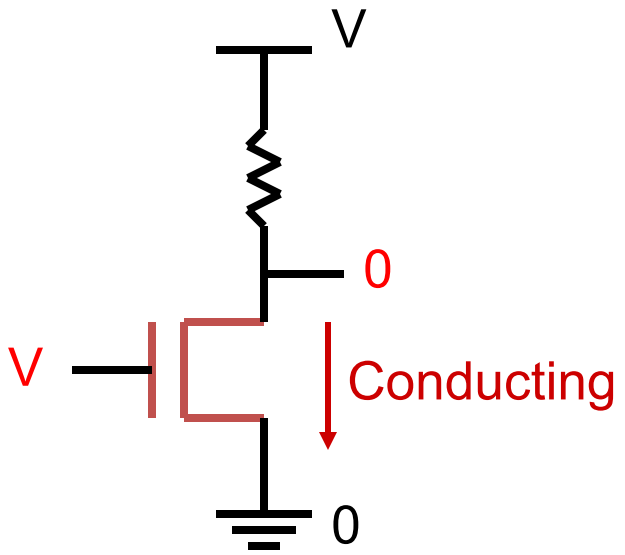
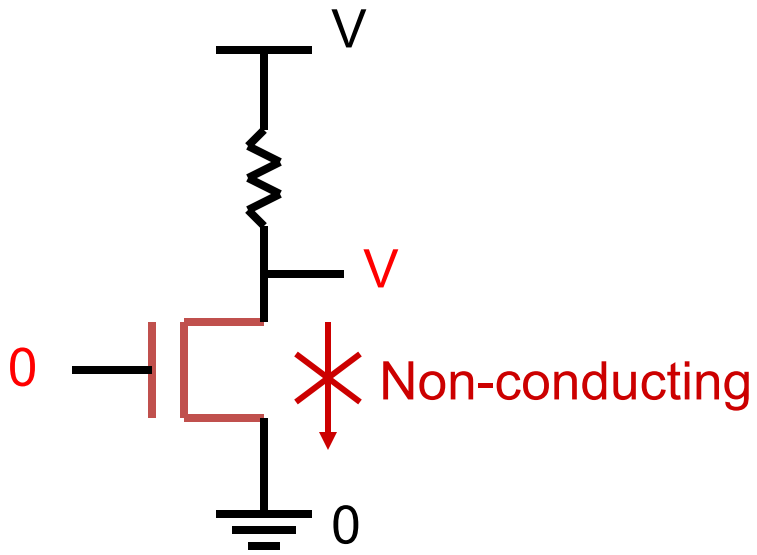
01076114
องค์ประกอบและสถาปัตยกรรมคอมพิวเตอร์
Computer Organization and Architecture

The Processor

Digital Design Basics

- ในระบบดิจิตอลจะมีระดับ voltage เพียง 2 ระดับ คือ High กับ Low (1 หรือ 0) ดังนั้นในคอมพิวเตอร์จึงประมวลผลโดยใช้เลขฐาน 2
- ส่วนประกอบหลักของคอมพิวเตอร์ คือ ทรานซิสเตอร์ ซึ่งจะทำงานเหมือนกับสวิตซ์ปิดเปิด
- คำถาม : รูปข้างล่างนี้ ทำงานเหมือนเกตชนิดใด





Logic Blocks

- การออกแบบ Hardware ก็คล้ายกับการเขียนโปรแกรม โดยจะมีการแบ่งเป็นส่วนๆ เรียกว่า Logic Blocks โดยแต่ละบล็อกก็จะมี Input เพื่อสร้าง Output ตามต้องการ โดยบล็อกอาจจะซับซ้อน หรือ อาจจะเป็นบล็อกง่ายๆ ที่มีทรานซิสเตอร์เพียงไม่กี่ตัว
- คำถาม : Logic Blocks ที่เป็น Combination ต่างจาก Sequential อย่างไร
- คำตอบ : Logic Blocks ที่เป็น Combination นั้น Output จะเป็น function ของ Input โดยตรง แต่ Logic Blocks ที่เป็น Sequential เนื่องจากใน Logic Blocks จะมี memory (state) อยู่ด้วย จึงส่งผลกระทบต่อ Output
- Logic Blocks พื้นฐานจะเรียกว่า Gate ได้แก่ AND, OR, NOT, ฯลฯ
- สำหรับในบทนี้จะกล่าวถึงเฉพาะ Combination Circuit

Truth Table

- ตาราง Truth Table ทำหน้าที่กำหนด Output ของ Logic Blocks สำหรับ Input แต่ละแบบ
- จงเขียน Truth Table โดยกำหนดให้มี 3 อินพุท คือ A, B, C โดยมี Output คือ E โดยกำหนดให้ Output เป็น True เมื่อมี**เพียง** 2 Input เป็น true

A	B	C	E

Truth Table

- ตาราง Truth Table ทำหน้าที่กำหนด Output ของ Logic Blocks สำหรับ Input แต่ละแบบ
- จงเขียน Truth Table โดยกำหนดให้มี 3 อินพุท คือ A, B, C โดยมี Output คือ E โดยกำหนดให้ Output เป็น True เมื่อมี**เพียง** 2 Input เป็น true

A	B	C	E
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Boolean Algebra

- เป็นคณิตศาสตร์ ที่กำหนดให้มีข้อมูลเพียง 2 ประเภท คือ True และ False และ Operator เพียง 3 ตัว ได้แก่
  - OR : symbol + , X = A + B ➔ X is true if at least one of A or B is true
  - AND : symbol . , X = A . B ➔ X is true if both A and B are true
  - NOT : symbol , X = $\overline{A}$ ➔ X is the inverted value of A

Boolean Algebra Rules

- Identity law : $A + 0 = A \quad ; \quad A \cdot 1 = A$
- Zero and One laws : $A + 1 = 1 \quad ; \quad A \cdot 0 = 0$
- Inverse laws : $A \cdot \overline{A} = 0 \quad ; \quad A + \overline{A} = 1$
- Commutative laws : $A + B = B + A \quad ; \quad A \cdot B = B \cdot A$
- Associative laws : $A + (B + C) = (A + B) + C$

  $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
- Distributive laws : $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$

  $A + (B \cdot C) = (A + B) \cdot (A + C)$
- DeMorgan's Laws : $\overline{A + B} = \overline{A} \cdot \overline{B}$

  $\overline{A \cdot B} = \overline{A} + \overline{B}$

Boolean Equation

- จากสไสด์ที่ 5 Output E เป็น True เมื่อมีเพียง 2 ใน 3 Input (A, B, C) เป็น true
- สามารถจะเขียนสมการได้หลายแบบ เช่น
- หากกำหนดว่า เฉพาะ 2 ใน 3 จะต้องเป็น true จะเป็น 3 ไม่ได้ สามารถเขียนเป็นสมการได้ดังนี้

$$E = ((A \cdot B) + (B \cdot C) + (A \cdot C)) \cdot \overline{(A \cdot B \cdot C)}$$

- หรือหากระบุ 3 Case ที่ทำให้เงื่อนไขเป็นจริง ก็จะเขียนได้ดังนี้

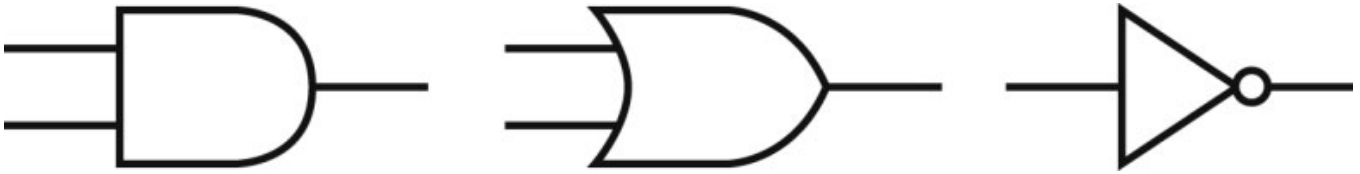
$$E = (A \cdot B \cdot \overline{C}) + (A \cdot C \cdot \overline{B}) + (C \cdot B \cdot \overline{A})$$

- **สมการทั้ง 2 Equivalent กันหรือไม่?**

Boolean Equation

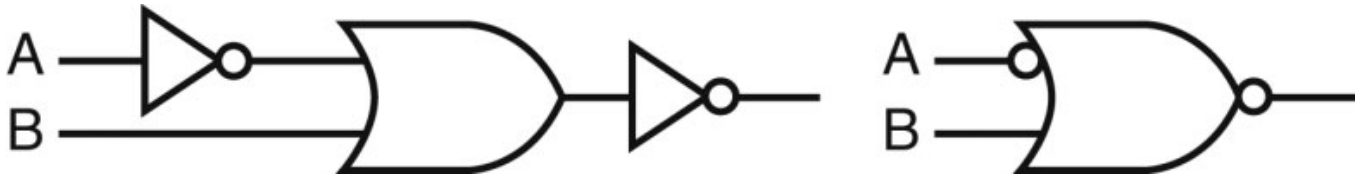
- $E = ((A \cdot B) + (B \cdot C) + (A \cdot C)) \cdot \overline{(A \cdot B \cdot C)}$
- $E = ((A \cdot B) + (B \cdot C) + (A \cdot C)) \cdot (\overline{A} + \overline{B} + \overline{C})$
- $E = ((A \cdot B).(\overline{A} + \overline{B} + \overline{C})) + ((B \cdot C).(\overline{A} + \overline{B} + \overline{C}))$
  $+ ((A \cdot C).(\overline{A} + \overline{B} + \overline{C}))$
- พิจารณาเฉพาะเทอมแรก $= A.B.\overline{A} + A.B.\overline{B} + A.B.\overline{C}$
- จาก Inverse Law $A.\overline{A} = 0$ ดังนั้นจะเหลือแค่ A.B.C
- เมื่อทำแบบเดียวกันทั้ง 3 เทอม จะได้
- $E = (A \cdot B \cdot \overline{C}) + (A \cdot C \cdot \overline{B}) + (C \cdot B \cdot \overline{A})$

Pictorial Representation



Source: H&P textbook

What logic function is this?



Source: H&P textbook

Sum of Products

- เป็นวิธีการที่จะสร้าง Logic Blocks ใดๆ โดยใช้ Operator AND OR หรือ NOT
  - เริ่มจากสร้าง Truth Table
  - ในแต่ละ Output ที่เป็น True ให้แทนค่า Input ที่ทำให้เกิด True เป็น Products
  - จากนั้นก็ Sum Products ทั้งหมด ก็จะได้สมการของ Logic Blocks

A	B	C	E
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(A . B . C) + (A . C . B) + (C . B . A)

- สามารถใช้ “product of sums” ก็ได้
- ทุกสมการ Boolean สามารถจะสร้างเป็น product of sums หรือ sum of products ก็ได้

