الوحدة الأولى: تمثيل البيانات في الأنظمة الرقمية

محتويات الوحدة

تمهيد

مرحباً بك عزيزي الدارس في الوحدة الأولي من مقرر "أساسيات التصميم المنطقي". توضح هذه الوحدة الطريقة التي يتم بها تمثيل مختلف أنواع البيانات داخل جهاز الحاسوب، أو داخل الأنظمة الرقمية (Digital Systems) بصفة عامة. حيث يتم تعريف الأنواع الأساسية من البيانات و تشمل الأعداد الصحيحة و الأعداد الحقيقية و الرموز، وتوضيح طريقة تمثيل كل نوع منها، ومدى القيم التي يقبلها كل نوع، والاستخدامات المناسبة لكل نوع. كما تتناول الوحدة بعض أنواع الشفرات القياسية المستخدمة في الحاسوب لتمثيل البيانات.

أهداف الوحدة

عزيزي الدارس، بعد دراسة هذه الوحدة ينبغي أن تكون قادراً على:

- التعرف علي الأنواع الأساسية من البيانات و طريقة تمثيلها.
- استنتاج مدى القيم الذي يمكن استخدامه مع كل نوع.
- التعرف على أنواع الشفرات القياسية المستخدمة في الحاسوب.

1- الأعداد الصحيحة (Integers)

حتى يتمكن أي نظام رقمي مثل الحاسوب من التعامل مع أي نوع من أنواع البيانات فان تلك البيانات يجب أن تكون ممثلة فى الصورة الثنائية (Binary)، أي فى صورة مجموعة من الـــ 0's والـــ 1's. حيث يتم تمثيل القيمة المنطقية 0 بمستوى جهد معين داخل الدوائر الإلكترونية للنظام الرقمي، و يتم تمثيل القيمة المنطقية 1 بمستوى جهد آخر. مثلاً الجهد 5+ Volt يمثل القيمة المنطقية 1 والجهد 0 Volt يمثل القيمة المنطقية 0.

أنواع الأعداد الصحيحة

تنقسم الأعداد الصحيحة إلى عدة أنواع حسب المساحة المستخدمة في تخزين العدد:

(1) عدد صحيح قصير (Short Integer) و طوله 1 Byte = 8 bits

(2) عدد صحيح (Integer) و طوله 2 Byte = 16 bits

(3) عدد صحيح طويل (Long Integer) و طوله 4 Byte = 32 bits

من ناحية أخرى تنقسم الأعداد الصحيحة حسب طبيعة الأعداد التي يتم تخزينها فيها إلي نوعين وهما:

(1) الأعداد الصحيح بدون إشارة (Unsigned Integers) وفيها يتم تخزين الأعداد الموجبة فقط.

(2) الأعداد الصحيحة بإشارة (Signed Integers) وفيها يتم تخزين الاعداد الموجبة والسالبة

1-1 الأعداد الصحيحة بدون إشارة Unsigned Numbers

لتمثيل العدد الصحيح 135 مثلاً، يجب تحويله أولاً من الصورة العشرية (Decimal) إلى الصورة الثنائية (Binary). و يتم ذلك، كما تعلم عزيزى الدارس، بالقسمة المتكررة على 2 والاحتفاظ بباقي القسمة (كما تعلمت في مقرر "مبادئ علوم الحاسوب" الذي قمت بدراسته). فالعدد الصحيح العشري 135 يكافئ العدد الثنائي 10000111، ويكتب ذلك رياضياً في الصورة:

$$(135)_{10} = (10000111)_2$$

و للتحقق من ذلك يمكن أن نقوم بالعملية العكسية، أي تحويل العدد الثنائي 10000111 إلى الصورة العشرية.

تدريب 1:

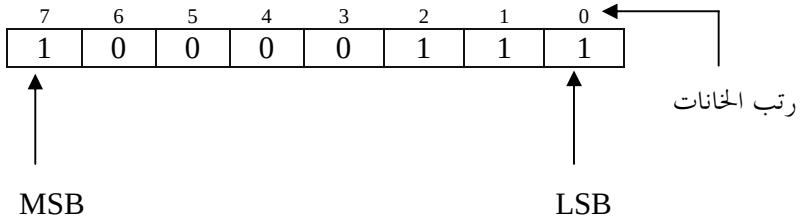
حول الأعداد العشرية التالية إلى الصورة الثنائية:			
55 (1)	200 (2)	300 (3)	600 (4)

تدريب 2:

حول الأعداد الثنائية التالية إلى الصورة العشرية:			
10111011 (1)	10000000 (2)	11100111 (3)	11111111 (4)

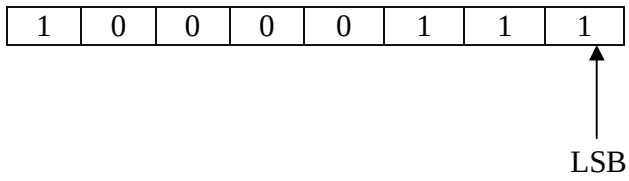
تسمى الخانة الواقعة في أقصى اليمين في العدد الثنائي بالخانة الدنيا (Least Significant Bit)، أو LSB اختصاراً، و ذلك لأنها الخانة الأقل وزناً. في حين تسمى الخانة الواقعة في أقصى اليسار بالخانة العليا (Most Significant Bit)، أو MSB اختصاراً، و ذلك لأنها الخانة الأعلى وزناً.

تذكر أن وزن الخانة هو عبارة عن الأساس 2 مرفوع لأس يساوى رتبة الخانة، وأننا نحصل على رتب الخانات بترقيم الخانات ابتداء من الخانة التي تقع في أقصى اليمين، مبتدئين الترقيم بالقيمة صفر.

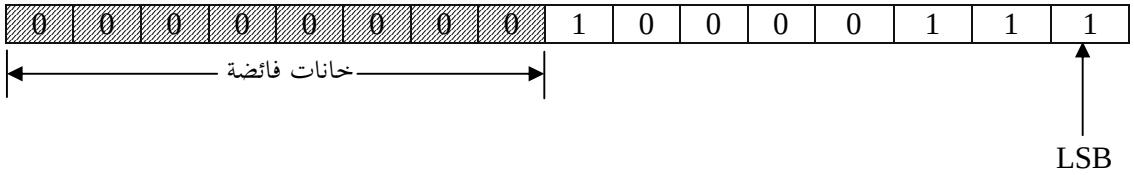


بعد تحويل العدد إلي الصورة الثنائية ننظر إلى المساحة المتاحة لتخزين العدد و نقوم بوضع الخانات بالترتيب فيها مبتدئين بالخانة الدنيا (LSB)، مع ملء أي خانات فائضة إلى اليسار بأصفار (0's).

مثلاً إذا كانت المساحة المتاحة 1 Byte = 8 bits فإن التخزين سيتم كالتالي:



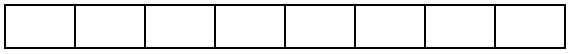
أما إذا كانت المساحة المتاحة 2 Bytes = 16 bits فإن التخزين سيتم كالتالي:



أي انه إذا كان طول العدد الثنائي أقل من المساحة المتاحة يتم محاذاته إلى اليمين ثم تملأ الخانات الزائدة إلى اليسار بأصفار (0's). تسمى هذه العملية **بالمحاذاة إلى اليمين مع الملء بأصفار (Right Justify- Zero Fill).**

يمكن حساب مدى القيم التي يمكن تخزينها في صورة عدد صحيح قصير (Short Integer) كالتالي:

المساحة المتاحة هي 1 Byte = 8 bits أي 8 خانات ثنائية



نحصل علي أصغر قيمة بملء جميع الخانات بـــ 0's

0	0	0	0	0	0	0	0	=	**0**

و نحصل علي أكبر قيمة بملء جميع الخانات بـــ 1's

1	1	1	1	1	1	1	1	=	**255**

و عليه فإن مدى القيم التي يمكن تمثيلها فى صورة عدد صحيح قصير (Short) هو $0 \sim 255$ أو $0 \sim (2^8 - 1)$
و بالمثل يمكن اثبات أن مدى القيم التي يمكن تمثيلها فى صورة عدد صحيح (Integer) هو $0 \sim (2^{16} - 1)$
وعموماً إذا كان عدد الخانات المتاحة هو N فإن المدى هو $0 \sim (2^N - 1)$

الجدول التالي يوضح أنواع الأعداد الصحيحة وطول كل منها ومدي القيم الذي يمكن تخزينه في كل نوع

نوع العدد الصحيح	طوله	مدى القيم
Short Integer	1 Byte = 8 bits	$0 \sim (2^8 - 1)$ $0 \sim 255$
Integer	2 Bytes = 16 bits	$0 \sim (2^{16} - 1)$ $0 \sim 65,535$
Long Integer	4 Bytes = 32 bits	$0 \sim (2^{32} - 1)$ $0 \sim 4,294,967,295$
————	N	$0 \sim (2^N - 1)$

تسمى الأعداد الصحيحة التي تعاملنا معها في ما سبق بالأعداد الصحيحة بدون إشارة (Unsigned Integers)

تدريب 3:

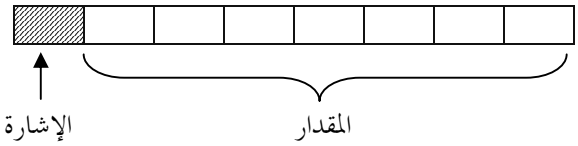
طالما أن مدى القيم التي يمكن تخزينها في الأعداد الصحيحة يزداد كلما إزداد طول العدد فلماذا تم استخدام أطوال مختلفة للأعداد (حيث استخدمت الأطوال 8 و 16 و 32 خانة)؟

1-2 الأعداد الصحيحة ذات الإشارة (Signed Integers)

تناولنا في الجزء السابق طريقة تمثيل الأعداد الصحيحة بدون إشارة (Unsigned Integers) والتي يتم تخزين قيم موجبة فقط بها. و بالتالي فإن أصغر قيمة يمكن تخزينها فيها هي الصفر.

والسؤال الآن هو كيف يتم تمثيل الأعداد السالبة في الحاسوب؟

لتمثيل الأعداد السالبة يتم حجز خانة bit لتمثيل إشارة العدد (Sign)، وعادة ما تكون هذه الخانة هي الخانة العليا MSB، و يتم تخزين مقدار العدد (Magnitude) في بقية الخانات



و عادة ما تستخدم القيمة 0 في الخانة العليا MSB لتمثيل الإشارة الموجبة، في حين تستخدم القيمة 1 لتمثيل الإشارة السالبة. فلمعرفة إشارة العدد ننظر إلى الخانة العليا MSB فإذا كان:

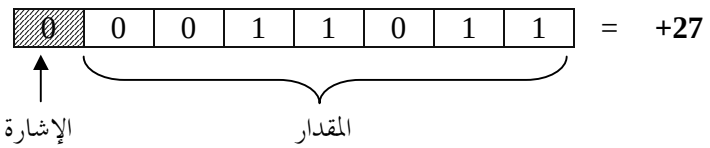
MSB = 0 فالعدد موجب

MSB = 1 فالعدد سالب

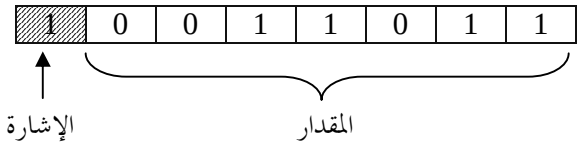
مثلاً إذا أردنا تمثيل القيمة 27+ في صورة عدد صحيح بإشارة في مساحة تبلغ 1 Byte = 8 bits فإننا نتجاهل إشارة القيمة مؤقتاً و نقوم بتحويل المقدار من الصورة العشرية إلى الصورة الثنائية

$$27 = (11011)_2$$

المساحة المتاحة تبلغ 8 خانات، نستبعد منها الخانة العليا MSB لتمثيل الإشارة، فيتبقى 7 خانات لتمثيل المقدار. يتم تخزين مقدار العدد الصحيح ذو الإشارة في المساحة المتاحة له بنفس طريقة تخزين الأعداد الصحيحة بدون إشارة (Unsigned Integers). و أخيراً نضع 0 في خانة الإشارة لأن القيمة موجبة.

