Digital Building Blocks

ARITHMETIC CIRCUITS

الدوائر الحسابية هي اللبنات الاساسية لأجهزة الكمبيوتر.

الكمبيوترات و المنطق الرقمي تؤدي الى عديد من الوظائف الحاسبية منها :

الجمع/addition

الطرح/subtraction

المقارنات/comparisons

التحويل/shifts

الضرب/multiplication

القسمة/division

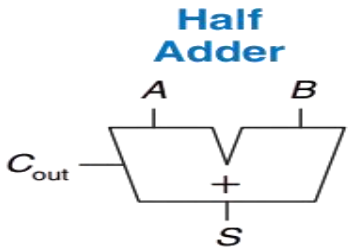
Addition

تعتبر عملية الاضافة من اكثر العمليات شيوعا في الانظمة الرقمية.

سننظر اولا في كيفية اضافة رقمين ثنائيين كل منها 1 بت.

الـ Adders بوضح بين السرعة والتعقيد.

Half Adder



A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$
$$C_{out} = AB$$

نبدأ في بناء 1-bit half adder كما هو موضح في الصورة يحتوي half adder على مدخلين A & B ومخرجين Cout & S .

الـ S هو مجموع A & B اذا كان الـ A و B الاثنان 1 يعني S = 2 وهو لا يمكن تمثيله رقم واحد كـ ثنائي.

بدلا من ذالك تتم الاشارة اليه بعامود الثاني Cout (carry out).

وايضا يمكن بناء الـ half adder على XOR & AND Gate .

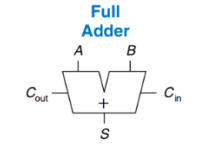
في الـ multi-bit adder تتم اضافة Cout او نقلها الى MSB .

على سبيل المثال في شكل 5.2 البته المحموله :

```
  1
 0001
+0101
 0110
```

البته التي باللون الازرق هي عند الـ Cout .

Full Adder



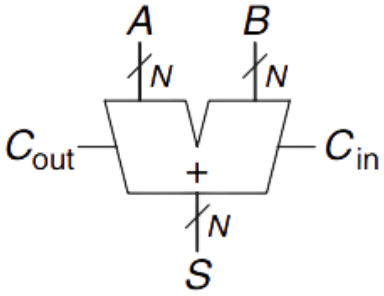
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

الفرق يوجد لديه Cin .

Carry Propagate Adder

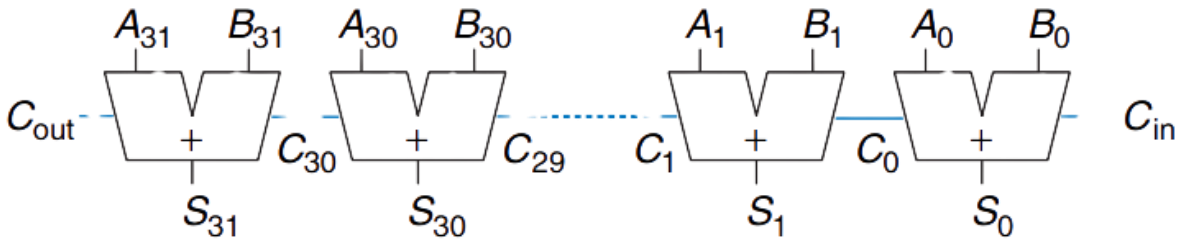
الـ N-bit adder يجمع اثنان N-bit inputs الي هم A & B والـ Carry في Cin لأنتاج نتيجة في مخرجات الـ S و Cout .



C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

هاذي صورة الـ CPA هو مثل الـ Full Adder لكن الفرق ان A , B & S عبارة عن Busses وليست single bits .

Ripple-Carry Adder



جمع عددين كل عدد عبارة عن n-bit ويمكن بناؤه باستخدام full adders مع كل full adder في carry out ومتصل في Cin C Input .

اسهل طريقة هي عشان تبني Ripple Carry Adder هي انك تجمع اكثر من full adder لكن عندها عيب واحد هو انو لما يكون في N كثير بصير ابطئ.