PLA

- วงจรใดๆ สามารถทำเป็น array of AND และ array of OR

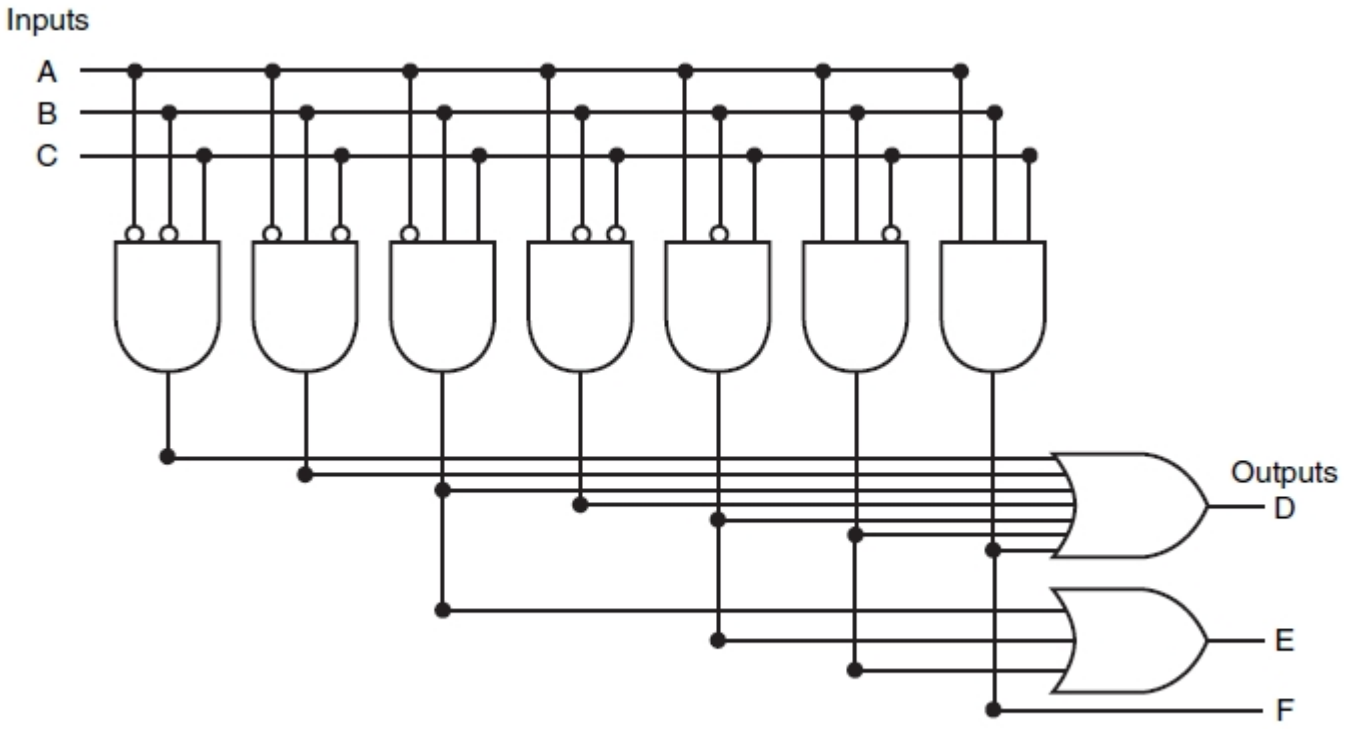
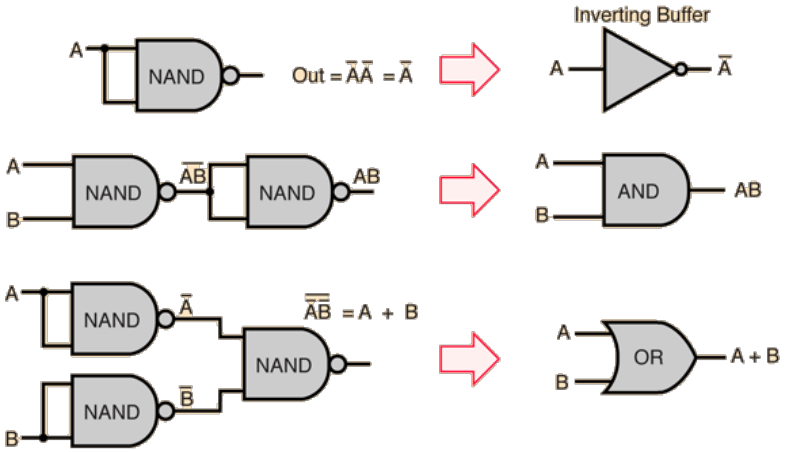
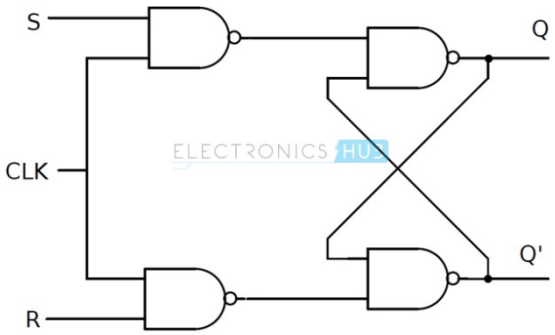


FIGURE C.3.4 The PLA for implementing the logic function described in the example.

NAND and NOR

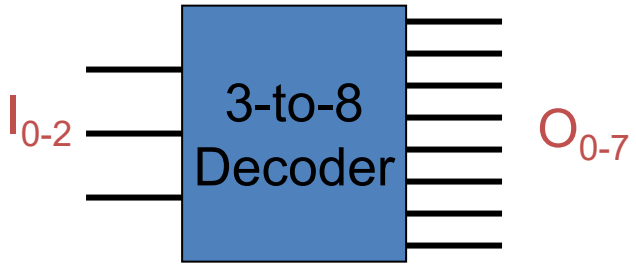
- NAND คือ Not ของ AND : A nand B = $\overline{A \cdot B}$
- NOR คือ Not ของ OR : A nor B = $\overline{A + B}$
- ในการสร้าง Logic Function เกท NAND และ Nor มีความสำคัญมาก เนื่องจากสามารถจะสร้างฟังก์ชันได้หลากหลายโดยใช้เกทเพียง 2 ตัวนี้ บางครั้งเรียกว่า *universal gates*





Common Logic Blocks - Decoder

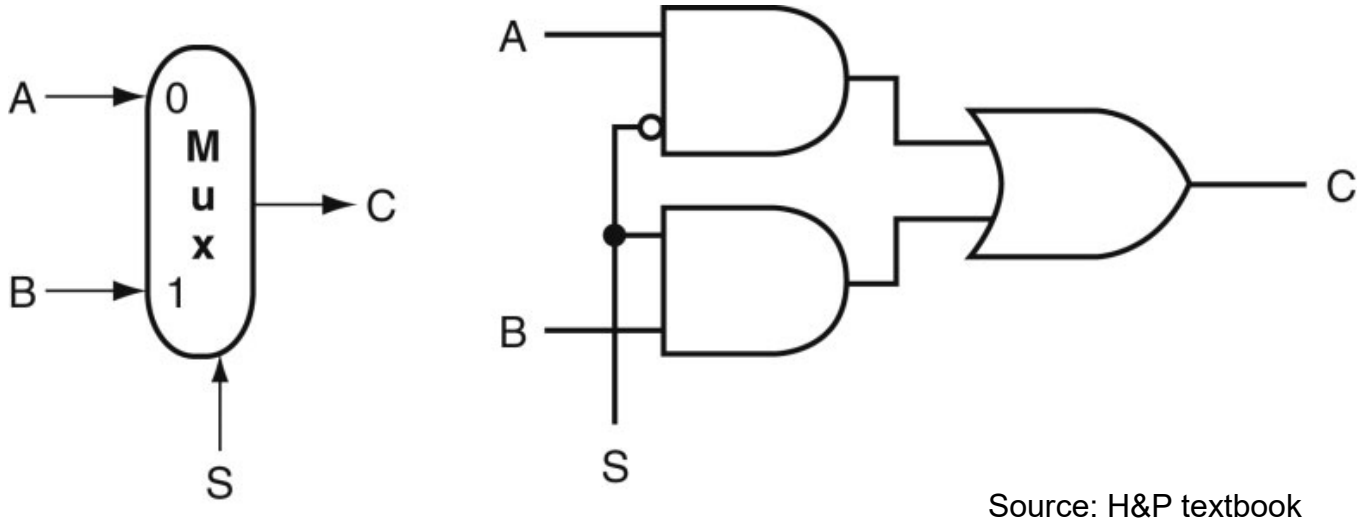
- เป็นวงจรที่มีใช้งานบ่อย ทำหน้าที่รับ N Input และสร้าง 2^N Output (เช่น memory)



I_0	I_1	I_2	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_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

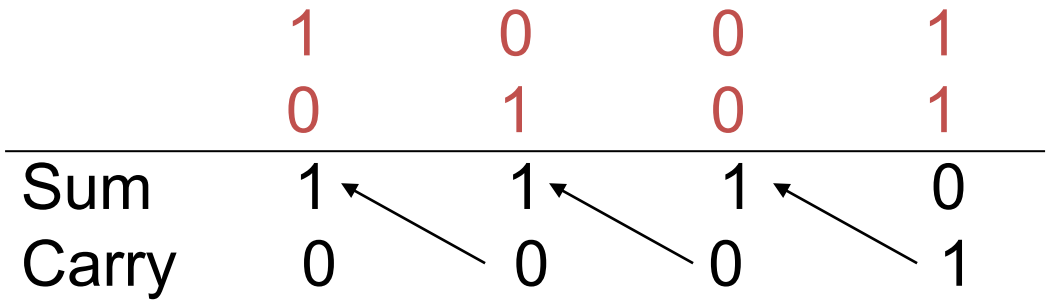
Common Logic Blocks - Multiplexer

- Multiplexer หรือ Selector ทำหน้าที่เลือก 1 ใน Input ให้ออกที่ Output
- Selector N Bit จะเลือกได้ 2^N Input หรือ S = $\log_2 N$



Source: H&P textbook

Common Logic Blocks - Adder



Truth Table for the above operations:

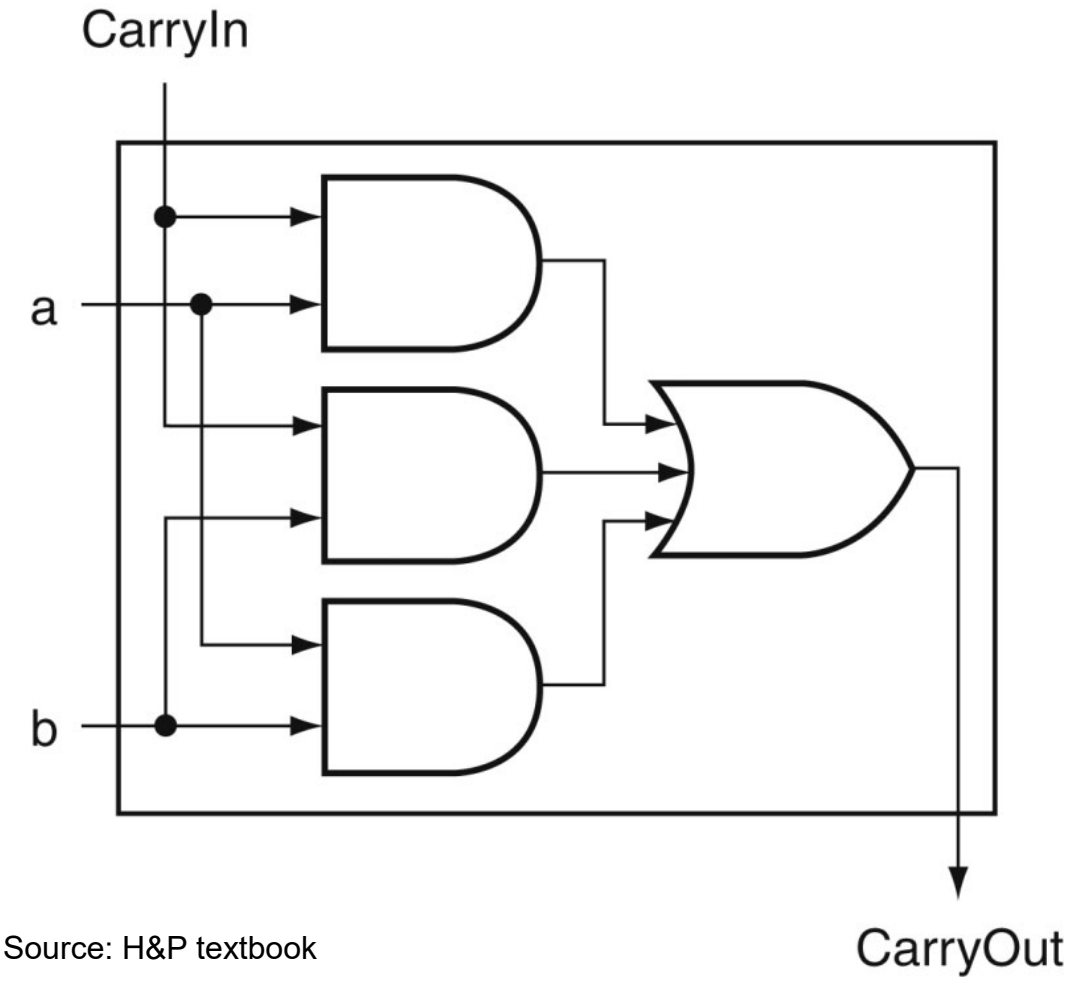
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Equations:

$$\text{Sum} = Cin \cdot \overline{A} \cdot \overline{B} + B \cdot \overline{Cin} \cdot \overline{A} + A \cdot \overline{Cin} \cdot \overline{B} + A \cdot B \cdot Cin$$

$$\text{Cout} = A \cdot B \cdot Cin + A \cdot B \cdot \overline{Cin} + A \cdot Cin \cdot \overline{B} + B \cdot Cin \cdot \overline{A}$$
$$= A \cdot B + A \cdot Cin + B \cdot Cin$$

