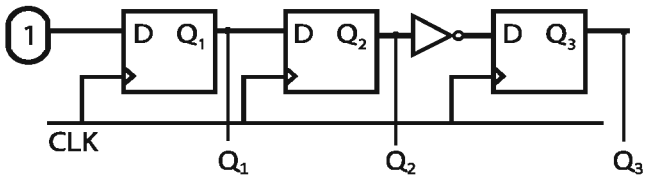
যেহেতু D-এর মানের বিপরীত মানটি অর্থাৎ $\bar{Q}$-এ লোড হয়, তাই প্রথম ফ্লিপফ্লপে Q-এর মান একবার 0 এবং পরের বার 1 হতে থাকবে। পাশের টেবিলে সেটা দেখানো হয়েছে। (লক্ষ্য কর, টেবিলে প্রথম q_0-এর মান সবচেয়ে ডানদিকে বসিয়ে অন্যগুলো ক্রমান্বয়ে তার বামে বসানো হয়েছে) পরের ফ্লিপফ্লপ একই

ব্যাপারে ঘটবে তবে যেহেতু Q_0 কে Q_1 এর ক্লক হিসেবে ব্যবহার করা হচ্ছে তাই দ্বিতীয় ফ্লিপফ্লপের Q আউটপুট পরিবর্তিত হবে প্রকৃত CLK-এর দুটি পালস পরে পরে-যেটি টেবিলে দেখানো হয়েছে। একইভাবে পরের ফ্লিপফ্লপের আউটপুট পরিবর্তিত হবে প্রকৃত CLK-এর চার পালস পরে পরে।

টেবিলে প্রথম q_0-এর মান সবচেয়ে ডানদিকে বসিয়ে q_1, q_2, q_3 গুলো ক্রমান্বয়ে তার বামে বসানোর কারণে আমরা টেবিলের দিকে তাকালেই দেখতে পাব $q_3, q_2\ q_1\ q_0$ আসলে একটি কাউন্টারের আউটপুট যেটি ক্লক পালসকে বাইনারি সংখ্যা হিসেবে গুণছে। এই ধরনের কাউন্টারকে বলা হয় রিপল কাউন্টার।

রিপল কাউন্টার ছাড়াও আরো নানা ধরনের কাউন্টার রয়েছে যেগুলো নানাভাবে গণনা করতে পারে।

নিজে কর : এই তিনটি ফ্লিপফ্লপের Q_1, Q_2 এবং Q_3-এর মান যথাক্রমে 0, 1 এবং 1, তিনটি ক্লক পালসের পর Q_1, Q_2 এবং Q_3-এর মান কত হবে?



অনুশীলনী

বহুনির্বাচনি প্রশ্ন

১. ইউনিকোডে বিটের সংখ্যা কত?

ক. 4 খ. 8

গ. 16 ঘ. 32

২. ইউনিকোডে মোট কতগুলো ভিন্ন অক্ষরকে কোডের অন্তর্ভুক্ত করা যায়?

ক. 2^2 খ. 2^4

গ. 2^8 ঘ. 2^{16}

৩. 4, 8, C অণুক্রমটির পরের মান কত?

ক. D খ. F

গ. 10 ঘ. 16

৪. দশমিক সংখ্যা -12 এর 2's complement কত?

ক. 00001100 খ. 11111100

গ. 11110011 ঘ. 11110100

৫. $(1110.11)_2$ এর সমকক্ষ হেক্সাডেসিমাল সংখ্যা কোনটি?

ক. E.3 খ. E.8

গ. E.C ঘ. C.E

৬. যে গেটের সকল ইনপুট 0 হলে আউটপুট 1 হবে-

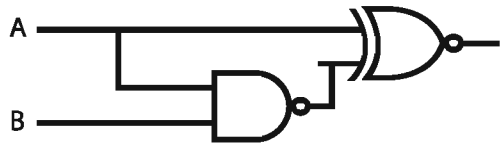
i. NAND
ii. NOR
iii. OR

নিচের কোনটি সঠিক?

ক. i ও ii খ. i ও iii

গ. ii ও iii ঘ. i, ii ও iii

চিত্রটি লক্ষ কর এবং ৭ ও ৮ নম্বর প্রশ্নের উত্তর দাও:



৭. F এর মান কোনটি?

ক. AB খ. $\overline{A}B$

গ. $A\overline{B}$ ঘ. $\overline{A\overline{B}}$

৮. XNOR এর স্থলে কোন গেট বসালে আউটপুট 0 হবে?

ক. AND খ. OR

গ. NAND ঘ. NOR

৯. $(110110)_2$ এর সমকক্ষ মান-

i. $(66)_8$

ii. $(54)_{10}$

iii. $(36)_{16}$

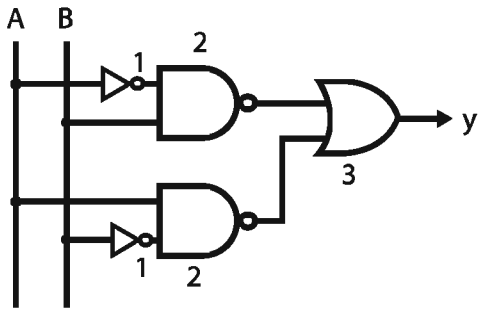
নিচের কোনটি সঠিক?