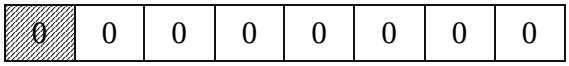


و تمثيل القيمة 27- يتم بنفس الطريقة و لكن مع وضع 1 في خانة الإشارة لأن القيمة سالبة.

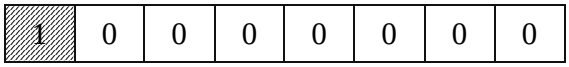


يسمى هذا الأسلوب في تمثيل الأعداد الصحيحة ذات الإشارة بطريقة **المقدار-الإشارة (Sign-Magnitude)**، حيث تم الفصل بصورة كاملة ما بين إشارة القيمة و مقدارها.

هذا الأسلوب فى تمثيل الأعداد الصحيحة ذات الإشارة به مشكلة خطيرة تتمثل فى أن القيمة صفر لها شكلين



صفر موجب



صفر سالب

و وجود شكلين للصفر يعتبر مشكلة لأن عملية فحص قيمة معينة لمعرفة ما إذا كانت مساوية للصفر أم لا هي من أكثر العمليات التي يتم إجراؤها داخل الأنظمة الرقمية، و وجود شكلين للصفر يعنى أن هذه العملية يجب إجراؤها مرتين، مما يقلل كثيراً من كفاءة النظام الرقمي.

حلاً لهذه المشكلة يستخدم أسلوب **المكمل الثاني (2's Complement)** لتمثيل الأعداد الصحيحة ذات الإشارة.

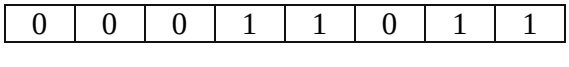
مثلاً إذا أردنا تمثيل القيمة 27+ في صورة عدد صحيح بإشارة في مساحة تبلغ 1 Byte = 8 bits فإننا نتجاهل إشارة القيمة مؤقتاً و نقوم بتحويل المقدار من الصورة العشرية إلى الصورة الثنائية

$$27 = (11011)_2$$

المساحة المتاحة تبلغ 8 خانات، لذلك نقوم بإكمال طول العدد الثنائي إلى 8 خانات و ذلك بإضافة أصفار (0's) إلى يسار العدد.

$$(11011)_2 = (00011011)_2$$

و أخيراً نقوم بوضع العدد الثنائي في المساحة المتاحة له



أما لتمثيل القيمة 27- فإننا نبدأ بنفس خطوات تمثيل القيمة 27+، حيث نتجاهل إشارة القيمة مؤقتاً و نقوم بتحويل المقدار من الصورة العشرية إلى الصورة الثنائية، ثم نقوم بإكمال طول العدد الثنائي إلى 8 خانات و ذلك بإضافة أصفار (0's) إلى يسار العدد.

$$27 = (11011)_2 = (00011011)_2$$

و بما أن القيمة المطلوب تمثيلها سالبة فإننا نحتاج إلى إيجاد المكمل الثاني (2's Complement) للعدد الثنائي الناتج، حيث أن المكمل الثاني لعدد ثنائي هنا يمثل القيمة السالبة للعدد.
إيجاد المكمل الثاني لعدد ثنائي يتم في خطوتين. الخطوة الأولى هي إيجاد المكمل الأول (1's Complement) و ذلك بعكس جميع خانات العدد الثنائي، أي تحويل أي 0 إلى 1 و تحويل أي 1 إلى 0. الخطوة الثانية هي إضافة 1 للمكمل الأول لنحصل على المكمل الثاني.

العدد	00011011
المكمل الأول	11100100
	+ 1
المكمل الثاني	11100101

أخيراً نقوم بوضع العدد الثنائي الناتج في المساحة المتاحة له.

1	1	1	0	0	1	0	1

= **-27**

لاحظ الآتي :

- الخانة العليا MSB هنا ما زالت تمثل إشارة العدد، فـ MSB=0 للقيمة الموجبة 27+ و MSB=1 للقيمة السالبة 27-.
- المكمل الثاني (2's Complement) لعدد ثنائي يمثل سالب ذلك العدد.
- لا يوجد فصل ما بين مقدار العدد (Magnitude) و إشارته (Sign)، حيث أن جميع الخانات بما في ذلك خانة الإشارة تدخل في حساب مقدار العدد.

تدريب 4:

وضح طريقة تمثيل كل من القيم التالية في صورة عدد صحيح قصير بإشارة (Signed Short Integer)

(1) 15+ و 15- (2) 65+ و 65- (3) 1+ و 1- (4) 127+ و 127-

إيجاد مقدار العدد السالب

المطلوب مثلاً إيجاد القيمة العشرية للعدد الثنائي 11100101 إذا كان يُمَثِل عدداً صحيحاً قصيراً بإشارة.

نبدأ بتحديد إشارة العدد و ذلك بالنظر للخانة العليا MSB. في هذه الحالة نجد أن الخانة العليا MSB=1 مما يعني أن العدد سالب. لإيجاد مقدار عدد سالب نقوم بإيجاد المكمل الثاني له، لأن سالب العدد السالب عبارة عن عدد موجب كما نعلم.

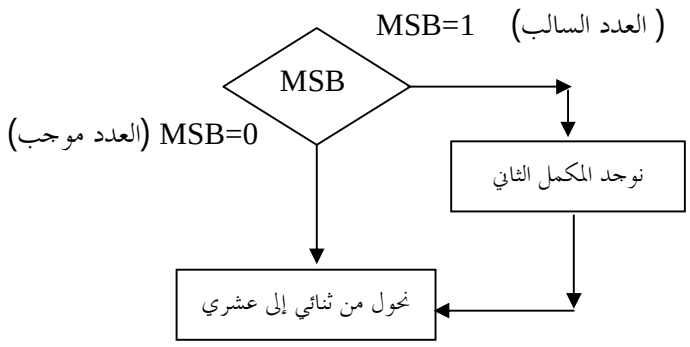
العدد	11100101
المكمل الأول	00011010
	1 +
المكمل الثاني	00011011

أخيراً نقوم بتحويل المقدار من الصورة الثنائية للصورة العشرية

$$(00011011)_2 = (11011)_2 = 27$$

إذن العدد هو 27-

وعموماً لإيجاد قيمة عدد صحيح بإشارة يمكن إستخدام المخطط التالي



مثال:

وضح طريقة تمثيل القيمة 13- في صورة:

(أ) عدد صحيح قصير بإشارة (Signed Short Integer)

(ب) عدد صحيح بإشارة (Signed Integer)

نقوم أولاً بتحويل المقدار إلى الصورة الثنائية $13 = (1101)_2$

(أ) عدد صحيح قصير بإشارة:

نكمل طول العدد إلى 8 خانات ثم نقوم بإيجاد المكمل الثاني له

العدد 00001101

المكمل الأول 11110010

$+ \quad 1$

المكمل الثاني 11110011

أي أن $-13 = (11110011)_2$

(ب) عدد صحيح بإشارة:

نكمل طول العدد إلى 16 خانة ثم نقوم بإيجاد المكمل الثاني له

العدد 0000000000001101

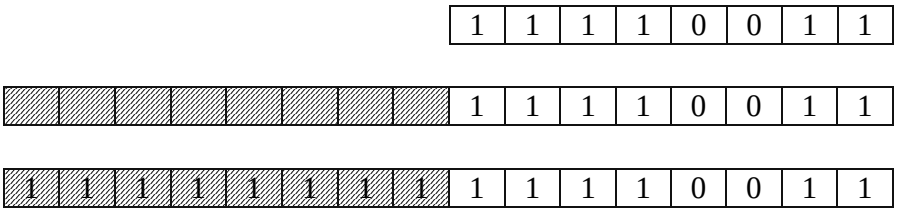
المكمل الأول 1111111111110010

$+ \quad 1$

المكمل الثاني 1111111111110011

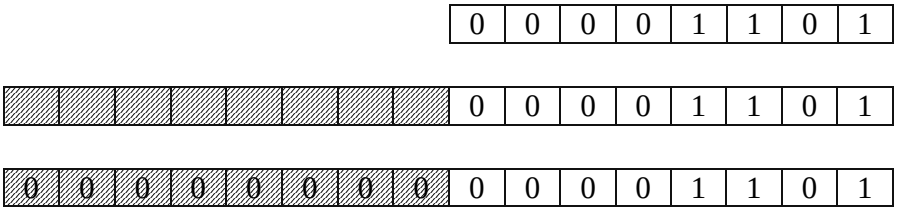
أي أن $-13 = (1111111111110011)_2$

في المثال السابق قمنا في (أ) بتمثيل العدد الصحيح ذو الإشارة 13- في 8 خانات ثم قمنا في (ب) بزيادة طول العدد إلى 16 خانة



لاحظ أننا قد قمنا بملء الخانات الفائضة إلى اليسار بـــ 1's

و بالمقارنة إذا أردنا تمثيل القيمة الموجبة 13+ في 8 خانات ثم في 16 خانة



لاحظ هنا أننا قد قمنا بملء الخانات الفائضة إلى اليسار بـــ 0's

يمكن بصورة عامة القول أنه عند زيادة طول العدد الصحيح ذو الإشارة فإننا نقوم بملء الخانات الفائضة إلى اليسار بإشارة العدد. و تسمى هذه العملية بـــ**تمديد الإشارة (Sign Extension)**.

سؤال تقويم ذاتي

قارن بين عملية تمديد العدد، أي زيادة طوله، في كل من الأعداد الصحيحة بدون إشارة (Unsigned Integers) والأعداد الصحيحة ذات الإشارة (Signed Integers)

مثال:

أوجد القيمة العشرية للعدد الثنائي 11110101 وذلك إذا كان يمثل:

(أ) عدد صحيح قصير بدون إشارة (Unsigned Short Integer).

(ب) عدد صحيح قصير بإشارة (Signed Short Integer).

(أ) العدد بدون إشارة (Unsigned) و بالتالي فإن كل الخانات تمثل مقدار العدد، و ما علينا إلا التحويل من الصورة الثنائية للصورة العشرية

$$(11110101)_2 = 2^7 + 2^6 + 2^5 + 2^4 + 2^2 + 2^0 = 128 + 64 + 32 + 16 + 4 + 1 = 245$$

(ب) العدد بإشارة (Signed) و عليه ننظر للخانة العليا MSB لتحديد إشارته. MSB=1 مما يعني أن العدد سالب.

لحساب المقدار نقوم بإيجاد المكمل الثاني

العدد		11110101
المكمل الأول		00001010
	+	1
المكمل الثاني		00001011

ثم نحول المقدار للصورة العشرية $(00001011)_2 = (1011)_2 = 2^3 + 2^1 + 2^0 = 11$

أي أن القيمة هي 11-

تدريب 5:

أوجد القيمة العشرية لكل من الأعداد الثنائية التالية إذا كان كل منها يمثل عدد قصير بإشارة (Signed Short)

(1) 10111111 (2) 10000000 (3) 01111111 (4) 11111111

مدى القيم التي يمكن تخزينها في مساحة معينة في صورة عدد صحيح بإشارة

لتوضيح الأمر نبدأ بالمثال التالي.