Carry Out Logic



Source: H&P textbook

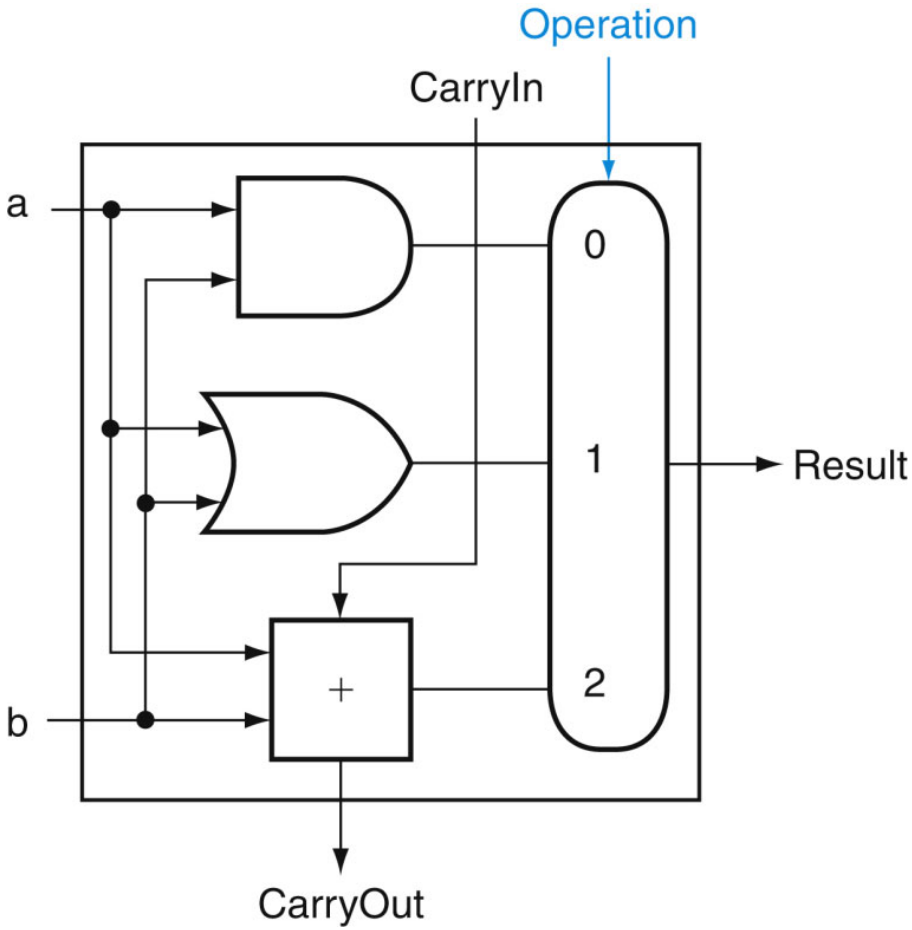
Equations:

$$\text{Sum} = Cin \cdot \overline{A} \cdot \overline{B} + B \cdot \overline{Cin} \cdot \overline{A} + A \cdot \overline{Cin} \cdot \overline{B} + A \cdot B \cdot Cin$$

$$\begin{aligned} \text{Cout} &= A \cdot B \cdot Cin + A \cdot B \cdot \overline{Cin} + A \cdot Cin \cdot \overline{B} + B \cdot Cin \cdot \overline{A} \\ &= A \cdot B + A \cdot Cin + B \cdot Cin \end{aligned}$$

1-Bit ALU with Add, Or, And

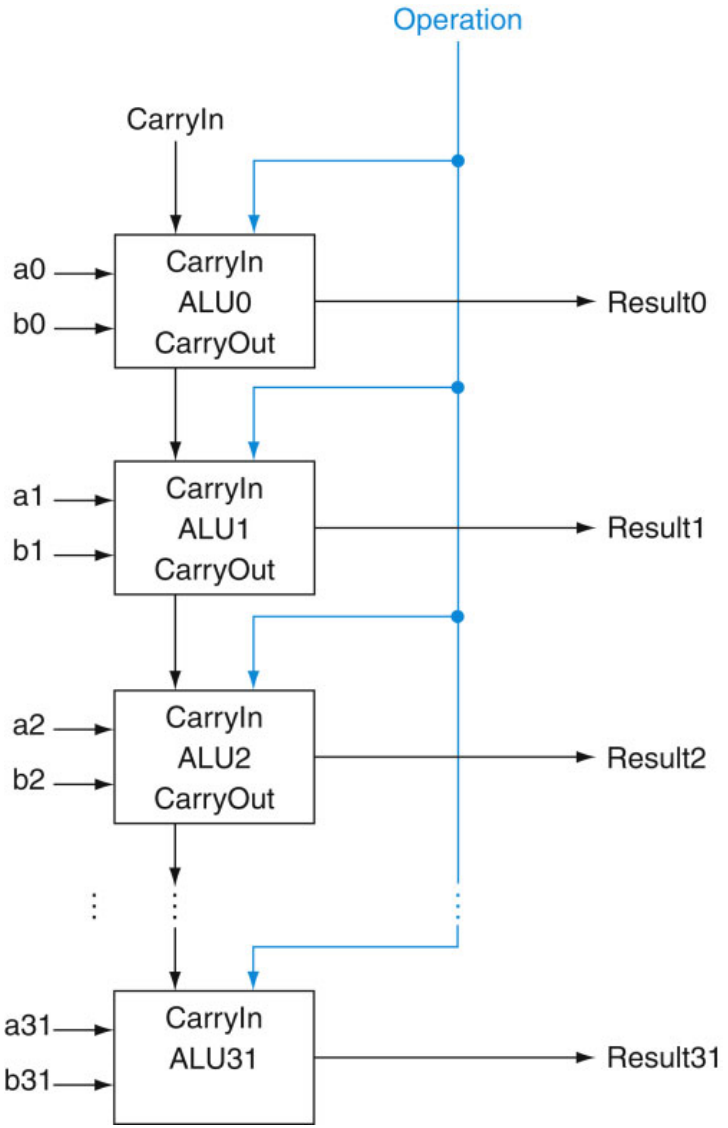
- เราสามารถสร้าง ALU ขนาด 1 บิตโดยนำโมดูล Adder พร้อมกับเกท AND และ OR มารวมกัน
- โดยมี Multiplexer เป็นตัวเลือกการกระทำผ่านสัญญาณ Operation



Source: H&P textbook

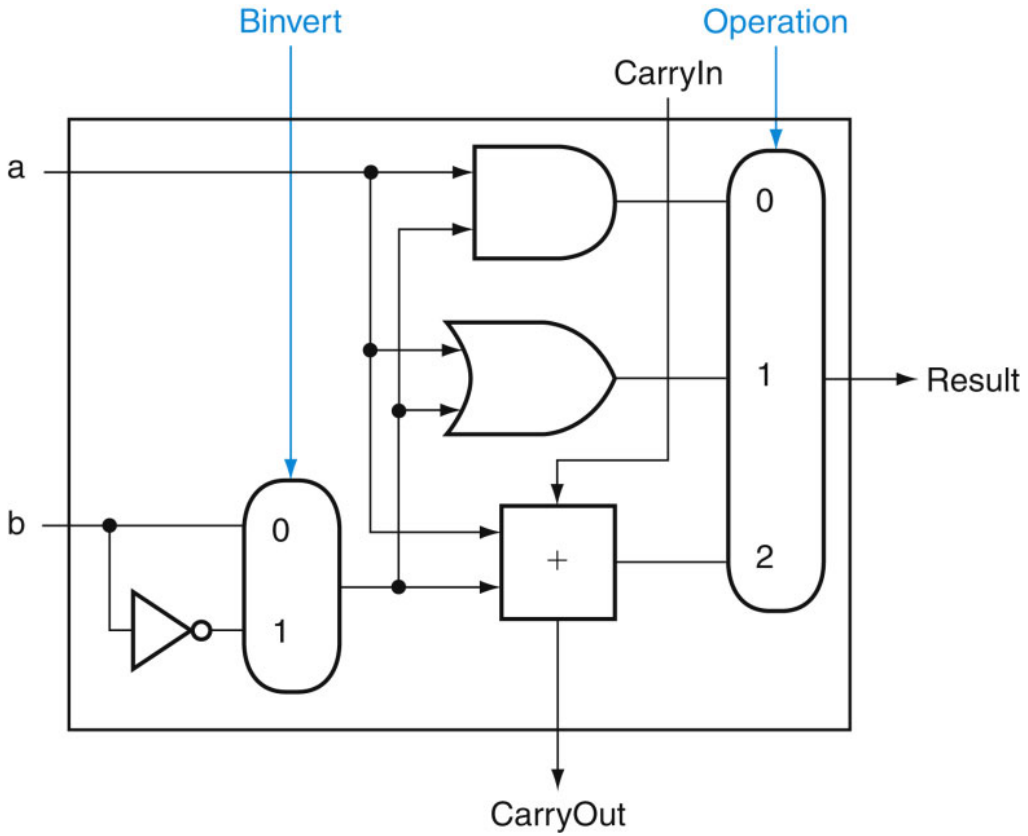
32-bit Ripple Carry Adder

- เมื่อนำ 1-bit ALU มาเชื่อมต่ออนุกรมกันไปเรื่อยๆ โดยนำ Carry Out ของบิตก่อนหน้ามาเข้า Carry In ของบิตต่อไป เราก็สามารถจะสร้าง 32-bit Adder ได้ (ตามรูป)
- การเชื่อมต่อตามรูป ทำให้การบวกไม่สามารถบวกพร้อมกันทุกบิต เนื่องจากบิตถัดไปจะต้องรอ Carry จากบิตก่อนหน้า