الـ Cin بكون في مرحلة واحدة على اليسار والـ Cout على اليمين ويعمل هذا الـ Carry Adder كـ 32-bit الـ S31 يعتمد على C30 و C30 تعتمد على C29 و C29 تعتمد على C28 حتى وصولك الى Cin .

Carry-Lookahead Adder

اسمها الجامع مبدئ انظر الى الامام.

وعيبها الوحيد ان كنت تريد ان تاخذ اي جمع في اي Block لا يمكنك الا اذا انتهى جميع الـ Blocks التي خلفها مثال

تريد S3 ؟

يجب ان ينتهي B3 , A3 & C3 ولينتهي C3 يجب ان ينتهي B2 , A2 & C2 الخ.

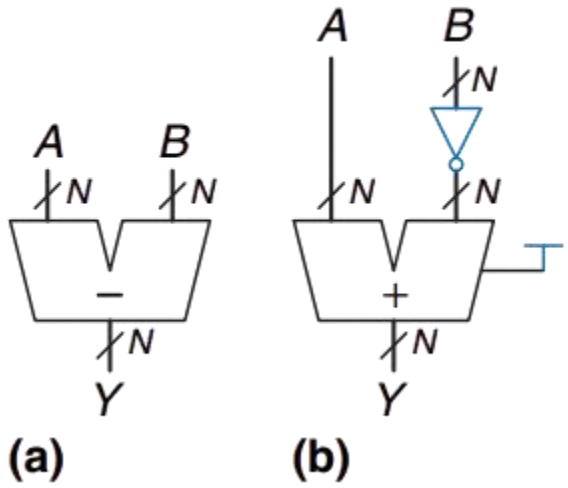
سبب الرئيسي لبطئ Ripple Carry Adder هو الـ Carry Signals تنتقل الى جميع البتات في Adder .

Carry lookahead adder نوع اخر من carry adder وتحل هاذي المشكلة عن طريق تقسيم الـ Adder واضافة دوائر لتحديد تنفيذ الكتلة بسرعة بمجرد معرفة عملية carry .

على سبيل المثال يمكن تقسيم الـ Adder 32-bit الى 4-byte وتكون 8 blocks .

الـ CLA تستخدم generate (G) and propagate (P) اشارات والتي تصف كيفية تحديد العمود او Block Carry Out .

Subtraction



الـ Adders بتقدر تجمع ارقام سالبة وموجبة كيف عن طريق الـ 2's comp الي شرحناها عملية الطرح بنفس السهولة تقريبا مثل ما شرحناها فقط اقلب علامة الرقم الثاني وبعدها اضافة 1 بعدها اجمع .

Comparators

او تسمى المقارنة اذا كان رقمان الثنائيان متساويان او اذا كان احدهما اكبر او اقل من الاخر الخ هاي الدائرة بتحل الموضوع.

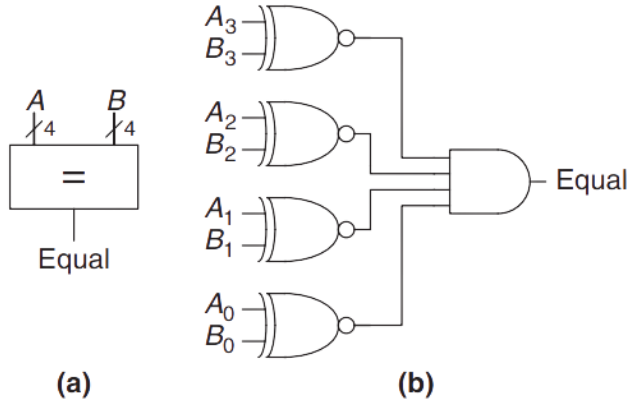
تاخذ N-bit Binary Number و مدخلين A & B .

هناك نوعان شائعان في المقارنات .

الـ equality comparator : تنتج مخرجات واحد تشير اذا كان يوجد مساواه مثال A تساوي B هكذا : (A == B) .

الـ magnitude comparator : تنتج مخرجا واحد او اكثر يشير يشير الى الارقام النسبية لـ A & B .

الـ equality comparator هي ابسط قطعة في الـ Hardware .