مثال: حدد جميع الأعداد الصحيحة ذات الإشارة (Signed Integers) التي يمكن تمثيلها في مساحة قدرها 4 خانات.

قيم موجبة (MSB=0)	القيمة العشرية (Decimal)	قيم سالبة (MSB=1)	القيمة العشرية (Decimal)
0000	+0	1000	-8
0001	+1	1001	-7
0010	+2	1010	-6
0011	+3	1011	-5
0100	+4	1100	-4
0101	+5	1101	-3
0110	+6	1110	-2
0111	+7	1111	-1

و عليه فإن مدى القيم التي يمكن تمثيلها في صورة عدد صحيح بإشارة (Signed Integer) طوله 4 خانات هو

$$-8 \sim +7$$

$$-2^3 \sim +2^3 - 1$$

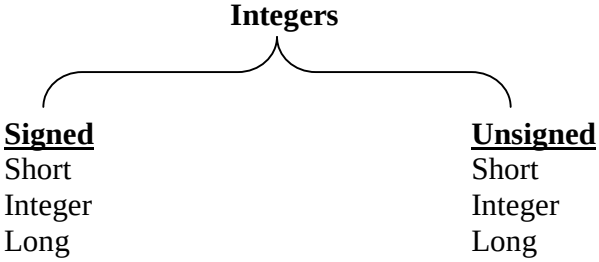
$$-2^{4-1} \sim +2^{4-1} - 1$$

و بصورة عامة فإن مدى الأعداد الصحيحة ذات الإشارة (Signed Integers) التي يمكن تمثيلها فى مساحة تبلغ N خانة هو

$$-2^{N-1} \sim +2^{N-1} - 1$$

و كملخص لما سبق فإن الأعداد الصحيحة (Integers) تنقسم من حيث الإشارة إلى نوعين: بإشارة (Signed) و بدون إشارة (Unsigned)

كما تنقسم الأعداد الصحيحة (سواء كانت بإشارة أو بدون إشارة)، من حيث الطول، إلى ثلاثة أنواع: Short و Integer و Long



ملاحظة:

عادة لا تذكر كلمة Signed صراحة في لغات البرمجة و إنما تفهم ضمناً، فمثلاً Integer تعني Signed Integer و Short Integer تعني Signed Short Integer. أما كلمة Unsigned فيجب أن تذكر صراحة.

الجدول التالي يوضح أنواع أنواع الأعداد الصحيحة و مدى القيم الذي يقبلها كل نوع

نوع العدد الصحيح	طوله	مدى القيم	
		Unsigned	**Signed**
Short Integer	1 Byte = 8 bits	$0 \sim (2^8 -1)$ $0 \sim 255$	$-2^7 \sim +(2^7 -1)$ $-128 \sim +127$
Integer	2 Bytes =16 bits	$0 \sim (2^{16} -1)$ $0 \sim 65,535$	$-2^{15} \sim +(2^{15} -1)$ $-32,768 \sim +32,767$
Long Integer	4 Bytes = 32 bits	$0 \sim (2^{32} -1)$ $0 \sim 4,294,967,295$	$-2^{31} \sim +(2^{31} -1)$ $-2,147,483,648 \sim +2,147,483,647$
———	N bits	$0 \sim (2^N -1)$	$-2^{N-1} \sim +(2^{N-1} -1)$

مما سبق يتضح لنا أن الأعداد الصحيحة يتم تمثيلها دون أي خطأ، أي بالدقة الكاملة، طالما أن عدد الخانات المتاحة يكفي لتمثيل القيمة. المشكلة الوحيدة التي يمكن أن تظهر في تمثيل الأعداد الصحيحة هي أن تكون القيمة المطلوب تخزينها خارج المدى المحدد للمساحة المتاحة، عند ذلك يحدث ما يسمى **Mathematical Over Flow**

تدريب 6:

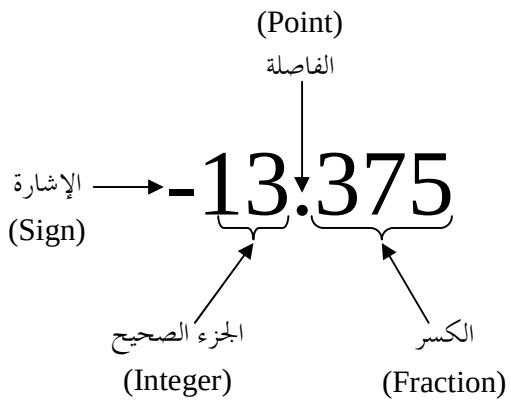
وضح ما يحدث إذا أردنا أن نقوم بتخزين القيمة العشرية 200 في صورة

(أ) عدد صحيح قصير بدون إشارة (Unsigned Short Integer)

(ب) عدد صحيح قصير بإشارة (Signed Short Integer)

2- الأعداد الحقيقية (Real Numbers)

العدد الحقيقي (Real Number) هو العدد الذي قد يكون محتوياً على كسر (Fraction)، مثل 13.375- أو 0.1 أو 5.3333 يتكون العدد الحقيقي من جزئين: عدد صحيح (Integer) و كسر (Fraction)، تفصل بينهما الفاصلة (Point)، و التي يطلق عليها في النظام العشري الفاصلة العشرية (Decimal Point). وللعدد الحقيقي إشارة (Sign). الشكل التالي يوضح أجزاء العدد الحقيقي



التحويل من الصورة العشرية للصورة الثنائية

لتمثيل العدد الحقيقي في الحاسوب، أو الأنظمة الرقمية عموماً، يجب أن يتم تحويله أولاً من الصورة العشرية (Decimal) إلى الصورة الثنائية (Binary). و هنا يتم تحويل كل من جزئي العدد الحقيقي على حدة. حيث نبدأ بتحويل الجزء الصحيح إلى الصورة الثنائية، و ذلك بنفس طريقة تحويل الأعداد الصحيحة، أي بالقسمة المتكررة على 2 و الاحتفاظ ببواقي القسمة. فإذا قمنا بتحويل الجزء الصحيح من العدد الحقيقي 13.375- إلى الصورة الثنائية نحصل على

$$13 = (1101)_2$$

لاحظ هنا أننا قد تجاهلنا إشارة العدد الحقيقي مؤقتاً، و سنقوم لاحقاً بتوضيح طريقة التعامل مع الإشارة.

بعد ذلك نقوم بتحويل الكسر من الصورة العشرية إلى الصورة الثنائية، و يتم ذلك بالضرب المتكرر في 2 و الاحتفاظ بالجزء الصحيح من النتيجة. فإذا أردنا تحويل الكسر في العدد الحقيقي 13.375- إلى الصورة الثنائية يتم ذلك كالتالي

$$0.375 \times 2 = 0.75 \rightarrow 0$$

عندما قمنا بضرب الكسر 0.375 في 2 حصلنا على 0.75 ، نقوم بالاحتفاظ بالجزء الصحيح من النتيجة و هو في هذه الحالة 0، ثم نقوم بضرب الكسر المتبقي في 2

$$0.75 \times 2 = 1.5 \rightarrow 1$$

عندما ضربنا الكسر المتبقي 0.75 في 2 حصلنا على 1.5 ، نحتفظ بالجزء الصحيح من النتيجة 1 ثم نضرب الكسر المتبقي في 2. لاحظ أن الكسر المتبقي هنا هو 0.5 بعد احتفاظنا بالـــ 1.

$$0.5 \times 2 = 1.0 \rightarrow 1$$

بضرب الكسر المتبقي 0.5 في 2 حصلنا على 1.0 ، نحتفظ بالجزء الصحيح من النتيجة 1 ، و نجد أن الكسر المتبقي قد أصبح 0.0 فنتوقف عن الضرب في 2 و بذلك تكون عملية تحويل الكسر إلى الصورة الثنائية قد انتهت. لمزيد من الوضوح نقوم بوضع خطوات التحويل معاً

$$\begin{array}{lcllll} 0.375 & \times & 2 & = & 0.75 \rightarrow & 0 \\ 0.75 & \times & 2 & = & 1.5 \rightarrow & 1 \\ 0.5 & \times & 2 & = & 1.0 \rightarrow & 1 \\ 0.0 & & & & & \end{array} \quad \downarrow$$

لتكوين الكسر في الصورة الثنائية نأخذ الأجزاء الصحيحة التي احتفظنا بها و نضعها بالترتيب (من أعلى إلى أسفل) على يمين الفاصلة (Point)، و نضع 0 يسار الفاصلة

$$0.011$$

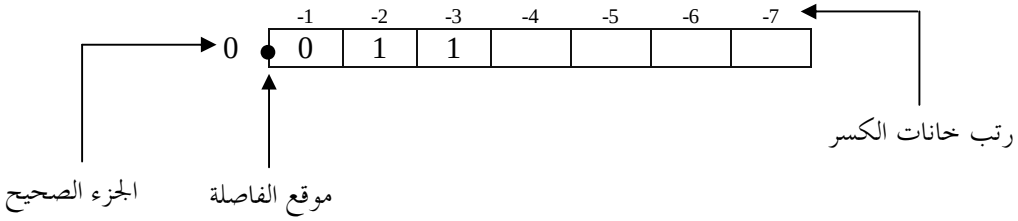
و عليه فإن الكسر العشري 0.375 يساوي 0.011 في الصورة الثنائية، و نكتب ذلك كالتالي

$$0.375 = (0.011)_2$$

لاحظ أن الفاصلة (Point) في الكسر الثنائي يطلق عليها الفاصلة الثنائية (Binary Point)، في حين يطلق عليها الفاصلة العشرية (Decimal Point) في الكسر العشري. و تسهيلاً للأمور سنستخدم تسمية الفاصلة (Point) فقط في النظامين الثنائي و العشري.

و للتحقق من صحة النتيجة التي حصلنا عليها عند تحويل الكسر من الصورة العشرية للصورة الثنائية يمكن أن نقوم بالعملية العكسية، أي تحويل الكسر من الصورة الثنائية و إعادته مرة أخرى إلى الصورة العشرية. و يتم ذلك بطريقة مشابهة لتحويل العدد الصحيح من الصورة الثنائية للصورة العشرية، أي بجمع أوزان الخانات الحاوية على القيمة 1 في

الكسر الثنائي. و وزن الخانة هنا هو أيضاً عبارة عن الأساس 2 مرفوع لأس يساوي رتبة الخانة. و نحصل على رتب الخانات هنا بترقيم الخانات بقيم سالبة مبتدئين بالقيمة 1- لأول خانة يمين الفاصلة ثم 2- للخانة التي تليها ... و هكذا.