ক. i ও ii খ. i ও iii

গ. ii ও iii ঘ. i, ii ও iii

সৃজনশীল প্রশ্ন

১.



ক. 2 এর পরিপূরক কী?

খ. বাইনারি 1+1 ও বুলিয়ান 1+1 এক নয়- ব্যখ্যা কর।

গ. উদ্দীপক অনুসারে y এর সরলীকৃত মান নির্ণয় কর।

ঘ. উদ্দীপকের 2 ও 3 নম্বর চিহ্নিত গেট দু'টির পারস্পরিক পরিবর্তনে যে লজিক সার্কিট পাওয়া যায় তা বাইনারি যোগের বর্তনীতে ব্যবহার উপযোগী- যুক্তি দাও।

২. X, Y ও Z তিন বন্ধু বাজারে গিয়ে যথাক্রমে $(110110)_2$, $(36)_8$ এবং $(A9)_{16}$ টাকার বই কিনল।

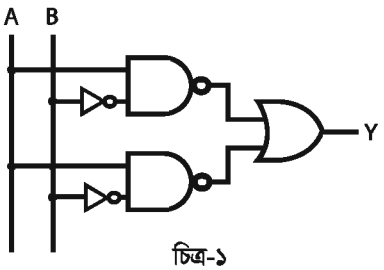
ক. কোড কী?

খ. ২-এর পরিপূরক গঠনের প্রধান কারণটি বর্ণনা কর।

গ. উদ্দীপকের "Z" এর ক্রয়কৃত বইয়ের মূল্য ডেসিমেল পদ্ধতিতে নির্ণয় কর।

ঘ. "Y" এর চেয়ে "X" বেশি মূল্যের বই কিনল। পরিপূরক পদ্ধতি ব্যবহার করে বিশ্লেষণ করে দেখাও।

৩.



চিত্র-১

চিত্র-২

ক. অ্যাডার কী?

খ. M (M+M) = M ব্যাখ্যা কর।

গ. চিত্র-১ এর মান সত্যক সারণিতে দেখাও।

ঘ. চিত্র-২ এর প্রতিনিধিত্বকারী গেট দিয়ে চিত্র-১ এর সমতুল্য সার্কিট বাস্তবায়ন করা সম্ভব কি? যুক্তিসহ বিশ্লেষণ কর।

৪. আইসিটি শিক্ষক ক্লাসে ছাত্রদের বললেন, কম্পিউটার A-কে সরাসরি বুঝতে পারে না, বরং একে একটি লজিক সার্কিটের সাহায্যে ৮বিটের বিশেষ সংকেতে রূপান্তর করে বুঝে থাকে। তিনি আরো বললেন, উক্ত সংকেতায়ন পদ্ধতিতে বাংলা কম্পিউটারকে বুঝানো যায় না। এ জন্য ভিন্ন একটি সংকেতায়ন পদ্ধতির প্রয়োজন হয়।

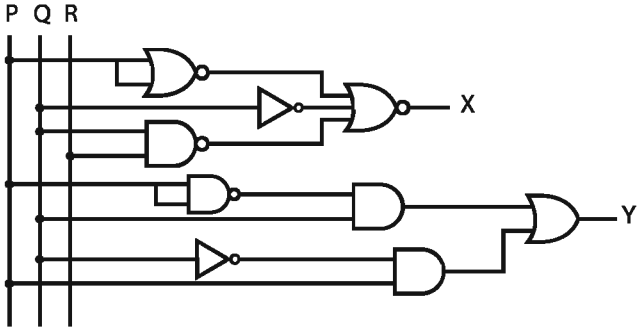
ক. ডিকোডার কী?

খ. শিফট রেজিস্টারের বৈশিষ্টটি বর্ণনা কর।

গ. উদ্দীপকে উল্লিখিত লজিক সার্কিটটির কাজের ধারা ব্যাখ্যা কর।

ঘ. উদ্দীপকের সংকেতায়ন পদ্ধতি দু'টির মধ্যে কোনটি অধিক সুবিধাজনক - তোমার মতামত যুক্তিসহ উপস্থাপন কর।

৫.



ক. কাউন্টার কী?

খ. নর গেটের সকল ইনপুট একই হলে গেটটি কীভাবে মৌলিক গেট হিসেবে কাজ করে তা ব্যাখ্যা কর।

গ. Y- এর মান সত্যক সারণিতে দেখাও।

ঘ. X-এর সরলীকৃত মান NOR গেটের সাহায্যে বাস্তবায়ন করে প্রমাণ কর যে, এটি সুবিধাজনক।