Incorporating Subtraction

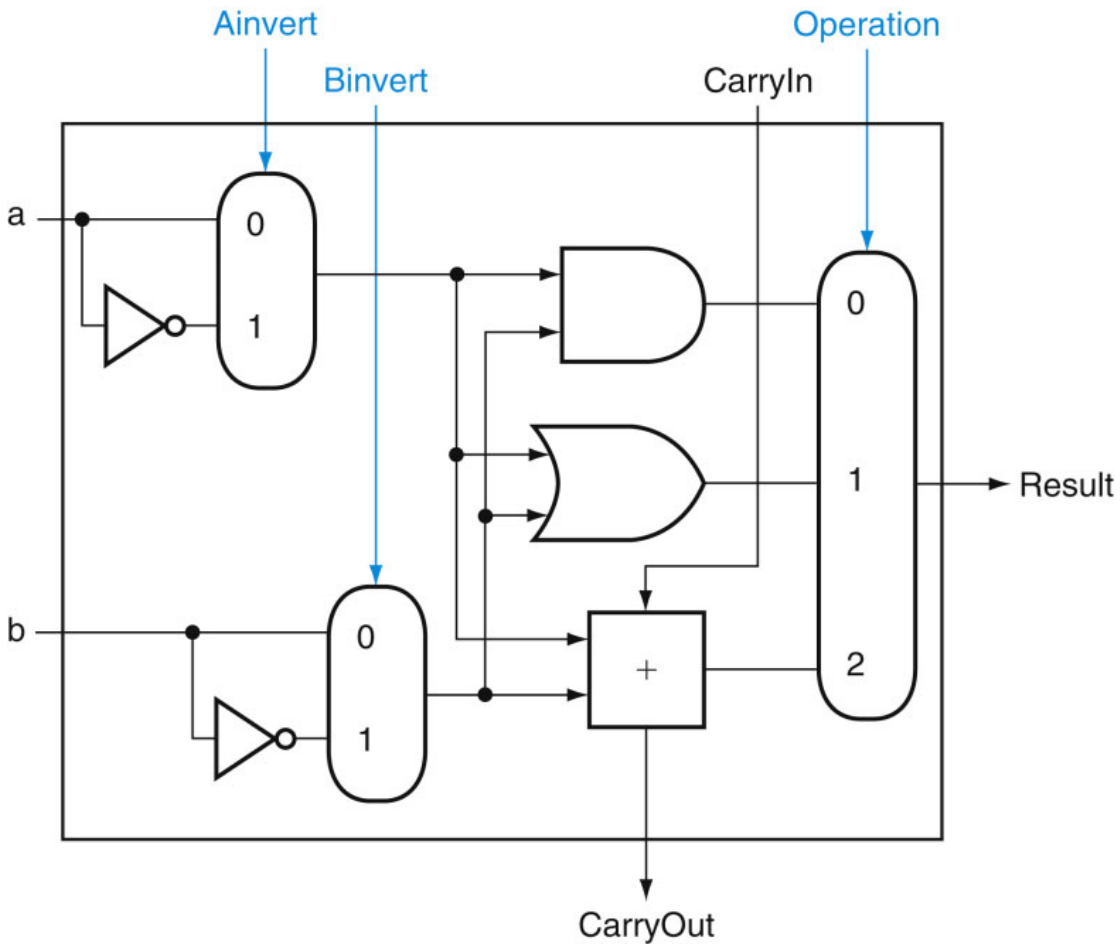
- เนื่องจาก A – B = A + (-B)
- -B = (Not B) + 1
- บล็อกของการลบ สร้างได้โดย Invert B จากนั้นก็ป้อน 1 เข้าที่ Carry In ก็จะเป็น +1
- และใช้สัญญาณ Binvert ทำหน้าที่เลือก MUX ว่าสัญญาณ b จะให้ invert หรือไม่



Source: H&P textbook

Incorporating NOR and NAND

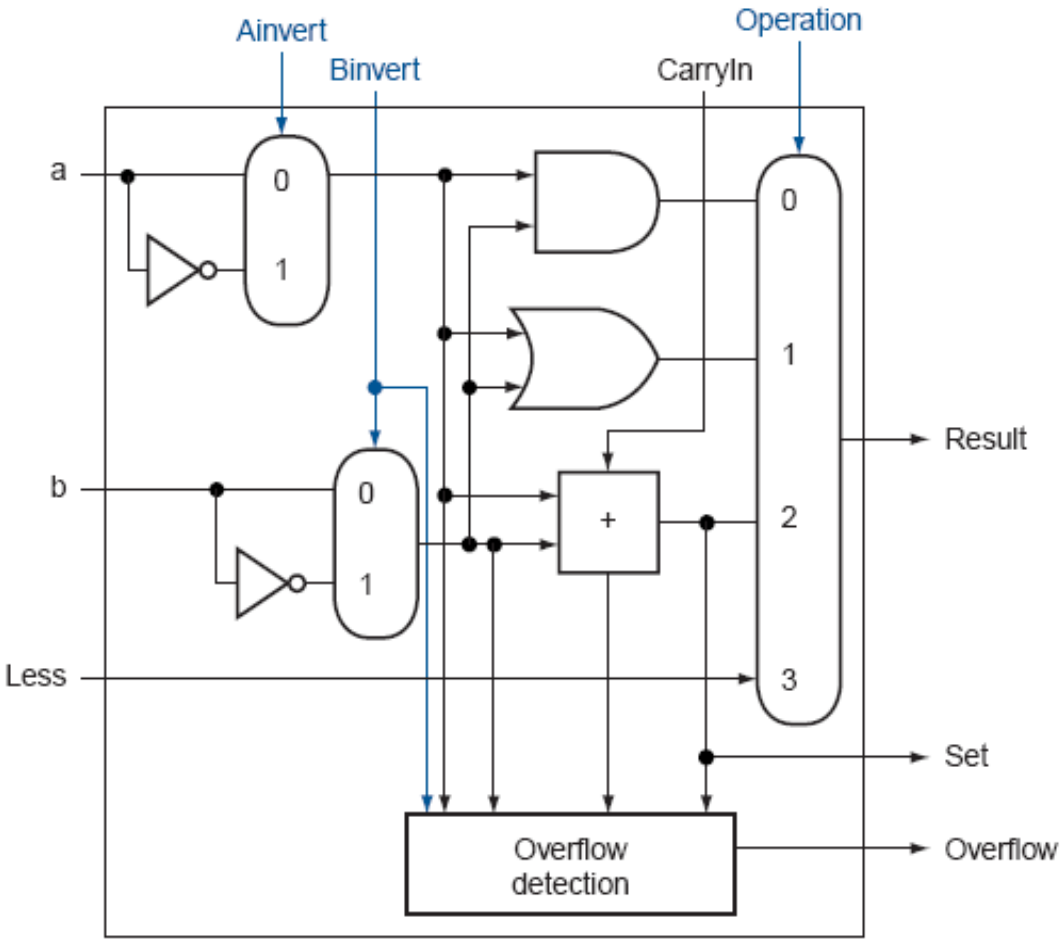
- เพื่อให้ทำงานได้ทั้ง Arithmetic และ Logical จึงต้องเพิ่ม NOR กับ NAND เข้าไปด้วย
- และเพิ่มสัญญาณ Ainvert เข้าไปอีกเส้นหนึ่ง
- $\overline{A.B} = \overline{A}+\overline{B}$ ดังนั้นหากทำ Operation NAND ต้อง invert ทั้ง A และ B แล้วใช้ OR
- $\overline{A+B} = \overline{A}.\overline{B}$ ดังนั้นหากทำ Operation NOR ต้อง invert ทั้ง A และ B แล้วใช้ AND



Source: H&P textbook

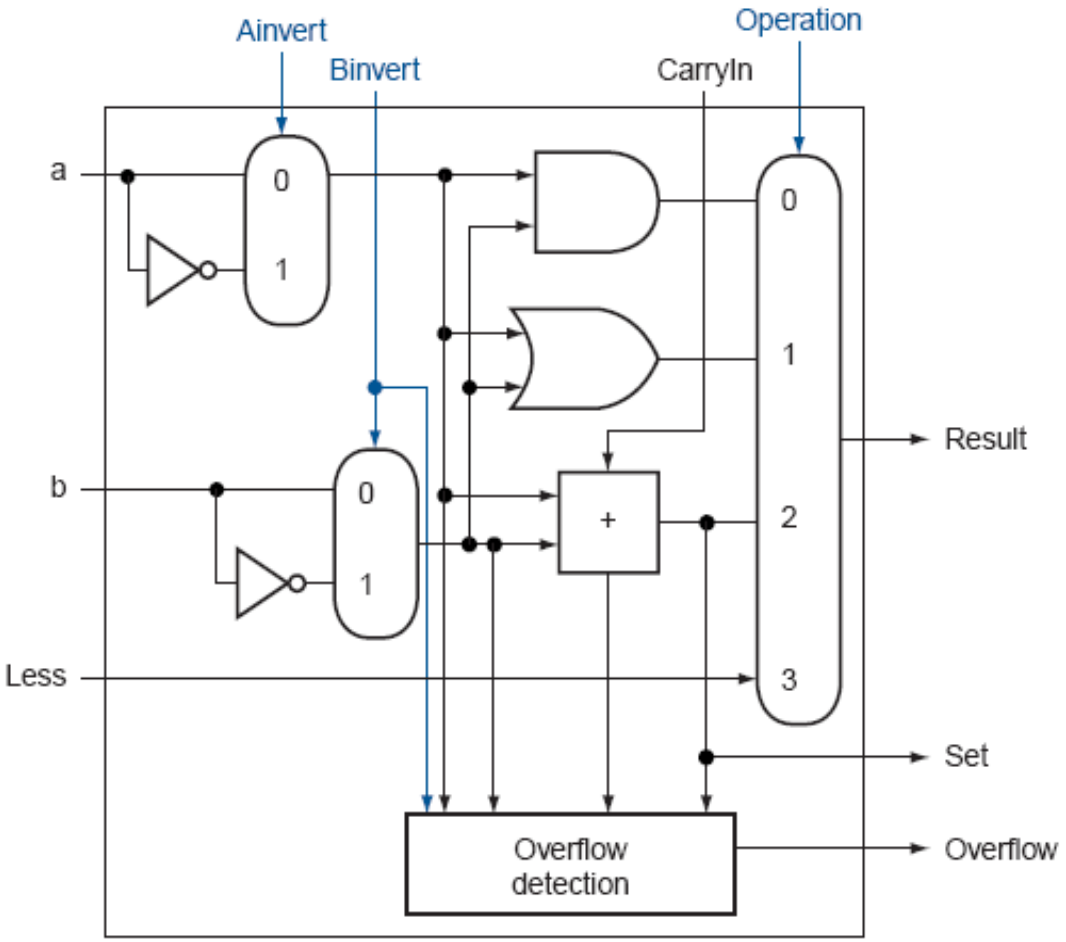
Incorporate SLT

- ในโพรเซสเซอร์ MIPS จะมีคำสั่ง SLT (set if less than) เช่น SLT r1,r2,r3 คือ ถ้า r1 น้อยกว่า r3 ทำให้ r1 เป็น 1
- คล้ายกับคำสั่ง CMP ใน ARM
- เช่น CMP r1,r2 ก็คือการใช้ r1-r2 ถ้า r1 > r2 ผลลัพธ์จะเป็นบวก แต่ถ้า r1 < r2 จะได้เป็นลบ
- จึงสามารถใช้การลบแต่ไม่เอาผลลัพธ์ ดูแต่ status อย่างเดียว
- ถ้าเป็นลบ บิตสูงสุดจะเป็น 1