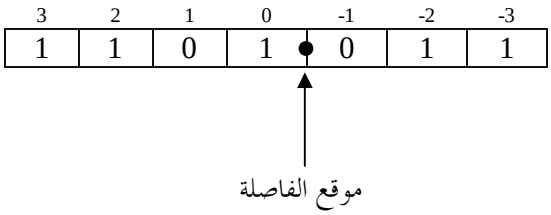
الخانات الحاوية على القيمة 1 هنا هي الخانة ذات الرتبة 2- و الخانة ذات الرتبة 3-. و أوزان هاتين الخانتين هي 2^{-2} و 2^{-3} على التوالي. و بناء عليه

$$(0.011)_2 = 2^{-2} + 2^{-3} = \frac{1}{2^2} + \frac{1}{2^3} = \frac{1}{4} + \frac{1}{8} = 0.25 + 0.125 = 0.375$$

الآن و بعد أن قمنا بتحويل جزئي العدد الحقيقي، الجزء الصحيح و الكسر، كل على حدة، من الصورة العشرية إلى الصورة الثنائية نقوم بوضع الجزئين معاً لتكوين العدد الحقيقي في الصورة الثنائية

$$13 = (1101)_2$$
$$0.375 = (0.011)_2$$
$$13.375 = (1101.011)_2$$

من الممكن أن نقوم بتحويل العدد الحقيقي كاملاً (بجزئيه) من الصورة الثنائية إلى الصورة العشرية كالتالي



بعد ترقيم الخانات و تحديد رتبها ما علينا إلا جمع أوزان الخانات الحاوية على 1، و وزن الخانة، كما ذكرنا من قبل، هو عبارة عن الأساس 2 مرفوع لأس يساوي رتبة الخانة. و عليه

$$(1101.011)_2 = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} = 8 + 4 + 1 + 0.25 + 0.125 = 13.375$$

كنوع من التسهيل فإن الجدول التالي يحتوي على أوزان الخانات الثمان الأولى ذات الرتب الموجبة و الخانات الثمان الأولى ذات الرتب السالبة

رتبة الخانة	وزنـــهـــــــــا		رتبة الخانة	وزنـــهـــــــــا	
0	2^0	1			
1	2^1	2	-1	2^{-1}	0.5
2	2^2	4	-2	2^{-2}	0.25
3	2^3	8	-3	2^{-3}	0.125
4	2^4	16	-4	2^{-4}	0.0625
5	2^5	32	-5	2^{-5}	0.03125
6	2^6	64	-6	2^{-6}	0.015625
7	2^7	128	-7	2^{-7}	0.0078125
8	2^8	256	-8	2^{-8}	0.00390625

مثال:

حول العدد العشري 9.625 إلى الصورة الثنائية.

نبدأ بتحويل الجزء الصحيح إلى الصورة الثنائية $9 = (1001)_2$

ثم نقوم بتحويل الكسر إلى الصورة الثنائية

$$\begin{array}{lcccl} 0.625 & \times & 2 & = & 1.25 \rightarrow 1 \\ 0.25 & \times & 2 & = & 0.5 \rightarrow 0 \\ 0.5 & \times & 2 & = & 1.0 \rightarrow 1 \\ 0.0 & & & & \end{array} \quad \downarrow$$

أي أن $0.625 = (0.101)_2$

و عليه $9.625 = (1001.101)_2$

إذا أردنا تحويل العدد الثنائي الناتج مرة أخرى إلى الصورة العشرية نقوم بترقيم الخانات ثم بجمع أوزان الخانات الحاوية على 1

3	2	1	0		-1	-2	-3
1	**0**	**0**	**1**	**•**	**1**	**0**	**1**

و عليه $(1001.101)_2 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 8 + 1 + 0.5 + 0.125 = 9.625$

مثال:

حول العدد الحقيقي 0.34375 من الصورة العشرية للصورة الثنائية.

العدد الحقيقي هنا مكون من كسر فقط و الجزء الصحيح فيه يساوي صفراً، لذلك نقوم فقط بتحويل الكسر من الصورة العشرية للصورة الثنائية

0.34375	×	2	=	0.6875	→	0	
0.6875	×	2	=	1.375	→	1	
0.375	×	2	=	0.75	→	0	↓
0.75	×	2	=	1.5	→	1	
0.5	×	2	=	1.0	→	1	
0.0							

أي أن $0.34375 = (0.01011)_2$

و يمكنك التحقق من صحة النتيجة بتحويل العدد الثنائي مرة أخرى إلى الصورة العشرية.

مثال:

حول العدد الحقيقي 0.1 من الصورة العشرية إلى الصورة الثنائية.

العدد هنا عبارة عن كسر فقط نقوم بتحويله إلى الصورة الثنائية

0.1	×	2	=	0.2	⟶	0	
0.2	×	2	=	0.4	⟶	0	
0.4	×	2	=	0.8	⟶	0	
0.8	×	2	=	1.6	⟶	1	
0.6	×	2	=	1.2	⟶	1	
0.2	×	2	=	0.4	⟶	0	
0.4	×	2	=	0.8	⟶	0	
0.8	×	2	=	1.6	⟶	1	↓
0.6	×	2	=	1.2	⟶	1	
0.2	×	2	=	0.4	⟶	0	
0.4	×	2	=	0.8	⟶	0	
0.8	×	2	=	1.6	⟶	1	
0.6	×	2	=	1.2	⟶	1	
0.2	×	2	=	0.4	⟶	0	
0.4	×	2	=	0.8	⟶	0	
M				M		M	

تذكر أننا نتوقف عن الضرب في 2 عندما يصبح الكسر المتبقي مساوياً صفراً، و هو ما لا يحدث في هذا المثال. السبب في ذلك هو أن الكسر في الصورة الثنائية عبارة عن كسر غير منتهي. و الكسور غير المنتهية أمر شائع في النظام العشري مثلاً

$$\frac{1}{3} = 0.333333333333333\,\mathbf{L}$$

$$\frac{1}{6} = 0.166666666666666\,\mathbf{L}$$

$$\frac{1}{12} = 0.08333333333333\,\mathbf{K}$$

$$\frac{1}{\sqrt{2}} = 0.7071067811865\,\mathbf{L}$$

في مثل هذه الحالات التي يكون فيها الكسر في الصورة الثنائية غير منتهي نتوقف عن الضرب في 2 عندما نحصل على عدد كافي من الخانات. و العدد الكافي من الخانات هنا غالباً ما تحدده المساحة المتاحة لتخزين الكسر، و هذه المساحة دائماً محدودة. و في المثال الذي نحن بصدده مثلاً توقفنا بعد الحصول على 15 خانة من الكسر الثنائي. أي أن

$$0.1 = (0.000110011001100\,\mathbf{L})_2$$

و بالطبع فإن 15 خانة فقط في الكسر الثنائي لا تمثل الكسر العشري 0.1 بدقة كاملة و إنما تمثله فقط بصورة تقريبية، و كلما زاد عدد خانات الكسر الثنائي كان أكثر دقة في تمثيل الكسر العشري، و لكن يتعذر تمثيل الكسر العشري بدقة كاملة لأن ذلك يتطلب عدداً لا نهائياً من الخانات في الكسر الثنائي.

من المثال السابق نستنتج أن تحويل بعض الأعداد الحقيقية إلى الصورة الثنائية قد ينتج عنه كسر غير منتهي، و مثل هذه الكسور غير المنتهية يتعذر تخزينها كاملة لأن عدد الخانات المتاحة للتخزين دائماً محدود.

تدريب 7:

حول القيم التالية من الصورة العشرية إلى الصورة الثنائية، ثم تحقق من صحة النتيجة بإعادة العدد الثنائي الناتج مرة أخرى إلى الصورة العشرية.		
53.8125 (1)	0.1015625 (2)	0.7 (3)

بهذا نكون قد أنجزنا الخطوة الأولى في تمثيل الأعداد الحقيقية (Real Numbers)، و هي تحويل العدد الحقيقي من الصورة العشرية إلى الصورة الثنائية، أي إلى مجموعة من الـــ0's و الـــ1's، لأن هذه هي الصورة الوحيدة التي يمكن أن تتعامل معها الدوائر الإلكترونية للحاسوب و بقية الأنظمة الرقمية (Digital Systems).

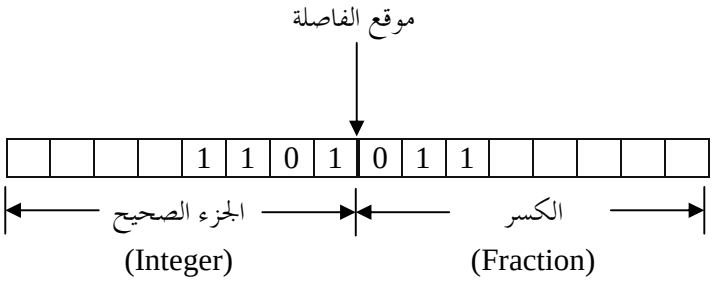
الخطوة التالية هي وضع العدد الحقيقي في صورته الثنائية في المساحة التخزينية المتاحة له. و هنا تواجهنا أول مشكلة، و هي أن العدد الحقيقي مكون من جزئين: جزء صحيح (Integer) و كسر (Fraction)، تفصل بينهما الفاصلة (Point). مثلاً إذا كان العدد الحقيقي المطلوب تمثيله هو 13.375-، فقد وجدنا أن

$$13.375 = (1101.011)_2$$

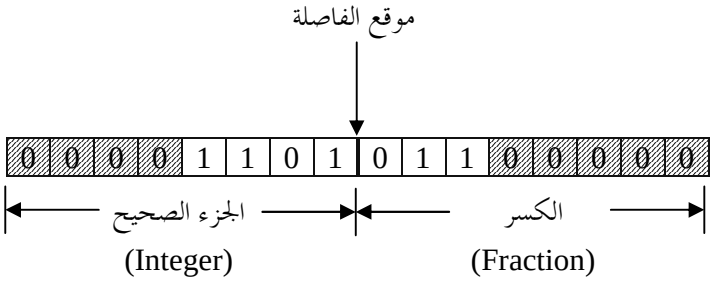
فإذا كانت المساحة التخزينية المتاحة عبارة عن 16 خانة ثنائية (16 bit)، فأين نضع الجزء الصحيح؟ و أين نضع الكسر؟ و كيف نقوم بتمثيل الفاصلة؟

أسلوب الفاصلة الثابتة (Fixed Point)

أحد الحلول البديهية التي قد تتبادر إلى الذهن هو أن نقوم بتقسيم المساحة المتاحة مابين الجزء الصحيح و الكسر. فنقول مثلاً أن الخانات الثماني الواقعة على اليمين للكسر و الخانات الثماني الواقعة على اليسار للجزء الصحيح

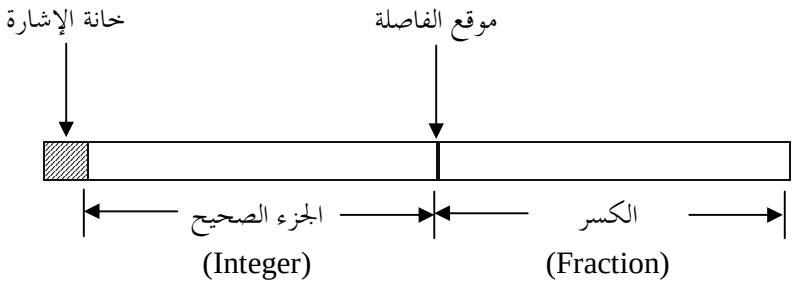


لاحظ أننا قد قمنا بمحاذاة الجزء الصحيح إلى يمين الجزء المخصص له، في حين قمنا بمحاذاة الكسر إلى يسار الجزء المخصص له، ثم نقوم بملء الخانات الفائضة على كل من يمين الكسر و يسار الجزء الصحيح بأصفار



لاحظ أنه في هذا الأسلوب لتمثيل الأعداد الحقيقية نفترض وجود الفاصلة (Point) عند الحد الفاصل ما بين الجزء من المساحة المخصص للعدد الصحيح و الجزء المخصص للكسر. و موقع الفاصلة هنا ثابت (Fixed)، لذلك يسمى هذا الأسلوب في تمثيل الأعداد الحقيقية بأسلوب **الفاصلة الثابتة (Fixed Point).**

أما عن إشارة العدد الحقيقي فيمكن تمثيلها هنا بأن يتم تخصيص خانة لها، و لتكن الخانة العليا (MSB)، و نضع في هذه الخانة القيمة 0 إذا كان العدد الحقيقي موجباً و القيمة 1 إذا كان سالباً.