هاي صورة بتوضح لك الـ equality comparator لـ 4-bit يقوم اولا بتحديد اذا كانت متساوية من خلال الـ XNOR Gate يجري عمليته الحسابيه عليها التي تدل ان كانت اصفار 0 0 تساوي 1 وان كانت 1 1 تساوي 1 .

اذا كانت الارقام متساويه جميع الاعمدة متساوية .

عادة تتم عملية المقارنة عن طريق A – B والنظر الى الاشارة MSB مثل الصورة :

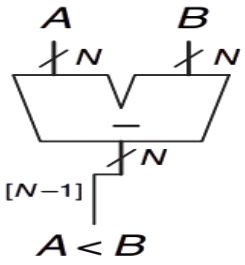


Figure 5.12 ***N*-bit magnitude comparator**

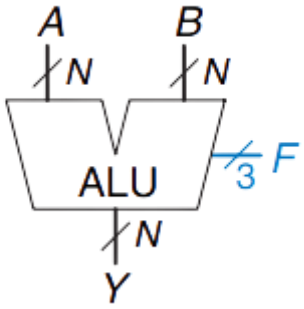
اذا كانت النتيجة سلبية اي ان الاشارة 1 بعدها A اقل من B .

حالة غير او A اكبر او تساوي B .

Arithmetic/Logical Unit (ALU)

تجمع بين مجموعة من التعليمات الرياضية والمنطقية في single unit .

على سبيل المثال قد تقوم وحدة ALU باجراء عملية جمع وطرح ومقارنة وعمليات OR & AND الـ ALU يشكل كـ قلب في معظم الكمبيوترات.



الصورة هاي توضح لك الـ symbol الخاص في ALU الي عندها N-bit كـ مدخلات و N-bit كـ مخرجات ويتلقى اشارة F هي Control Signal التي تحدد الوظيفة التي سيتم تنفيذها.

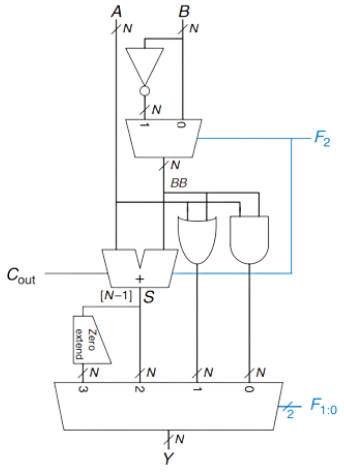
الاشارات الزرقاء هنا لونها ازرق ليتم تميزها عن المدخلات والمخرجات الخ .

هذا الجدول الوظائف النموذجية التي يمكن ان تقوم بها وحدة الـ ALU :

$F_{2\text{-}0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT

يتم استخدمها الـ SLT لمقارنة.

الصورة هاذي توضح تنفيذ وحدة ALU :



الصورة توضح يوجد N-bit Adder و بوابة AND و OR كل وحده 2 Inputs .

وايضا يحتوي على MUX و عاكس العاكس ليقلب B ويوجد اشارة F2 عندما يتم تاكيدها.

الـ MUX هو 4:1 ويعتمد على الاشارة F1:0 .

وبشكل اكثر تحديدا يعمل الـ ALU على A and BB اما `B & B يعتمد على F2 .

عندما تكون F1:0 == 00 الـ MUX سيختار A & BB .

بمعنى سيكون A & BB الحساب A AND BB .

اما اذا كان F1:0 == 01 سيتم اخذ A و BB وستكون العمليه الحسابيه OR يعني A OR BB .

واذا كانت F1:0 == 10 ستقوم الـ ALU بعمل عملية جمع او طرح .

لاحظ الـ F2 Control هو كمان عند الـ Adder .

اذا كانت الـ F2 = 0 فان راح يكون عملية جمع A + B واذا كانت F2 = 1 فان ستحسب A – B لكن تذكر العمليات هاذي تعمل على 2's comp هنا .

اذا كان الـ F2:0 == 111 تقوم وحدة الـ ALU بتنفيذ set if less than (SLT) عندما A < B, Y= 1 .

بمعنى اخر يتم تعيم Y = 1 اذا كانت A < B .