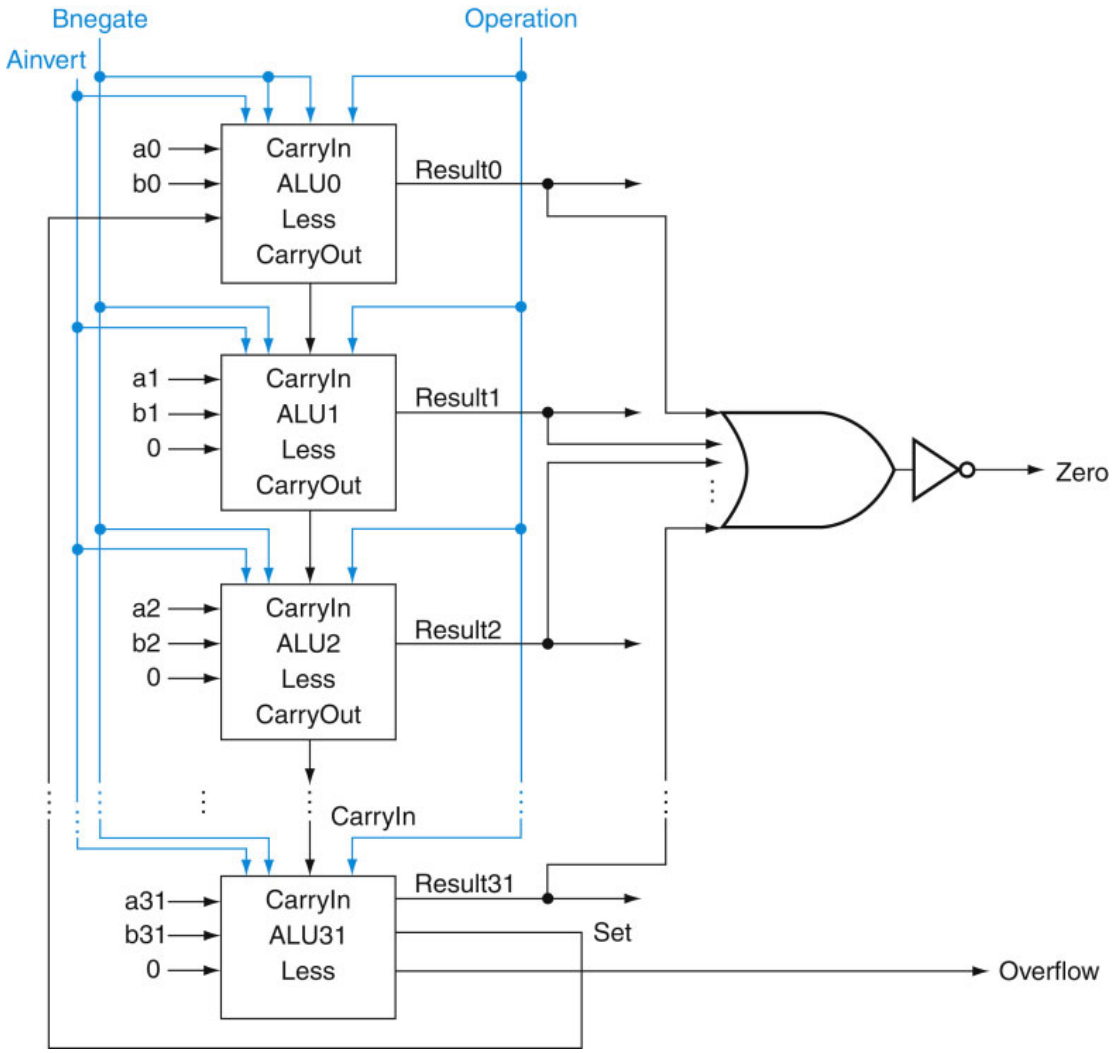
Incorporate SLT

- สัญญาณ Less จะป้อนให้กับบิตที่ 1-31 เพราะผลลัพธ์ของ SLT จะเกิดขึ้นที่บิต 0 เท่านั้น โดย Operation จะต้อง = 3
- ในบล็อกที่ 31 ของ ALU จะมีหน่วยพิเศษเพิ่มขึ้นมา คือ Overflow Detection ทำหน้าที่ตรวจสอบ overflow และ set (sign) เพื่อนำไปป้อนให้กับ Less ใน ALU บล็อกที่ 0



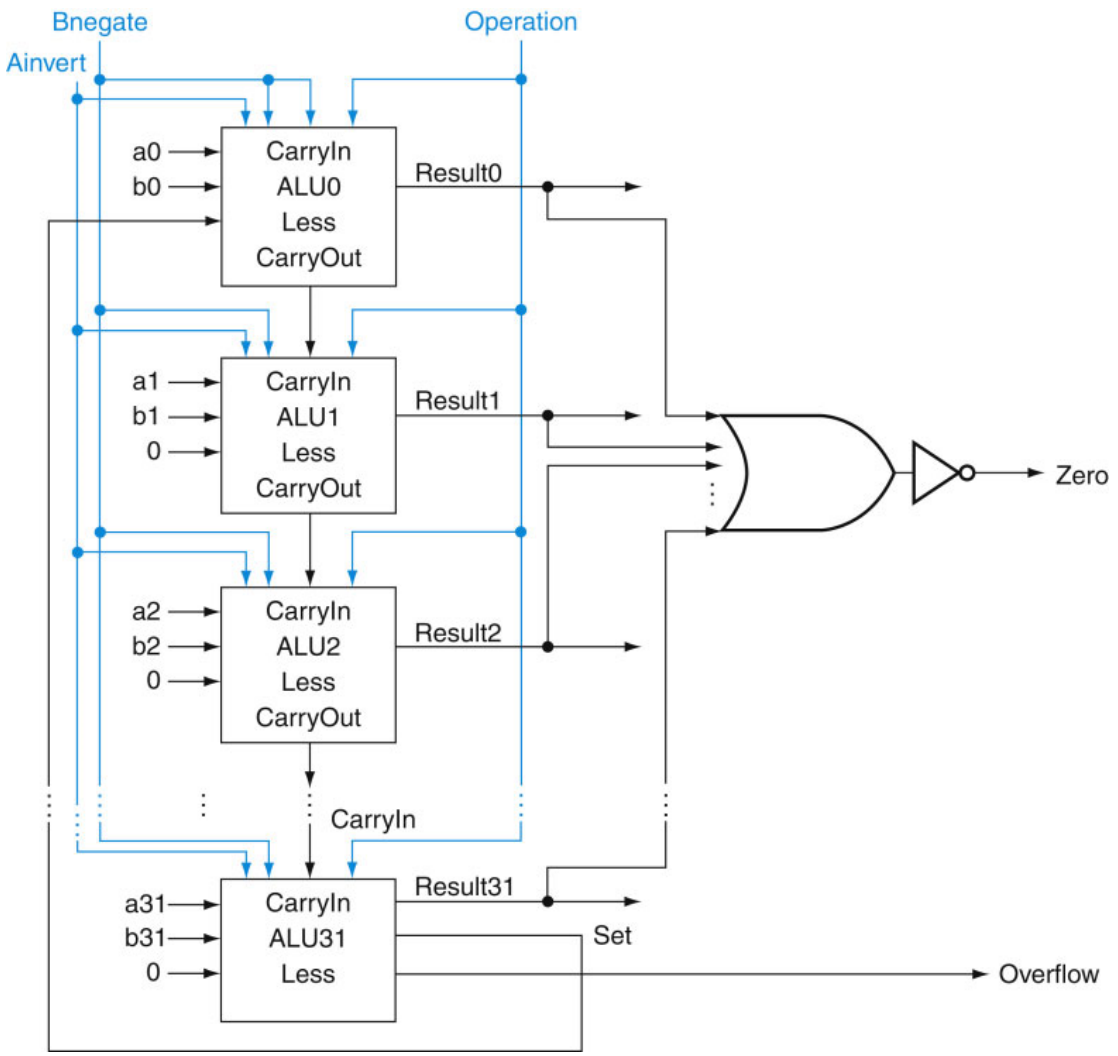
Incorporate BEQ

- สำหรับหลายๆ คำสั่งที่ทำงานเมื่อผลลัพธ์เป็น 0 เช่นคำสั่ง BEQ (Branch if Equal)
- การตรวจสอบ Zero จะใช้การลบเช่นกัน โดยจะใช้ข้อมูลจาก result ของทุกบิตมาผ่าน NOR gate (ต้องการเป็น 1)



Control Lines

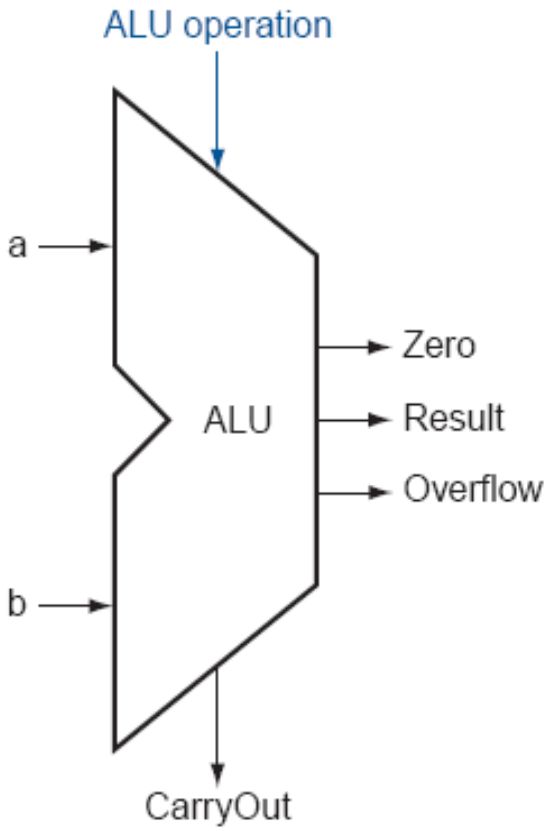
- ในการลบนั้นสำหรับ ALU 0 ถ้าสังเกต จะเห็นว่า Binvert กับ Carry In จะเป็น 0 พร้อมกัน (การบวก) และเป็น 1 พร้อมกัน (การลบ) ดังนั้นจึงเชื่อมเข้าหากันและตั้งชื่อใหม่
- ในส่วนต่อไป คือ ต้องพิจารณาว่าต้องใช้วงจรควบคุมอย่างไร จึงจะทำให้เกิดการทำงานตามที่ต้องการ



Control Lines

- หากจะมอง ALU เป็น Logic Blocks ก็สามารถเขียนได้ตามรูป
- โดย Blocks จะควบคุมผ่าน 3 สัญญาณ คือ ALU Operation, Ainvert และ Bnegate ซึ่งสามารถเขียน Truth Table ได้ดังนี้

	Ai	Bn	Op
AND	0	0	00
OR	0	0	01
ADD	0	0	10
SUB	0	1	10
SLT	0	1	11
NOR	1	1	00
NAND	1	1	01



Speed of Ripple Carry

- ที่ผ่านมาในวงจร Adder จะเป็นแบบบวกตามลำดับลงมา ทีละ 1 bitbox (ในขณะที่ AND, OR ทำงานได้พร้อมกันทุกบิต) ในแต่ละ bitbox มีประมาณ 5 gate delay (สมมติว่า 100 ps) ถ้า 32 บิตก็จะใช้เวลา 3,200 ps
- ได้กล่าวไว้ก่อนหน้านี้แล้วว่า Logic Equation ใดๆ สามารถเขียนเป็น Sum of Products ได้ ดังนั้นจึงมีความเป็นไปได้ที่จะสร้างวงจรบวก 32 บิตที่มี delay เพียง 2 gate
- แต่สำหรับ Input ขนาด 32+32 บิต โดยมี Output 32 บิต เมื่อสร้างเป็นสมการ ก็จะยาวมาก ทำให้มี gate จำนวนมาก และแต่ละ gate ก็มี Input จำนวนมาก วงจรก็จะมีขนาดใหญ่มาก ไม่คุ้มค่าที่จะสร้าง (Truth Table มีกี่ row?)
- จึงจำเป็นต้องหาวิธีการที่ gate ไม่มาก Input ไม่มาก และไม่ช้าเกินไป

Computing Carry In (Carry Look-ahead)

- เมื่อพิจารณา Carry Out (s.16)

Carry to ALO0 Input ALU0

  - CarryIn1 = b0.CarryIn0 + a0.CarryIn0 + a0.b0
  - CarryIn2 = b1.CarryIn1 + a1.CarryIn1 + a1.b1
    = b1.b0.c0 + b1.a0.c0 + b1.a0.b0 +
    a1.b0.c0 + a1.a0.c0 + a1.a0.b0 + a1.b1

  ...