يعيب هذا الأسلوب في تمثيل الأعداد الحقيقية عدم الإستغلال الأمثل للمساحة التخزينية المتاحة. فكثيراً ما يكون الجزء الصحيح من العدد الحقيقي مساوياً صفراً، أي أن العدد الحقيقي عبارة عن كسر فقط، و بالتالي يكون الجزء من المساحة التخزينية المخصص للعدد الصحيح غير مستغل. و يا حبذا لو كان في الإمكان أن نضيف هذه المساحة غير المستغلة للجزء المخصص للكسر، حيث أنه كلما زاد عدد خانات الكسر كان تمثيل العدد الحقيقي أكثر دقة (بسبب مشكلة الكسور غير المنتهية). و لكن مع الأسف هذا غير ممكن في أسلوب الفاصلة الثابتة (Fixed Point).

لاستغلال المساحة التخزينية المتاحة للعدد الحقيقي بصورة أكثر كفاءة يستخدم أسلوب آخر في تمثيل الأعداد الحقيقية يسمى بأسلوب **الفاصلة المتحركة (Floating Point)**.

أسلوب الفاصلة المتحركة (Floating Point)

يقوم هذا الأسلوب في تمثيل الأعداد الحقيقية على التخلص من الجزء الصحيح من العدد الحقيقي بحيث يصبح العدد بأكمله عبارة عن كسر. و يتم ذلك بتحريك أو إزاحة الفاصلة. و عملية تحريك الفاصلة من العمليات المألوفة بالنسبة لنا في الصورة العشرية (Decimal). مثلاً يمكن أن نقوم بتحريك الفاصلة في العدد الحقيقي العشري 123.456 يميناً أو يساراً كالتالي:

تحريك الفاصلة يساراً

$$\begin{aligned} 123.456 &= 12.3456\times10^{1} \\ &= 1.23456\times10^{2} \\ &= 0.123456\times10^{3} \\ &= 0.0123456\times10^{4} \end{aligned}$$

تحريك الفاصلة يميناً

$$\begin{aligned} 123.456 &= 1234.56\times10^{-1} \\ &= 12345.6\times10^{-2} \\ &= 123456.0\times10^{-3} \\ &= 1234560.0\times10^{-4} \end{aligned}$$

لاحظ أنه عندما يتم إزاحة الفاصلة من موقعها الأصلي يميناً أو يساراً يتم الحفاظ على قيمة العدد بالضرب في الأساس 10 مرفوعاً لأس يساوي عدد خانات الإزاحة، و يكون هذا الأس موجباً عندما تكون الإزاحة إلى اليسار و سالباً عندما تكون الإزاحة إلى اليمين. أي أن مقدار الأس يمثل عدد خانات الإزاحة للفاصلة، و إشارته تمثل اتجاه تلك الإزاحة.

و بنفس الطريقة يمكن إزاحة الفاصلة في الأعداد الحقيقية في الصورة الثنائية، و لكن الحفاظ على القيمة هنا يتم بالضرب في الأساس 2 مرفوعاً لأس يساوي مقداره عدد خانات الإزاحة و إشارته اتجاه الإزاحة. مثلاً يمكن تحريك الفاصلة في العدد الحقيقي الثنائي 1101.011 يميناً أو يساراً كالتالي:

تحريك الفاصلة يساراً

$$\begin{aligned} 1101.011 &= 110.1011\times 2^{1} \\ &= 11.01011\times 2^{2} \\ &= 1.101011\times 2^{3} \\ &= 0.1101011\times 2^{4} \end{aligned}$$

تحريك الفاصلة يميناً

$$\begin{aligned} 1101.011 &= 11010.11\times 2^{-1} \\ &= 110101.1\times 2^{-2} \\ &= 1101011.0\times 2^{-3} \\ &= 11010110.0\times 2^{-4} \end{aligned}$$

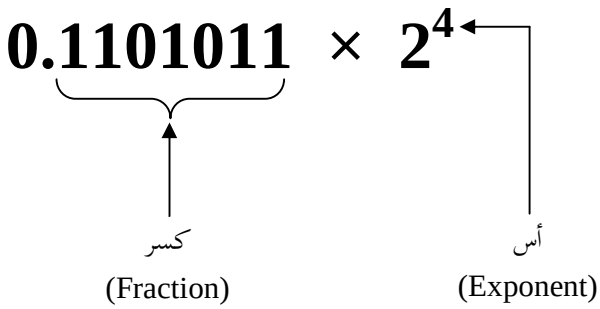
لتمثيل العدد الحقيقي نستخدم عملية تحريك الفاصلة للتخلص من الجزء الصحيح من العدد الحقيقي و تحويل العدد بأكمله إلى كسر. و تسمى هذه العملية بعملية **التطبيع (Normalization)**. فإذا أردنا مثلاً إجراء عملية تطبيع للعدد الحقيقي الثنائي 1101.011 يتم ذلك كالتالي:

$$1101.011 = 0.1101011\times 2^{4}$$

لاحظ أننا قد قمنا هنا بإزاحة الفاصلة 4 خانات إلى اليسار حتى أصبح الجزء الصحيح من العدد الحقيقي مساوياً 0. و بالطبع كان يمكن أن نقوم بإزاحة الفاصلة إلى اليسار أكثر من ذلك، 5 أو 6 خانات، و سيظل الجزء الصحيح من العدد الحقيقي مساوياً 0 أيضاً. و لكن هناك شرط آخر مهم في عملية التطبيع يضمن لنا عدم إزاحة الفاصلة أكثر من اللازم، و هذا الشرط هو أن تكون أول خانة ثنائية على يمين الفاصلة في الكسر حاوية على القيمة 1. أي أنه في عملية التطبيع نقوم بإزاحة الفاصلة عدداً من الخانات يميناً أو يساراً بحيث:

1- يصبح الجزء الصحيح مساوياً 0.
2- تكون أول خانة ثنائية على يمين الفاصلة حاوية على 1.

بعد إجراء عملية التطبيع (Normalization) يصبح العدد الحقيقي مكوناً من **كسر (Fraction)** و **أس (Exponent)**.



الكسر (Fraction) يشتمل على كل الخانات الثنائية (bits) الممثلة للعدد الحقيقي، والأس (Exponent) يحتفظ بموقع الفاصلة (Point).

مثال:

حول العدد العشري 3.625 إلى الصورة الثنائية ثم قم بتطبيعه.

نبدأ بتحويل الجزء الصحيح إلى الصورة الثنائية $3 = (11)_2$

ثم نقوم بتحويل الكسر إلى الصورة الثنائية

$$\begin{array}{lcllll} 0.625 & \times & 2 & = & 1.25 & \rightarrow & 1 \\ 0.25 & \times & 2 & = & 0.5 & \rightarrow & 0 \\ 0.5 & \times & 2 & = & 1.0 & \rightarrow & 1 \\ 0.0 & & & & & & \end{array}$$

أي أن $0.625 = (0.101)_2$

و عليه $3.625 = (11.101)_2$

لتطبيع العدد 11.101 نقوم بإزاحة الفاصلة خانتين إلى اليسار

$$11.101 = 0.11101 \times 2^2$$

مثال:

حول العدد العشري 0.203125 إلى الصورة الثنائية ثم قم بتطبيعه.

نقوم بتحويل الكسر إلى الصورة الثنائية

0.203125	×	2	=	0.40625	→	0	
0.40625	×	2	=	0.8125	→	0	
0.8125	×	2	=	1.625	→	1	↓
0.625	×	2	=	1.25	→	1	
0.25	×	2	=	0.5	→	0	
0.5	×	2	=	1.0	→	1	
0.0							

أي أن $0.203125 = (0.001101)_2$

لإجراء عملية التطبيع نحتاج لإزاحة الفاصلة خانتين إلى اليمين

$$0.001101 = 0.1101 \times 2^{-2}$$

تدريب 8:

حول القيم التالية من الصورة العشرية إلى الصورة الثنائية، ثم قم بتطبيعها.		
53.8125 (1)	0.1015625 (2)	0.7 (3)

تخزين الأعداد الحقيقية

بعد الانتهاء من تحويل العدد الحقيقي من الصورة العشرية إلى الصورة الثنائية ثم تطبيعه، مطلوب الآن وضع العدد في المساحة التخزينية المتاحة له. و المعلومات المطلوب وضعها في المساحة المتاحة هي:

1- الكسر (Fraction)
2- الأس (Exponent)
3- الإشارة (Sign)

المقصود بالإشارة (Sign) هنا هو إشارة العدد الحقيقي.

لاحظ أن الكسر (Fraction) في هذه المرحلة جاهز في الصورة الثنائية، و لكن الأس عبارة عن عدد صحيح بإشارة في الصورة العشرية و يجب تحويله إلى الصورة الثنائية.

في بدايات تعامل الحواسيب مع الأعداد الحقيقية (Real Numbers) - و كانت قبل ذلك تتعامل مع الأعداد الصحيحة (Integers) فقط - حدث اختلاف ما بين الشركات الرئيسية المصنعة للحواسيب في ذلك الوقت في طريقة

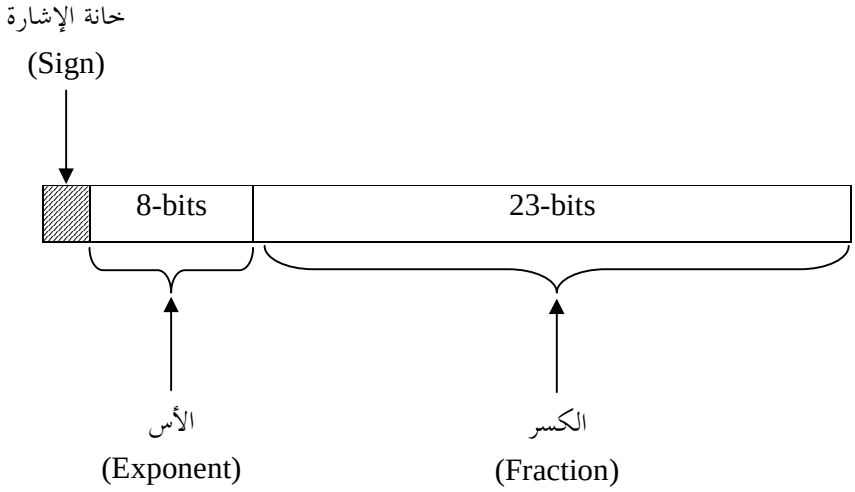
وضع المعلومات الممثلة للعدد الحقيقي (الكسر و الأس و الإشارة) داخل المساحة المخصصة لتخزين العدد الحقيقي، الأمر الذي أدى إلى حدوث مشاكل عند نقل البيانات ما بين تلك الحواسيب. استمر ذلك حتى قامت جمعية IEEE (**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers) بتدارك الأمر بوضع مواصفات قياسية (Standards) لتخزين الأعداد الحقيقية. تم في هذه المواصفات تعريف نوعين من الأعداد الحقيقية:

1- العدد الحقيقي ذو الدقة العادية (IEEE Single Precision Float)

2- العدد الحقيقي ذو الدقة المضاعفة (IEEE Double Precision Float)