يتم تنفيذ الـ SLT عن طريق S = A – B اذا كانت S سالبة A تكون اصغر من B .

الـ zero extend unit تنتج هاذي الوحدة مخرجات N-bit عن تسلسل مدخلاتها ذات 1 بت مع 0 من جهة MSB .

الـ Sign bit N – 1 من الـ S هو المدخل الى Zero Extend unit .

Shifters and Rotators

الـ Shifters and rotators تقوم بتحريك البتات وتقوم بضرب او تقسم على power 2 x^2

من الاسامي يتضح لك ماذا يقوم بعمله الـ shifters يقوم بازاحة الـ bit الى اليسار او اليمين بعدد محدد.

هناك عدة تعمل مثل هكذا لكن هذا الشائع :

Logical shifter : ينقل البت الى اليسار او على اليمين

shifts the number to the left (LSL)

or right (LSR)

ويملأ الاماكن الفارغة بالصفر.

على سبيل المثال :

Ex: 11001

LSR 2 : 00110

LSL 2 : 00100

Arithmetic shift left (ASL) is the same as logical shift left (LSL).

Arithmetic shifter : نفس الـ logical shifter ولا في فرق ولكن عند التحركات اليمنى يتم ملء البتات الـ MSB بنسخة القديمة لـ MSB .

مثال :

ASR 2 : 11001

ستصبح : 11110

ASL 2 : 11001

ستصبح : 00100

Rotator : يقوم بتدوير الارقام في دائرة بحيث يتم ملء النقاط الفارغة بالبتات التي تم ازاحتها من الطرف الاخر.

مثال :

11001

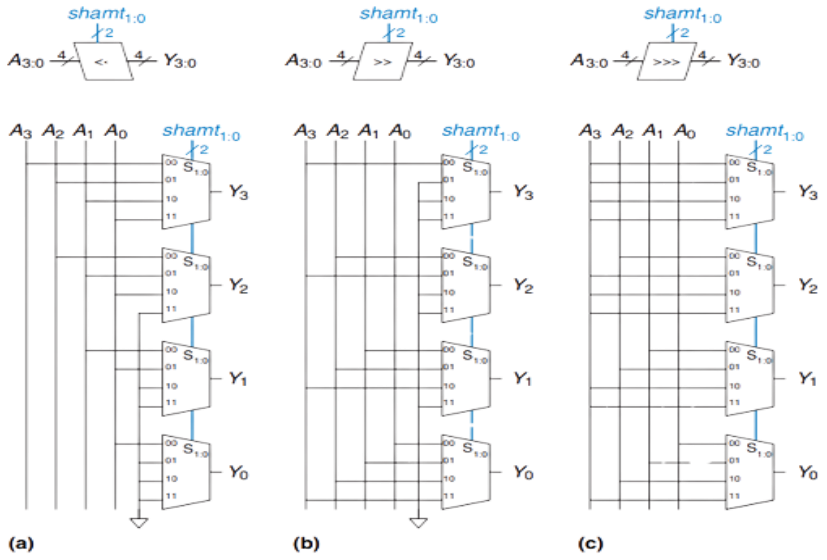
ROR 2 : 01110

ROL 2 : 00111

في الدوران اليساري يتم اعادة البتات من طرف اليسار الى طرف اليمين والعكس عند دوران اليمين.

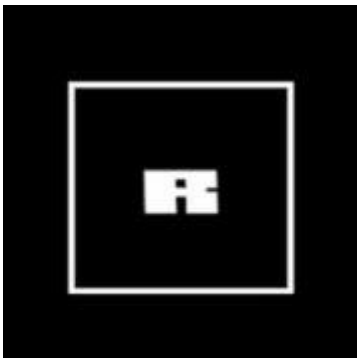
يمكن بناء N-bit Shifter من الـ MUX N:1 مقدار الازاحة من 0 الى N – 1 اعتمادا على Log2(n-bit) خطوط.

الصورة هاي بتبين لك 4-bit shifter .



الـ operators هذول <<< << >> تشير الى shift left و shift right و arithmetic shift right الخ.

rETKit



https://twitter.com/dr_retkit

https://www.youtube.com/@retkit1823/videos