  - CarryIn32 = ยาวมาก

- ตามที่กล่าวไปแล้ว วิธีการนี้แม้จะเร็วแต่ก็ใช้ gate จำนวนมาก และไม่มีประสิทธิภาพ

Generate and Propagate

- สามารถเขียนสมการในรูปทั่วไปได้
  - $C_{i+1} = a_i.b_i + a_i.C_i + b_i.C_i$

    $= (a_i.b_i) + (a_i + b_i).C_i$
- จะเห็นว่าคู่ของบิต จะสร้าง (generate) Carry เมื่อมีค่าเป็น 1 ทั้งคู่ หรืออาจจะถ่ายทอดจากบิตก่อนหน้า (propagate) ถ้ามีตัวใดตัวหนึ่งเป็น 1
- กำหนดให้
  - Generate signal = $a_i.b_i$
  - Propagate signal = $a_i + b_i$
- ดังนั้น $C_{i+1} = G_i + P_i . C_i$

Generate and Propagate

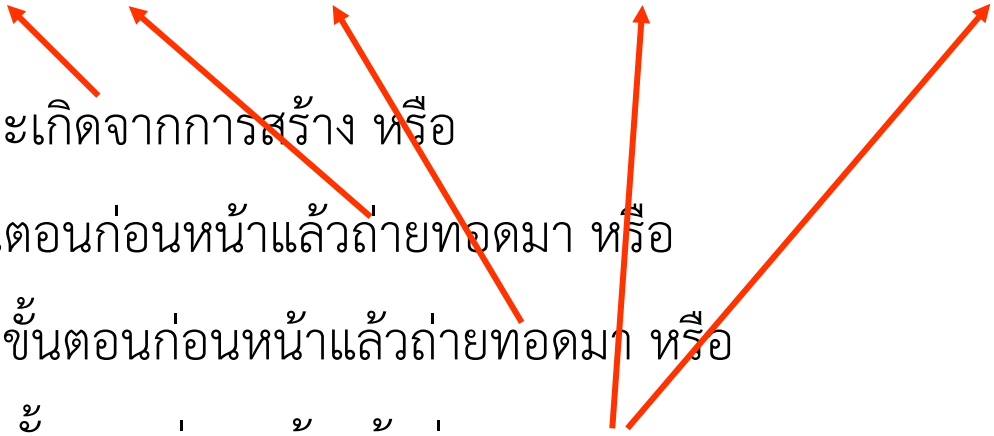
$$c1 = g0 + p0.c0$$

$$c2 = g1 + p1.c1$$

$$= g1 + p1.g0 + p1.p0.c0$$

$$c3 = g2 + p2.g1 + p2.p1.g0 + p2.p1.p0.c0$$

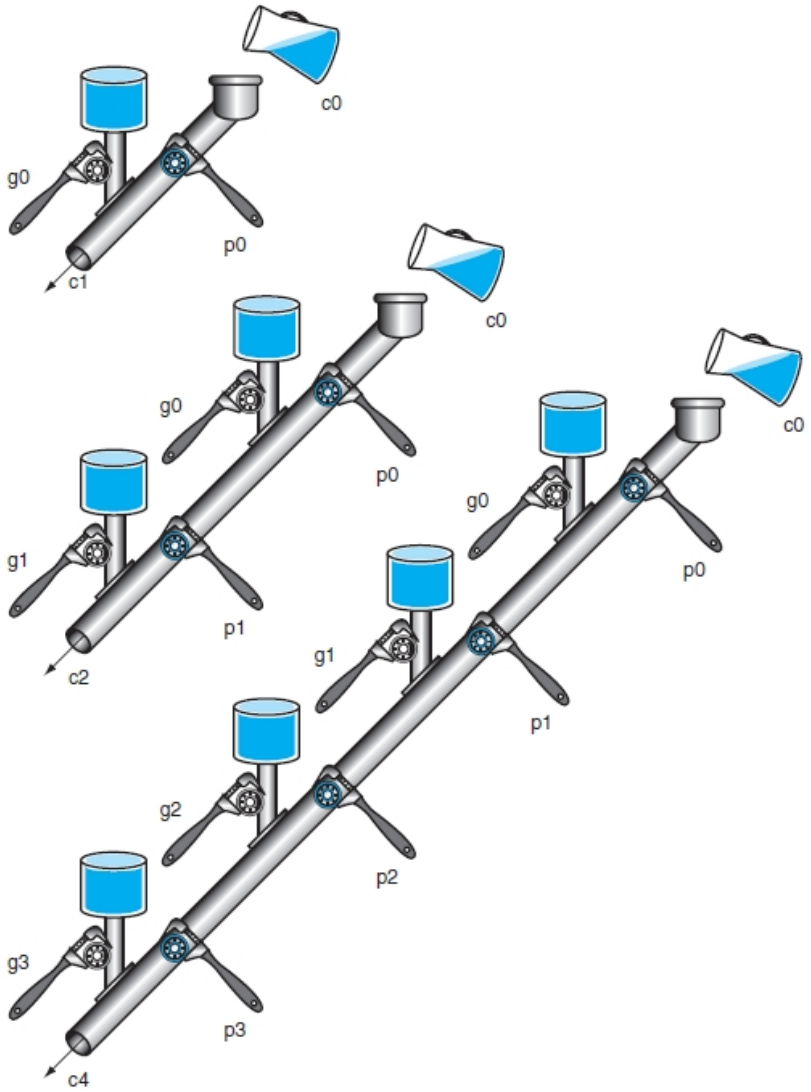
$$c4 = g3 + p3.g2 + p3.p2.g1 + p3.p2.p1.g0 + p3.p2.p1.p0.c0$$



- Carry อาจจะเกิดจากการสร้าง หรือ
- สร้างจากขั้นตอนก่อนหน้าแล้วถ่ายทอดมา หรือ
- สร้างจาก 2 ขั้นตอนก่อนหน้าแล้วถ่ายทอดมา หรือ
- สร้างจาก N ขั้นตอนก่อนหน้าแล้วถ่ายทอดมา

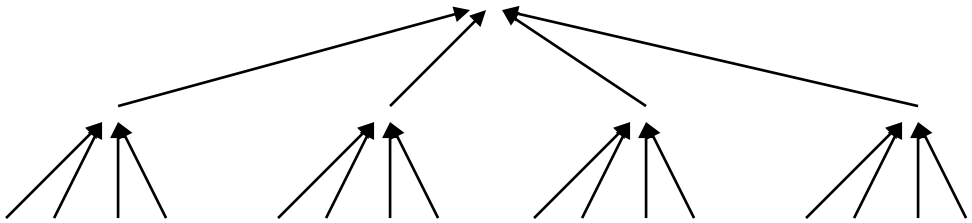
Generate and Propagate

- สามารถวาดเป็นรูปได้ดังนี้
  - c1 เกิดจาก c0 ถ่ายทอดมา หรือสร้างจาก g0 โดยตรง
  - c2 เกิดจาก c0 หรือ g0 แล้วถ่ายทอดมา หรือสร้างจาก g1 โดยตรง
- จะเห็นว่า
  - ถ้าเกิดจาก g ก็คือ a กับ b ที่ bit นั้นโดยตรง
  - แต่ถ้ามาจากขั้นตอนก่อนหน้า จะ propagate ได้ จะต้องมี a หรือ b เป็น 1 อย่างน้อย 1 ตัว



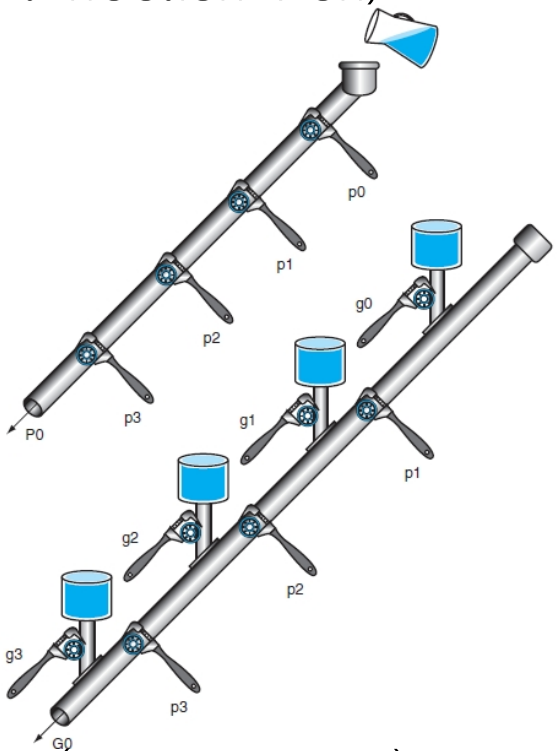
Divide and Conquer

- สมการก่อนหน้า (s.29) ยากที่จะสร้างเป็น logic function ระดับ 32 บิต เนื่องจากที่ c31 สมการจะยาวมาก และ logic gate ที่ใช้ก็มีจำนวนมากตามไปด้วย เช่น c31 = ... + c0[p0..p30]
- ดังนั้น เราจะใช้หลัก divide and conquer โดยแบ่งออกเป็นกลุ่มละ 4 บิต ภายใน 4 บิตก็จะมีการ propagate มากที่สุด ตามสมการ c4 เท่านั้น
- จากนั้นจะมีการสร้าง group-generate และ group-propagate ที่เป็นตัวแทนของกลุ่มขึ้นมาแทน



P and G for 4-bit Blocks

- เราจะกำหนด P (เรียกเป็น super-propagate คือเป็น propagate ระดับบล็อก 4 บิต) และ G (เรียกเป็น super-generate โดยจะเป็น generate ระดับบล็อก 4 บิต)
- ดังนั้น
  - P0 = p0.p1.p2.p3
  - G0 = g3 + g2.p3 + g1.p2.p3 + g0.p1.p2.p3
- Carry สำหรับ 4 บล็อกแรกสามารถเขียนได้ดังนี้
  - C1 = G0 + P0.c0
  - C2 = G1 + P1.G0 + P1.P0.c0
  - C3 = G2 + (P2.G1) + (P2.P1.G0) + (P2.P1.P0.c0)
  - C4 = G3 + (P3.G2) + (P3.P2.G1) + (P3.P2.P1.G0) + (P3.P2.P1.P0.c0)
- การดำเนินการเช่นนี้ จะทำให้แต่ละ gate จะมี Input น้อยลง



Example

ADD					
	a	0001	1010	0011	0011
	b	1110	0101	1110	1011
	g	0000	0000	0010	0011
	p	1111	1111	1111	1011
	P	1	1	1	0
	G	0	0	1	0

C4 = 1

Generate signal = ai.bi

Propagate signal = ai + bi

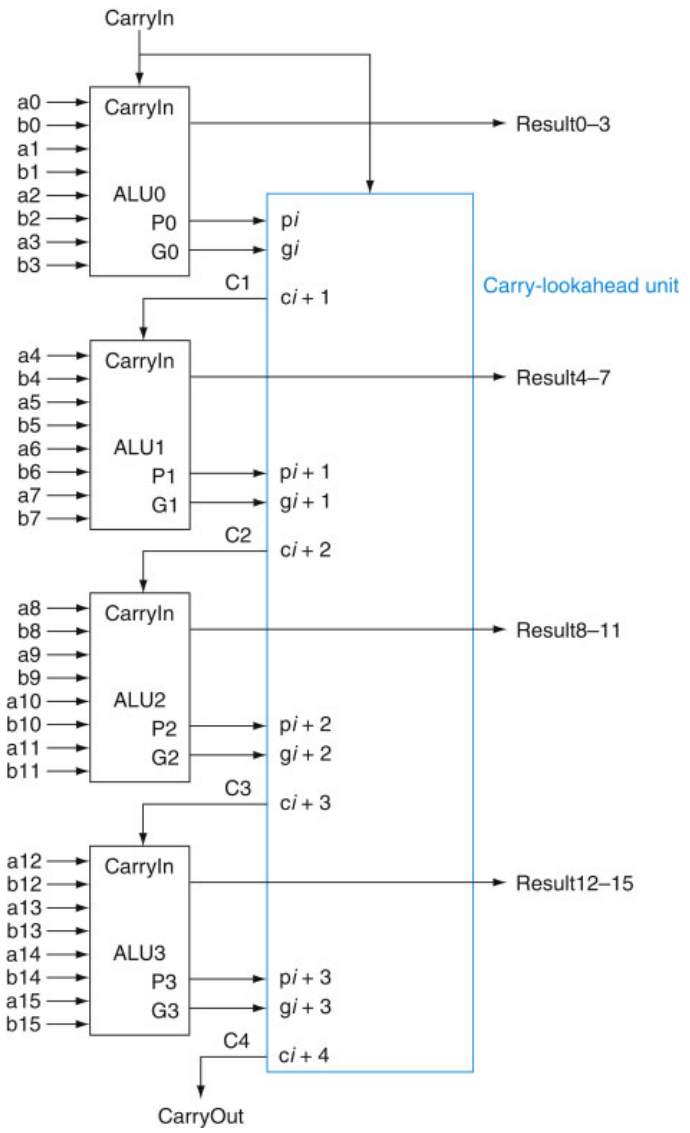
P0 = p0.p1.p2.p3

G0 = g3 + g2.p3 + g1.p2.p3 + g0.p1.p2.p3

C4 = G3 + (P3.G2) + (P3.P2.G1) + (P3.P2.P1.G0) + (P3.P2.P1.P0.c0)