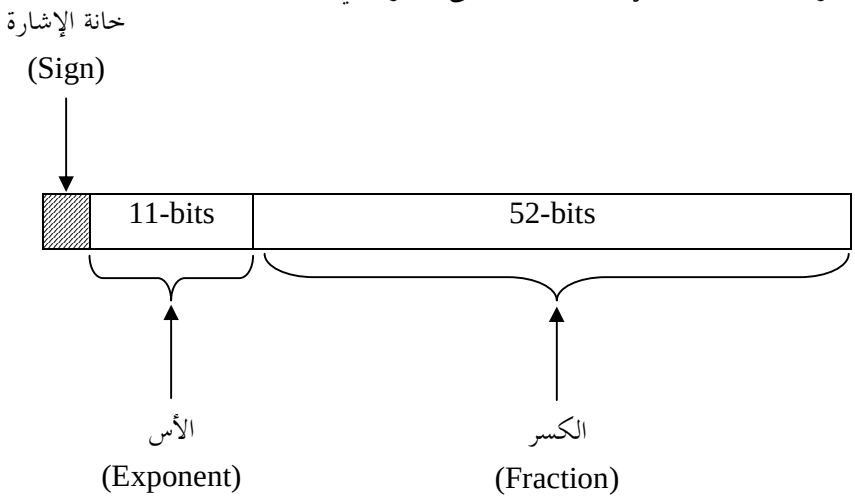
العدد الحقيقي ذو الدقة العادية (Single Precision)

طوله هو 4 Bytes = 32 bits ، مقسمة على النحو التالي:



العدد الحقيقي ذو الدقة المضاعفة (Double Precision)

طوله هو 8 Bytes = 64 bits ، مقسمة على النحو التالي:



عند وضع الكسر في الجزء المخصص له تتم محاذاته إلى اليسار، مع ملء الخانات الفائضة على يمينه بأصفار. أما الأس فيتم عند وضعه في الجزء المخصص له التعامل معه كعدد صحيح بإشارة (Signed Integer)، أي تتم محاذاته إلى اليمين مع ملء الخانات الفائضة على يساره بإشارته (راجع تمثيل الأعداد الصحيحة ذات الإشارة). أما خانة الإشارة فنضع فيها القيمة 0 إذا كان العدد الحقيقي موجباً و القيمة 1 إذا كان العدد سالباً.

إذا أردنا تمثيل العدد الحقيقي 0.1101011×2^{4} في صورة عدد حقيقي ذو دقة عادية (Single Precision) فإن ذلك يتم كالتالي:

- نقوم أولاً بتحويل الأس من الصورة العشرية للصورة الثنائية فنحصل على $(100)_2 = 4$
- نضع كل من الكسر و الأس في الجزء المخصص له
- نضع الإشارة

0	0 0 0 0 0 0 1 0 0	1 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

الخانات المظللة هنا هي الخانات الفائضة. فالخانات الفائضة على يمين الكسر يتم ملؤها بأصفار دائماً، أما الخانات الفائضة على يسار الأس فقد يتم ملؤها بأصفار (كما هو الحال هنا) إذا كان الأس موجباً، و قد يتم ملؤها بـــ1's إذا كان الأس سالباً (راجع تمثيل الأعداد الصحيحة ذات الإشارة).

إذا أردنا تمثيل نفس القيمة في صورة عدد حقيقي بدقة مضاعفة (Double Precision) يكون ذلك كالتالي:

0	00000000100	11010110000000000000000000000000000000000000000000000000

مثال:

وضح طريقة تمثيل القيمة 25.687- في صورة عدد حقيقي:

(أ) ذو دقة عادية

(ب) ذو دقة مضاعفة.

الخطوة الأولى هي التحويل من الصورة العشرية للصورة الثنائية $25.6875 = (11001.1011)_2$

الخطوة الثانية هي التطبيع $11001.1011 = 0.110011011\times2^{5}$

نقوم بتحويل الأس من الصورة العشرية إلى الصورة الثنائية $5 = (101)_2$

الخطوة الثالثة و الأخيرة هي التخزين:

الدقة العادية:

1	00000101	11001101100000000000000

الدقة المضاعفة:

1	00000000101	11001101100000000000000000000000000000000000000000000000000

لاحظ أن العدد الحقيقي في هذا المثال سالب لذلك قمنا بوضع القيمة 1 في خانة الإشارة.

مثال:

وضح طريقة تمثيل القيمة 0.1 في صورة عدد حقيقي:

(أ) ذو دقة عادية

(ب) ذو دقة مضاعفة.

الخطوة الأولى هي التحويل من الصورة العشرية للصورة الثنائية

و قد قمنا بتحويل القيمة 0.1 من الصورة العشرية إلى الصورة الثنائية في مثال سابق و كانت النتيجة

$$0.1 = (0.000110011001100\,\mathbf{L})_2$$

و لاحظنا حينذاك أن الكسر الثنائي الناتج غير منتهي

الخطوة الثانية هي التطبيع

$$0.000110011001100\,\mathbf{L} = 0.110011001100\,\mathbf{L} \times 2^{-3}$$

نقوم بتحويل الأس من الصورة العشرية إلى الصورة الثنائية $3 = (11)_2$

و لأن الكسر سالب يجب إيجاد المكمل الثاني (2's Complement) له

العدد (في 8 خانات)	00000011
المكمل الأول	11111100
	+ 1
المكمل الثاني	11111101

أي أن $-3 = (11111101)_2$

الخطوة الثالثة و الأخيرة هي التخزين:

لاحظ أننا نحتاج لملء الجزء المخصص للكسر إلى 23 خانة من خانات الكسر غير المنتهي في حالة الدقة العادية، و إلى 52 خانة في حالة الدقة المضاعفة، هذا بخلاف الأصفار الثلاثة الأولى يمين الفاصلة و التي يتم التخلص منها في عملية التطبيع. ففي عملية التحويل يجب أن نستمر في الضرب في 2 حتى نحصل على كل الخانات المطلوبة.

لكن إذا عدت إلى المثال الذي قمنا فيه بتحويل القيمة 0.1 من الصورة العشرية إلى الصورة الثنائية تجد أننا قد توقفنا عن الضرب في 2 بعد الحصول على 15 خانة فقط من خانات الكسر غير المنتهي. وسبب توقفنا هو أننا نلاحظ بعد ذلك أن الكسر عبارة عن كسر دوري (periodic) يتكرر فيه النمط 1100، بعد الأصفار الثلاثة الأولى يمين الفاصلة، باستمرار إلى ما لا نهاية. أي أن

$$0.1 = (0.0001100110011001100110011001100110011001100110011001100\,\mathbf{L})_2$$

الدقة العادية:

0	11111101	11001100110011001100110

لاحظ أنه قد تم تخزين أول 23 خانة فقط من الكسر غير المنتهي و اقتطاع الباقي.

الدقة المضاعفة:

0	11111111101	1100110011001100110011001100110011001100110011001100

لاحظ أنه قد تم تخزين أول 52 خانة فقط من الكسر غير المنتهي و اقتطاع الباقي.

مثل هذا الاقتطاع للكسر الذي تم في المثال السابق يؤدي إلى حدوث **خطأ (Error)** في تمثيل العدد الحقيقي. فالقيمة المخزنة الآن ليست هي القيمة 0.1 تماما و إنما تقريب لها. و بالطبع فإن الخطأ في حالة الدقة المضاعفة أقل منه في حالة الدقة العادية، حيث تمكنا من تخزين عدد أكبر من خانات الكسر في حالة الدقة المضاعفة. و مثل هذه الأخطاء غالباً ما تحدث عند تمثيل الأعداد الحقيقية و يطلق عليها أخطاء التدوير (Round off Errors)، و هذه الأخطاء غير موجودة في تمثيل الأعداد الصحيحة.

تدريب 9:

وضح طريقة تمثيل كل من القيم التالية في صورة عدد حقيقي ذو دقة عادية (Single Precision).

53.8125 (1) -0.1015625 (2) 0.7 (3)

حساب مقدار الخطأ في تمثيل العدد الحقيقي

ذكرنا أنه غالباً ما يكون هناك خطأ (Error) في تمثيل الأعداد الحقيقية، و أن الخطأ ناتج عن اقتطاع جزء من الكسر نظراً لمحدودية المساحة التخزينية المتاحة له.

سنقوم الآن بحساب مقدار الخطأ في تمثيل العدد العشري الحقيقي 0.1 في المثال السابق بالدقة العادية.

الأسلوب الذي سنتبعه في حساب مقدار الخطأ في التمثيل هو أن ننظر للقيمة المخزنة فعلاً و نقوم بتحويلها من الصورة الثنائية إلى الصورة العشرية، و نقارن النتيجة مع القيمة التي كان من المفترض تمثيلها. الفرق بين القيمتين يمثل مقدار الخطأ.

القيمة المخزنة هي $0.11001100110011001100110\times 2^{-3}$

نقوم بإعادة الفاصلة إلى موقعها الأصلي $0.00011001100110011001100110$

نوجد رتب جميع الخانات الحاوية على 1 و هي 4-، 5-، 8-، 9-، 12-، 13-، 16-، 17-، 20-، 21-، 24-، 25-

وعليه تكون القيمة المخزنة هي

$$\begin{aligned} x &= 2^{-4}+2^{-5}+2^{-8}+2^{-9}+2^{-12}+2^{-13}+2^{-16}+2^{-17}+2^{-20}+2^{-21}+2^{-24}+2^{-25} \\ &= 0.09999999404 \end{aligned}$$

أي ان مقدار الخطأ في التمثيل هو $0.1-x=0.1-0.09999999404=5.96\times 10^{-9}$

تدريب 10:

احسب مقدار الخطأ في تمثيل القيمة 0.7 في صورة عدد حقيقي ذو دقة عادية (Single Precision)

مثال:

أوجد القيمة العشرية للعدد الثنائي

11111111111010000000000000000000

إذا علمت أنه يمثل عدداً حقيقياً ذو دقة عادية (IEEE Single Precision Float).

نقوم بتقسيم العدد إلى إشارة و أس و كسر

1	11111111	10100000000000000000000

العدد الحقيقي سالب لأن خانة الإشارة تحتوي على القيمة 1.

الأس عبارة عن عدد صحيح سالب لأن الخانة العليا (MSB) فيه مساوية 1. فبإيجاد المكمل الثاني له و تحويله إلى الصورة العشرية نجد أن الأس يساوي 1-.

وعليه فإن القيمة المخزنة هي 0.1101×2^{-1}

بإعادة الفاصلة إلى موقعها الأصلي يصبح العدد 0.01101

بالتحويل من الصورة الثنائية إلى الصورة العشرية نجد أن مقدار القيمة المخزنة هو

$$x = 2^{-2} + 2^{-3} + 2^{-5} = 0.40625$$

و بأخذ الإشارة السالبة في الاعتبار تكون القيمة المخزنة هي -0.40625

مدى القيم التي يمكن تمثيلها في صورة أعداد حقيقية

- الأعداد الحقيقية ذات الدقة العادية (Single Precision) $\pm 10^{-38}$ – $\pm 10^{+38}$
- الأعداد الحقيقية ذات الدقة المضاعفة (Double Precision) $\pm 10^{-308}$ – $\pm 10^{+308}$

تمثيل الرموز (Characters)

المقصود بالرموز (characters) هنا هو

- الحروف الكبيرة (Capital Letters) **A, B, C, D, …, Z** (و عددها 26)
- الحروف الصغيرة (Small Letters) **a, b, c, d, …, z** (و عددها 26)
- الأرقام (Digits) **0, 1, 2, 3, …, 9** (و عددها 10)
- علامات الترقيم (Punctuation Marks)

! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ' { | } ~ (و عددها 32)

- الرموز البيضاء (White Characters) مثل:

Space، **Horizontal Tab**، **Carriage Return**، **New Line**، … (و عددها حوالي 6)

- رموز تحكم (Control Characters) مثل:

Del، **ESC**، **Bell**، **Back Space**، **Form Feed**، … (و عددها حوالي 10)

أي أن العدد الكلي للرموز هو 10+6+32+10+26+26 أي حوالي **110** رمزاً

و يتم تمثيل هذه الرموز باستخدام شفرة ثنائية (Binary Code) بحيث يكون لكل رمز منها شفرة فريدة تميزه.
و اقل عدد من الخانات يلزم لتمثيل جميع الرموز هو 7 خانات (7 bits)، حيث أن عدد الشفرات الثنائية المتاحة في هذه الحالة هو $2^7 = 128$ و هذا العدد يكفي لتمثيل جميع الرموز.

شفرات تمثيل البيانات

توجد طرق عديدة يمكن بها أن يتم تخصيص الشفرات الثنائية المتاحة للرموز المختلفة، مما قد يؤدي إلى اختلافات كبيرة في تمثيل البيانات. و منعاً للاختلاف تم الاتفاق عالمياً على طرق محددة لتمثيل البيانات، و تم توثيق هذه الطرق في المؤسسات المعنية، و يتم مراجعتها و تطويرها و نشرها بانتظام لكي يلتزم الجميع بها. الأمر الذي جعل تبادل البيانات على نطاق واسع، خاصة في عصر الإنترنت، أمراً ممكناً.

سنتعرض في الجزء التالي لعدد من الشفرات القياسية (Standard Codes) المستخدمة حالياً في تمثيل البيانات.

شفرة ASCII

كلمة ASCII اختصار للعبارة (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange). و شفرة ASCII عبارة عن شفرة ثنائية مكونة من سبعة خانات تستخدم في تمثيل الرموز. و تعتبر الشفرة الأكثر استخداماً لهذا الغرض و الأوسع انتشاراً حالياً. تم ابتكار شفرة ASCII في الأساس لتمثيل الرموز في آلات تسمى التيلي تايب (Teletype Machines) وهي عبارة عن وسيلة اتصال استخدمت في السابق لنقل البيانات، و تتكون مما يشبه الآلتين الكاتبتين (Typewriters)، إحداهما مرسلة و الأخرى مستقبلة. عند طباعة أي نص على لوحة مفاتيح الآلة المرسلة يظهر ذلك النص مطبوعاً على الورق في الآلة المستقبلة. و يعتبر جهاز التلكس (Telex) مثالاً لهذا النوع من الآلات.

و يوضح الجدول التالي الشفرات الثنائية المستخدمة لتمثيل الرموز في شفرة ASCII:

Code		Char
Binary	Hex	
0000000	00	**nul**
0000001	01	**soh**
0000010	02	**stx**
0000011	03	**etx**
0000100	04	**eot**
0000101	05	**enq**
0000110	06	**ack**
0000111	07	**bel**
0001000	08	**bs**
0001001	09	**ht**
0001010	0A	**nl**
0001011	0B	**vt**
0001100	0C	**ff**
0001101	0D	**cr**
0001110	0E	**so**
0001111	0F	**si**
0010000	10	**dle**
0010001	11	**dc1**
0010010	12	**dc2**
0010011	13	**dc3**
0010100	14	**dc4**
0010101	15	**nak**
0010110	16	**syn**
0010111	17	**etb**
0011000	18	**can**
0011001	19	**em**
0011010	1A	**sub**
0011011	1B	**esc**
0011100	1C	**fs**
0011101	1D	**gs**
0011110	1E	**rs**
0011111	1F	**us**

Code		Char
Binary	Hex	
0100000	20	**sp**
0100001	21	**!**
0100010	22	**“**
0100011	23	**#**
0100100	24	**$**
0100101	25	**%**
0100110	26	**&**
0100111	27	**‘**
0101000	28	**(**
0101001	29	**)**
0101010	2A	*****
0101011	2B	**+**
0101100	2C	**,**
0101101	2D	**-**
0101110	2E	**.**
0101111	2F	**/**
0110000	30	**0**
0110001	31	**1**
0110010	32	**2**
0110011	33	**3**
0110100	34	**4**
0110101	35	**5**
0110110	36	**6**
0110111	37	**7**
0111000	38	**8**
0111001	39	**9**
0111010	3A	**:**
0111011	3B	**;**
0111100	3C	**<**
0111101	3D	**=**
0111110	3E	**>**
0111111	3F	**?**

Code		Char
Binary	Hex	
1000000	40	**@**
1000001	41	**A**
1000010	42	**B**
1000011	43	**C**
1000100	44	**D**
1000101	45	**E**
1000110	46	**F**
1000111	47	**G**
1001000	48	**H**
1001001	49	**I**
1001010	4A	**J**
1001011	4B	**K**
1001100	4C	**L**
1001101	4D	**M**
1001110	4E	**N**
1001111	4F	**O**
1010000	50	**P**
1010001	51	**Q**
1010010	52	**R**
1010011	53	**S**
1010100	54	**T**
1010101	55	**U**
1010110	56	**V**
1010111	57	**W**
1011000	58	**X**
1011001	59	**Y**
1011010	5A	**Z**
1011011	4B	**[**
1011100	5C	****
1011101	5D	**]**
1011110	5E	**^**
1011111	5F	**_**

Code		Char
Binary	Hex	
1100000	60	**’**
1100001	61	**A**
1100010	62	**B**
1100011	63	**C**
1100100	64	**D**
1100101	65	**E**
1100110	66	**F**
1100111	67	**G**
1101000	68	**H**
1101001	69	**I**
1101010	6A	**J**
1101011	6B	**K**
1101100	6C	**L**
1101101	6D	**M**
1101110	6E	**N**
1101111	6F	**O**
1110000	70	**P**
1110001	71	**Q**
1110010	72	**R**
1110011	73	**S**
1110100	74	**T**
1110101	75	**U**
1110110	76	**V**
1110111	77	**W**
1111000	78	**X**
1111001	79	**Y**
1111010	7A	**Z**
1111011	7B	**{**
1111100	7C	**|**
1111101	7D	**}**
1111110	7E	**~**
1111111	7F	**Del**

إذا قمنا بالتدقيق في الجدول السابق نتوصل إلى بعض الملاحظات الدالة على أن تخصيص الشفرات الثنائية للرموز المختلفة في شفرة ASCII لم يتم بصورة اعتباطية و إنما تم بطريقة حكيمة. لاحظ مثلاً العلاقة ما بين الشفرات الممثلة للأرقام (Digits) **0-9** و قيم تلك الارقام، تجد أن هناك فرقاً ثابتاً مقداره $(30)_{16}$ ما بين شفرة الرقم و قيمته، مما يسهل من عملية تحويل رموز الأرقام إلى القيم المقابلة لها، و هي عملية نحتاج لها كثيراً في الحاسوب و الأنظمة الرقمية الأخرى. لاحظ أيضاً وجود علاقة رياضية ثابتة ما بين شفرة ASCII للحرف الكبير (Capital Letter) ونظيره الصغير (Small Letter)، تجد أن الفرق بين شفرتيهما هو $(20)_{16}$، مما يجعل من عملية تحويل الأحرف الكبيرة إلى أحرف صغيرة أو العكس في نص معين عملية سهلة.

عندما استخدمت شفرة ASCII في تمثيل الرموز في الحاسوب، ظهرت مشكلة الخانة الثامنة (8^{th} bit)، حيث أن التخزين في الحواسيب مبني على نظام الـــByte المكون من 8 bits، في حين أن شفرة ASCII عبارة عن شفرة مكونة من سبعة خانات (7-bit Code)، لذلك كان لابد من إيجاد استخدام للخانة الثامنة. و هنالك طريقتان لاستغلال هذه الخانة:

1. يمكن استخدام الخانة الثامنة لمضاعفة عدد الرموز التي يمكن تمثيلها بحيث يصبح 256 رمزاً بدلاً عن 128 رمزاً. هذه الـــ 256 رمزاً تكون الـــ 128 رمزاً الأولى منها هي رموز شفرة ASCII القياسية، أما الـــ 128 رمزاً الإضافية فيمكن استخدامها في تمثيل أحرف اللغات الأخرى، مثل اللغة العربية، أو في تمثيل بعض الرموز الخاصة المستخدمة مثلاً في الرسومات أو في بناء الجداول أو في كتابة المعادلات الرياضية و غير ذلك.
2. يمكن إستخدام الخانة الثامنة في عملية تسمى **عملية التحقق (Parity Check)**، وهي عملية تستخدم لاكتشاف حدوث خطأ (Error) في نقل البيانات. حيث أنه عند نقل البيانات لمسافات طويلة عبر وسائل الاتصال المختلفة قد تتعرض تلك البيانات لحدوث أخطاء. فلاكتشاف حدوث مثل هذه الأخطاء يتفق كل من الطرف المرسل للبيانات و الطرف المستقبل لها على أن يكون العدد الكلي للـــ 1's في أي رمز مرسل فردياً مثلاً، و هو مايسمى **بالتحقق الفردي (Odd Parity)**. و بناء على ذلك يقوم الطرف المرسل قبل إرسال أي رمز بحساب عدد الـــ 1's الموجودة فيه، فإذا وجد أن عددها فردي يقوم بوضع 0 في الخانة الثامنة، وذلك للحفاظ على العدد الكلي للـــ 1's في الرمز فردياً. أما إذا وجد أن عدد الـــ 1's في الرمز المرسل زوجياً فإنه يقوم بوضع 1 في الخانة الثامنة، بحيث يصبح العدد الكلي للـــ 1's في الرمز فردياً. أي أن مهمة الطرف المرسل هي التأكد من أن عدد الـــ 1's فردي في كل رمز يقوم بإرساله، و ذلك بوضع القيمة المناسبة في الخانة الثامنة، و التي يطلق عليها **خانة التحقق (Parity bit)**. أما بالنسبة للطرف المستقبل فإنه يقوم بحساب عدد الـــ 1's في أي رمز يصل إليه، فإذا وجد أن عددها فردي كان معنى ذلك عدم حدوث خطأ أثناء عملية النقل، أما إذا وجد أن عددها زوجي فمعنى ذلك حدوث خطأ. و الطريقة الوحيدة الممكنة لتصحيح الخطأ الذي حدث هنا هي أن يطلب الطرف المستقبل من الطرف المرسل إعادة إرسال الرمز الذي وصله خاطئاً، وهذا يتطلب بالطبع وجود إمكانية الاتصال في الاتجاهين، و هو أمر غير متاح في كثير من

الأحيان. لاحظ أن هذا الأسلوب فى اكتشاف حدوث الأخطاء يعجز عن اكتشاف حدوث خطأ فى خانتين فى وقت واحد. و لا توجد مشكلة هنا حيث أنه في أي نظام رقمي مصمم بصورة جيدة يكون احتمال حدوث خطأ في خانتين في وقت واحد أمراً نادر الحدوث بحيث يمكن تجاهله.

يمكن أيضاً أن يتفق الطرفان المرسل والمستقبل على أن يكون العدد الكلى للـــ 1's في أي رمز مرسل زوجياً، و يسمى هذا **بالتحقق الزوجي (Even Parity)**.