Carry Look-Ahead Adder

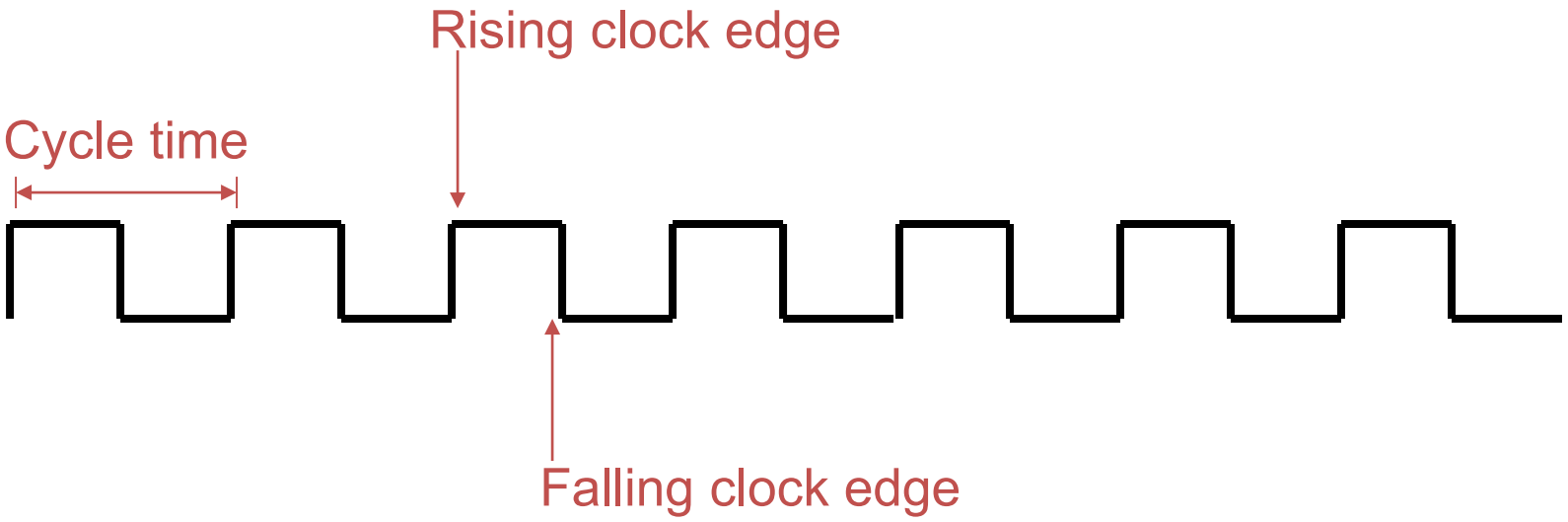
- 16 bit Ripple-carry จะมี gate delay เท่ากับ 32 step
- 4-bit Carry Look-Ahead Adder มี gate delay เท่ากับ ?



Clocks

- ภายในไมโครโพรเซสเซอร์ จะมีวงจรจำนวนมาก ที่ทำงานไปพร้อมๆ กัน ถ้าแต่ละวงจร x รับ Input ที่เวลา TI_x และใช้เวลา TE_x ในการทำงานในลอจิกนั้น จากนั้นจึงได้ Output ที่เวลา To_x จะทำอย่างไรให้วงจรแต่ละวงจรสามารถทำงานร่วมกันได้
- หลักคิดที่ผู้ออกแบบวงจรส่วนใหญ่ใช้ คือ ให้ทุกวงจรใช้สัญญาณนาฬิการ่วมกัน โดยสัญญาณนาฬิกามีหน้าที่บอกทุกๆ วงจร ว่าเมื่อใดจึงรับ Input เมื่อใดจึง Execute และเมื่อใดจึงให้ Output

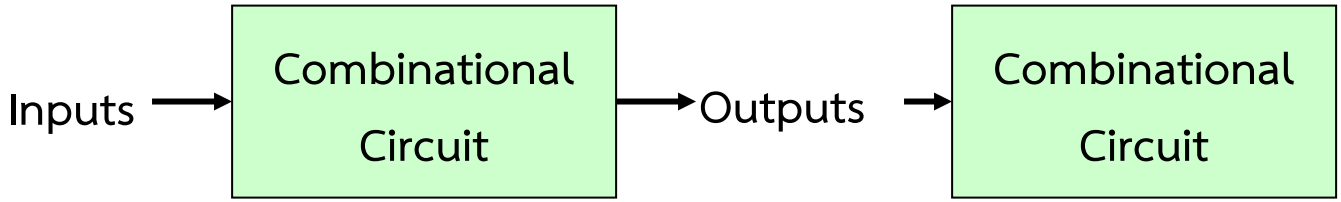
Clock Terminology



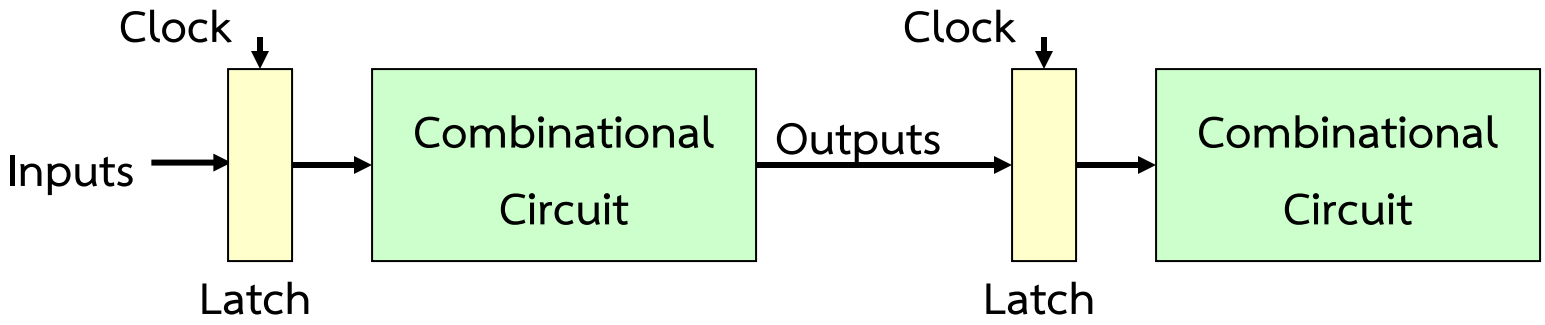
$$4\ \text{GHz} = \text{clock speed} = \frac{1}{\text{cycle time}} = \frac{1}{250\ \text{ps}}.$$

Sequential Circuits

- ที่ผ่านมากล่าวถึงเฉพาะวงจรแบบ Combination ซึ่ง Output จะขึ้นกับ Input โดยเมื่อ Input เปลี่ยนในเวลาไม่นาน Output ก็จะเปลี่ยนตาม (time = logic delay thru circuit) ซึ่งเราไม่สามารถควบคุมเวลาที่แน่นอนได้

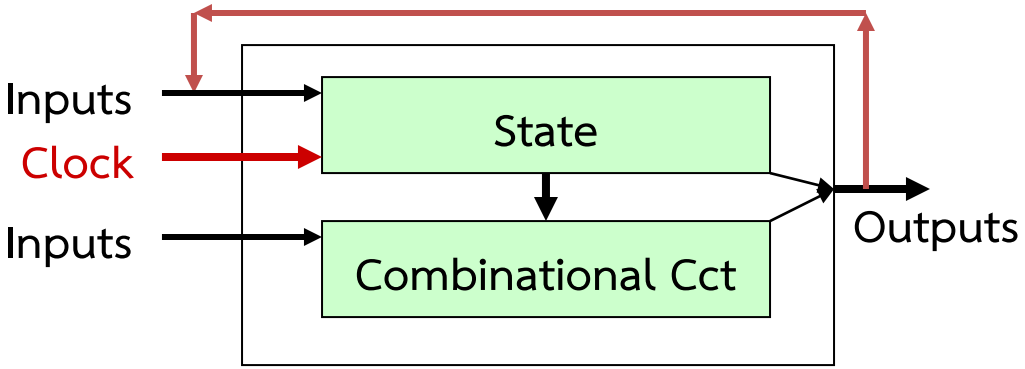


- หากเราต้องการใช้ clock ให้ทำงานคล้ายกับสัญญาณ start/stop เราจะเพิ่ม Latch เข้าไป โดย Latch จะเป็น Storage ที่เก็บสถานะ ทำให้สามารถเก็บข้อมูลสถานะของ Input และ Output เอาไว้ได้ ตาม clock ที่กำหนด



Sequential Circuits

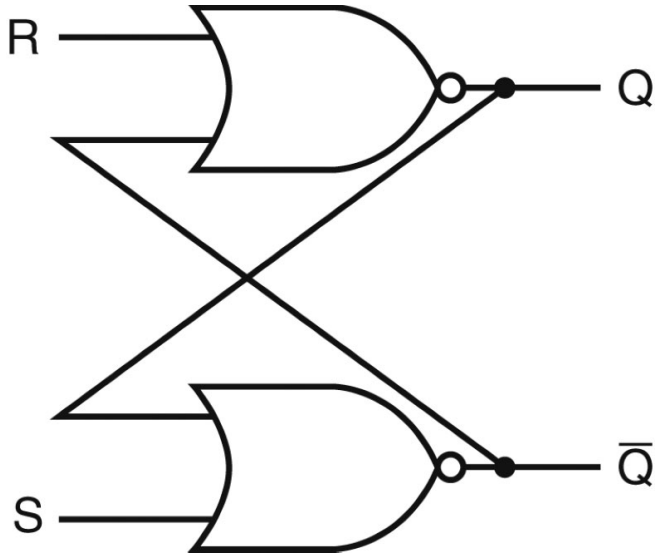
- วงจร Sequential ประกอบด้วย วงจร Combination และ Storage
- เมื่อได้รับ clock ขอบขาขึ้นจะทำให้ storage เก็บ state สำหรับ Input เอาไว้



- โดยค่าจะไม่เปลี่ยนตลอดการทำงาน (จนกว่าจะได้รับ clock ขอบขาขึ้นอันใหม่)
- ดังนั้นวงจร Sequential จะทำงานตามจังหวะ (state) ที่กำหนดให้ รับ Input และสร้าง Output
- ในบางกรณี Output อาจจะถูกนำมาป้อนกลับเข้ามาในวงจรอีกก็ได้

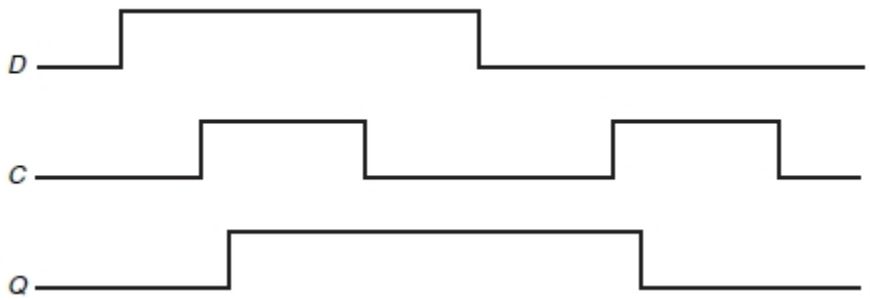
Designing a Latch

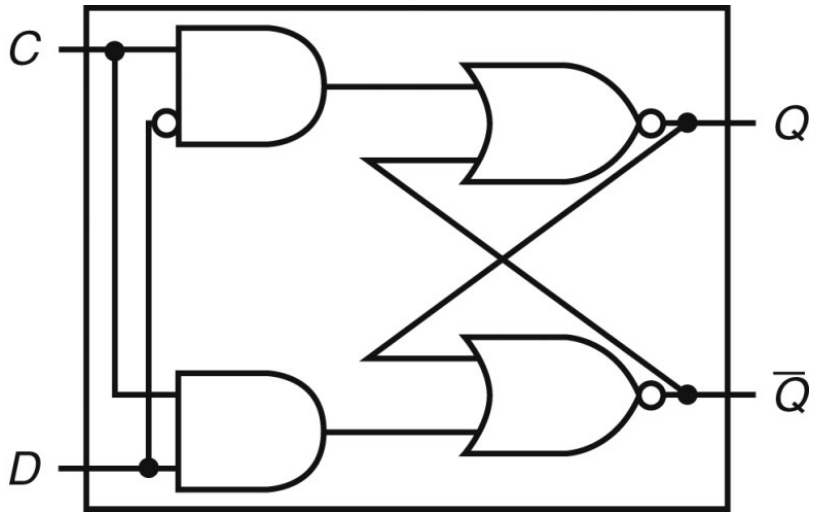
- ในการสร้าง Latch แบบที่ง่ายที่สุด คือ S-R Latch (Set / Reset Latch)
  - เมื่อ Set เป็น High ก็จะเก็บ 1
  - เมื่อ Reset เป็น High ก็จะเก็บ 0
  - ถ้า S และ R เป็น 0 ทั้งคู่ จะไม่เปลี่ยนแปลง
  - Note : เป็น High ทั้งคู่ไม่ได้



D Latch

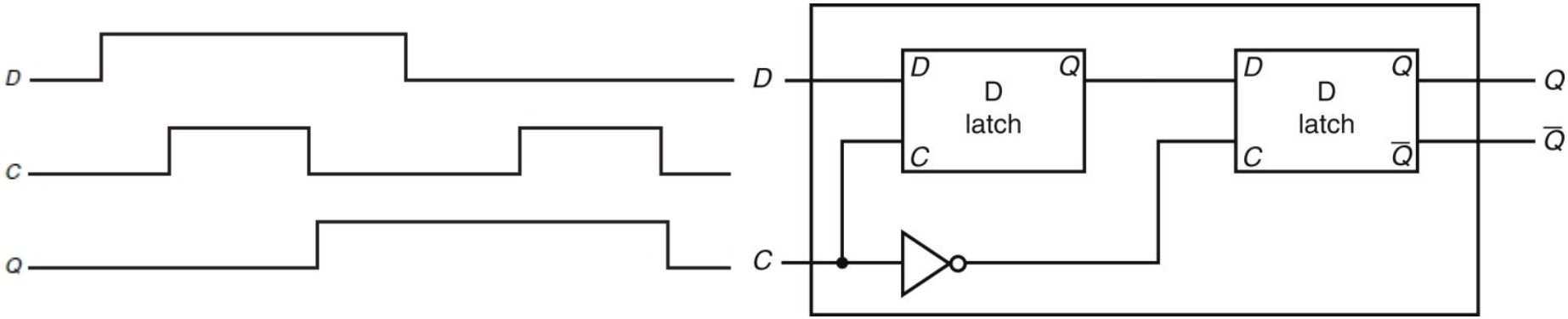
- วงจร S-R Latch ในรูปที่แล้ว ไม่สามารถทำงานร่วมกับ clock ได้
- จึงปรับปรุงเป็น D Latch โดยมี Input คือ D (Data) และจะเก็บข้อมูลเมื่อได้รับสัญญาณ C (clock) เมื่อ C เป็น Low จะไม่มีการเปลี่ยน state





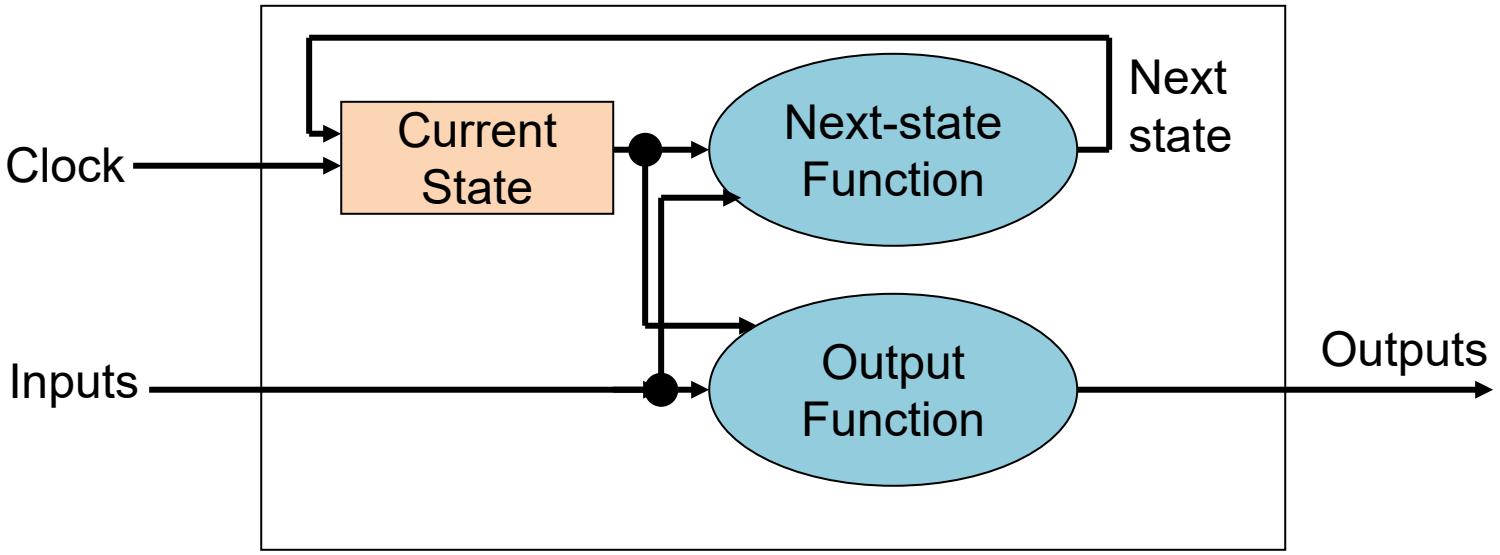
D Flip Flop

- คำนิยาม
  - Latch : Output สามารถเปลี่ยนแปลง เมื่อ clock เป็น High (Level)
  - Flip Flop : Output สามารถเปลี่ยนแปลง ที่ขอบของ clock (Edge)
- D Flip Flop จะใช้ D Latch 2 ตัว เพื่อให้ทำงานที่ขอบขาลง



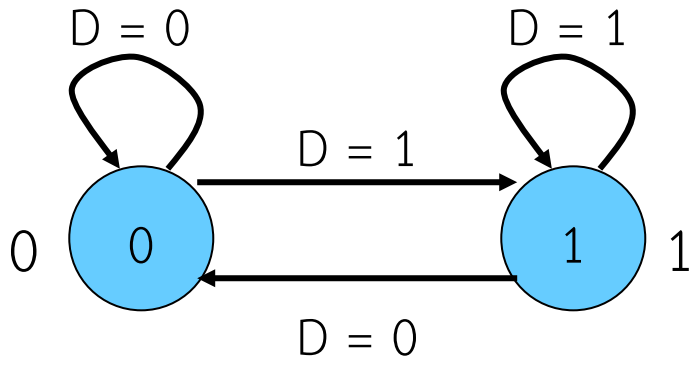
Finite State Machine

- เป็นวงจร Sequential ที่ทำงานตาม State Diagram (เป็นอีกรูปแบบหนึ่งของ Truth Table) จึงเรียกว่า Finite State Machine
- การเปลี่ยน State จะเกิดขึ้นจาก Input โดยมี clock เป็นตัวกำกับ



State Diagram

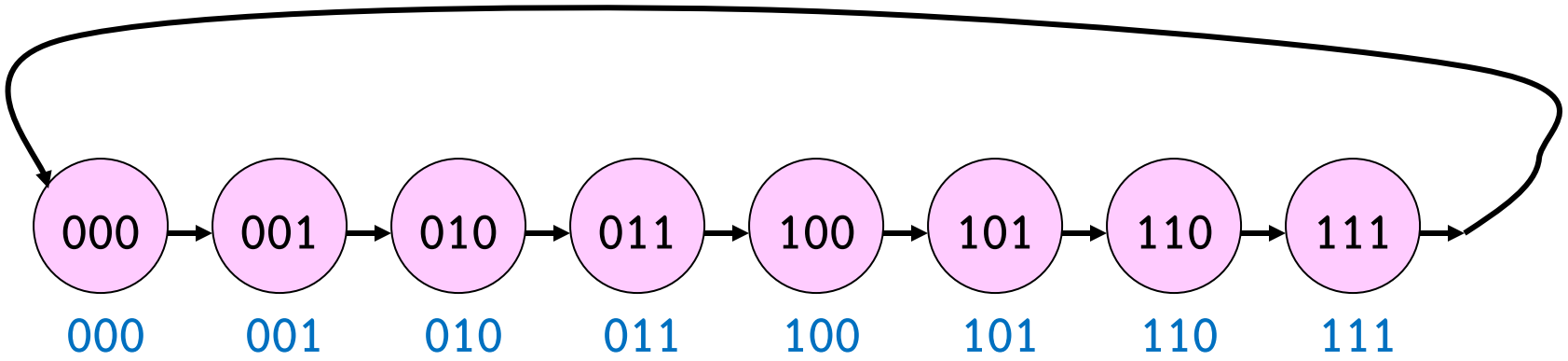
- แต่ละ state แสดงโดยใช้วงกลม ภายในวงกลมจะเขียนค่าของ state ซึ่งจะเป็นค่าของ output ด้วย
- เส้นโค้งแสดงการเปลี่ยนไปยัง state อื่น (หรือกลับมาที่ state เดิม) เมื่อได้รับ input ค่าต่างๆ



This is a state diagram for ___?

3-bit Counter

- ถ้าจะสร้างวงจรที่เก็บค่าและเพิ่มค่าขึ้นเรื่อยๆ ทุกๆ ขอบขาลงของ clock และเมื่อถึงค่าสูงสุด (ในที่นี้คือ 7) ให้กลับไปเป็น 0 ใหม่ จะเขียน state diagram อย่างไร?
  - จะมีกี่ state
  - จะต้องมีกี่ inputs



Traffic Light Controller

- กำหนดให้สี่แยกแห่งหนึ่ง มีไฟจราจรเพียงเขียวและแดง โดยมีแต่รถวิ่งตรงไป ไม่มีเลี้ยวซ้ายเลี้ยวขวา เมื่อไฟด้าน North-South เขียว รถด้าน North-South จะไปได้ และเมื่อไฟด้าน East-West เขียว รถด้าน East-West จึงจะไปได้ ห้ามเป็นแดงทั้งคู่และเขียวทั้งคู่
- บนถนนจะติดตั้ง Sensor ตรวจจับว่ามีรถอยู่บนถนนหรือไม่ ทั้ง 4 ทิศทาง
- กำหนดให้ไฟเปลี่ยนทุกๆ 30 วินาที โดยไฟจะเปลี่ยนเมื่อมีรถมารอที่ด้านนั้นเท่านั้น
  - State diagram เป็นอย่างไร มีกี่ state
  - มีกี่ input
  - มีกี่ output

Traffic Light Controller

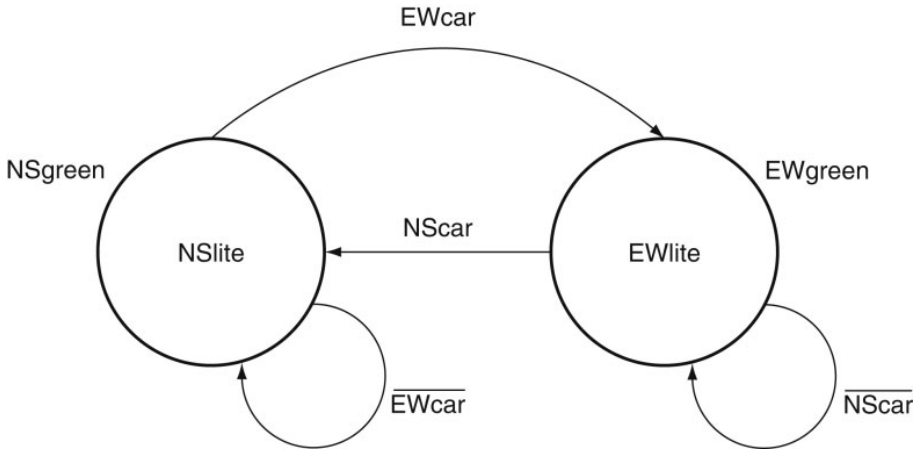
- State Transition Table

CurrState	InputEW	InputNS	NextState=Output
N	0	0	N
N	0	1	N
N	1	0	E
N	1	1	E
E	0	0	E
E	0	1	N
E	1	0	E
E	1	1	N

Traffic Light Controller

- State Diagram

CurrState	InputEW	InputNS	NextState =Output
N	0	0	N
N	0	1	N
N	1	0	E
N	1	1	E
E	0	0	E
E	0	1	N
E	1	0	E
E	1	1	N



For your attention