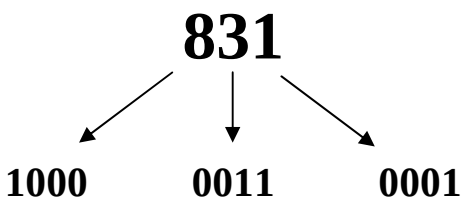
شفرة BCD (Binary Coded Decimal)

استخدمت هذه الشفرة في الماضي لتمثيل الأعداد الصحيحة (Integers) في الحواسيب الكبيرة (Main Frames) القديمة، خاصة تلك التي قامت بإنتاجها شركة IBM. في هذه الشفرة يتم تمثيل كل رقم من الأرقام من 0-9 باستخدام شفرة ثنائية مكونة من أربعة خانات (4-bits Binary Code)، و ذلك كالتالي:

الرقم (Digit)	الشفرة (Code)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

لاحظ أن الخانات الأربعة المستخدمة في التمثيل هنا تعطينا 16 شفرة (Code) مختلفة، استخدمنا منها فقط العشرة الأولى و تبقت 6 شفرات غير مستخدمة هي: 1010، 1011، 1100، 1101، 1110، 1111.

لتمثيل أي عدد صحيح باستخدام شفرة BCD نأخذ أرقام العدد فى الصورة العشرية و نستبدل كل رقم بشفرة BCD الخاصة به. مثلاً:



بتجميع شفرات BCD للأرقام فنحصل على $831 = (100000110001)_{BCD}$

لاحظ أن الأعداد الصحيحة الممثلة فى صورة BCD تشغل مساحة تخزينية أكبر من تلك التي تشغلها الأعداد الصحيحة الممثلة بالصورة التقليدية التى سبق لنا دراستها. كما أن إجراء العمليات الحسابية على الأعداد الممثلة فى صورة BCD به الكثير من المشاكل والصعوبات والتعقيدات.

تدريب 11:

ما هو أكبر عدد يمكن تخزينه في 1 Byte باستخدام شفرة BCD؟

شفرة EBCDIC (Extended Binary Coded Decimal Information Code)

هذه الشفرة هي عبارة عن تطوير لشفرة BCD بحيث تتمكن من تمثيل الرموز. و هي تشبه إلى حد كبير شفرة ASCII، إلا أن شفرة EBCDIC مكونة من 8 خانات (8 bits). استخدمت شفرة EBCDIC لتمثيل الرموز فى الحواسيب الكبيرة (Main Frames) التي تنتجها شركة IBM. و ما زالت إمكانية التعامل مع البيانات الممثلة باستخدام شفرة EBCDIC موجودة حتى الآن في الحواسيب التي تقوم بإنتاجها شركة IBM وذلك لتمكين مستخدمي هذه الأجهزة من الرجوع لبياناتهم القديمة.

الشفرة الرمادية (Gray Code)

يطلق على هذه الشفرة أيضاً تسمية **الشفرة المعكوسة (Reflected Code)**، و ذلك بسبب الأسلوب المستخدم في توليدها. تمتاز هذه الشفرة بأن كل رمزين متتاليين فيها يختلفان عن بعضهما البعض في bit واحد فقط. و يمكن أن نقوم بتوليد الشفرة الرمادية كالتالي:

شفرة رمادية من خانة واحدة:

الشفرة الرمادية المكونة من خانة واحدة هي نفس الشفرة الثنائية المكونة من خانة واحدة، أي

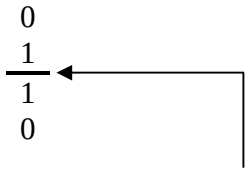
0
1

شفرة رمادية من خانتين:

لتوليد شفرة رمادية مكونة من خانتين نبدأ بالشفرة الرمادية المكونة من خانة واحدة، أي

0
1

نقوم بإجراء عملية عكس (Reflection) لهذه الشفرة (على سطح عاكس وهمي) فنحصل على



أخيراً نقوم بملء الخانة الثانية الواقعة إلى اليسار بـــ 0's أعلى السطح العاكس، و بـــ 1's أسفله، فنحصل على

0	0
0	1
1	1
1	0

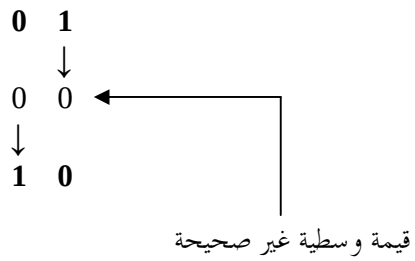
و بذلك تكون الشفرة الرمادية المكونة من خانتين هي

00
01
11
10

قارن مع الشفرة الثنائية المكونة من خانتين و هي

00
01
10
11

لاحظ الميزة الأساسية للشفرة الرمادية، و هي أنه عند الانتقال من قيمة إلى القيمة التي تليها فإن خانة واحدة فقط تتغير من 0 إلى 1 أو من 1 إلى 0. أما في الشفرة الثنائية فإنه عند الانتقال من 01 إلى 10 فإن كلا الخانتين تتغيران. و المشكلة هنا أنه من الصعب جداً في الأنظمة الرقمية أن نضمن حدوث التغير في الخانتين في نفس اللحظة، و غالباً ما تتغير إحدى الخانتين قبل الأخرى، مما قد ينتج عنه ظهور قيم وسطية غير صحيحة. مثلاً عند الانتقال من 01 إلى 10 إذا تغيرت الخانة اليمنى قبل الخانة اليسرى تصبح الخانة اليمنى 0 في حين أن الخانة اليسرى لم تزل 0 و لم تتغير بعد إلى 1 فتظهر القيمة الوسطية الخاطئة 00 لفترة زمنية قصيرة جداً، ثم تتغير الخانة اليسرى من 0 إلى 1 فتظهر القيمة الصحيحة 10.



و مثل هذه القيم الوسطية غير الصحيحة (Incorrect Intermediate Values) قد تؤدي إلى حدوث مشاكل في الأنظمة الرقمية على الرغم من فترة ظهورها القصيرة جداً. و في الشفرة الرمادية لا يمكن ظهور مثل هذه القيم.

شفرة رمادية من ثلاثة خانات:

لتوليد شفرة رمادية مكونة من ثلاثة خانات نبدأ بالشفرة الرمادية المكونة من خانتين التي قمنا بتوليدها، أي

0 0
0 1
1 1
1 0

نقوم بإجراء عملية عكس (Reflection) لهذه الشفرة (على سطح عاكس وهمي) فنحصل على

0 0
0 1
1 1
1 0
---
1 0
1 1
0 1
0 0

أخيراً نقوم بملء الخانة الثالثة الواقعة إلى اليسار بـــ 0's أعلى السطح العاكس، و بـــ 1's أسفله، فنحصل على

0 0 0
0 0 1
0 1 1
0 1 0
1 1 0
1 1 1
1 0 1
1 0 0

و بذلك تكون الشفرة الرمادية المكونة من ثلاثة خانات هي

000
001
011
010
110
111
101
100

شفرة رمادية من أربعة خانات:

لتوليد شفرة رمادية مكونة من أربعة خانات نبدأ بالشفرة الرمادية المكونة من ثلاثة خانات التي قمنا بتوليدها، أي

0 0 0
0 0 1
0 1 1
0 1 0
1 1 0
1 1 1
1 0 1
1 0 0

نقوم بإجراء عملية عكس (Reflection) لهذه الشفرة (على سطح عاكس وهمي) فنحصل على

0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0
1	0	0
1	0	1
1	1	1
1	1	0
0	1	0
0	1	1
0	0	1
0	0	0

أخيراً نقوم بملء الخانة الرابعة الواقعة إلى اليسار بـــ 0's أعلى السطح العاكس، و بـــ 1's أسفله، فنحصل على

0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	1	1
0	1	0	1
0	1	0	0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	1	0
1	0	1	1
1	0	0	1
1	0	0	0

تستخدم الشفرة الرمادية فى التطبيقات الصناعية التي تستخدم فيها الأنظمة الرقمية في التحكم فى الآلات. وسبب استخدام الشفرة الرمادية هنا هو عدم إمكانية ظهور قيم وسطية خاطئة.

الخلاصة:

عزيزي الدارس، ها نحن قد وصلنا إلي نهاية الوحدة الأولى، و التي تناولنا فيها طريقة تمثيل أنواع البيانات المختلفة في الحاسوب و بقية الأنظمة الرقمية. حيث تعرفنا أولاً على طريقة تمثيل الأعداد الصحيحة بأنواعها المختلفة، قصيرة (Short) أو عادية (Integer) أو طويلة (Long)، بإشارة (Signed) أو بدون إشارة (Unsigned). كما حددنا مدى القيم التي يقبلها كل نوع. و عرفنا أن الأعداد الصحيحة يتم تمثيلها بدقة كاملة دون أي خطأ. ثم انتقلنا إلى الأعداد الحقيقية و درسنا طريقة تمثيلها، و عرفنا أنه من غير الممكن تمثيلها بدقة كاملة نظراً لمحدودية المساحة المتاحة لتخزينها. و خلصنا إلى أنه كلما زاد حجم المساحة المتاحة لتخزين العدد، سواء كان عدد صحيح أو عدد حقيقي، فإننا نحصل على إمكانية التعامل مع قيم أكبر، كما نحصل علي دقة أفضل في حالة الأعداد الحقيقية. بعد ذلك تعرضنا لبعض أنواع الشفرات المستخدمة لتمثيل البيانات في الحاسوب، حيث وضحنا الخصائص العامة لكل شفرة.

لمحة مسبقة عن الوحدة التالية:

في الوحدة الثانية سنتعرف على أساسيات الجبر البولياني (Boolean Algebra)، و هو جبر المتغيرات المنطقية. و المتغيرات المنطقية هو نوع المتغيرات الذي يتم التعامل معه في الدوائر المنطقية (Logic Circuits)، التي تسمى أيضاً بالدوائر الرقمية (Digital Circuits). حيث نتعرف على المتغير المنطقي (Logical Variable)، و العمليات المنطقية الأساسية الثلاث NOT و AND و OR، و البوابات المنطقية (Logic Gates) التي تقوم بإجراء هذه العمليات، و هذه البوابات هي الوحدات الأساسية المستخدمة في بناء أي نظام رقمي. كما نتعرف على التعبير المنطقي (Logical Expression) و جدول الصواب (Truth Table). وفي نهاية الوحدة نتعرف على بعض نظريات الجبر البولياني و استخدامها في تبسيط التعبيرات المنطقية. و تبسيط التعبيرات المنطقية هي خطوة مهمة جداً في تصميم الدوائر المنطقية، و هو موضوع الوحدة الثالثة.

إجابات التدريبات

تدريب 1:

(1) $55 = (110111)_2$

(2) $200 = (11001000)_2$

(3) $300 = (100101100)_2$

(4) $600 = (1001011000)_2$

تدريب 2:

(1) $(10111011)_2 = 187$

(2) $(10000000)_2 = 128$

(3) $(11100111)_2 = 231$

(4) $(11111111)_2 = 255$

تدريب 3: لا نستخدم مدى من القيم، و بالتالي طولاً، أكثر مما نحتاج إليه توفيراً للمساحة التخزينية، حيث أنها محدودة دوماً في الأنظمة الرقمية.

تدريب 4:

(1) $+15 = (00001111)_2$

$-15 = (11110001)_2$

(2) $+65 = (01000001)_2$

$-65 = (10111111)_2$

(3) $+1 = (00000001)_2$

$-1 = (11111111)_2$

(4) $+127 = (01111111)_2$

$-127 = (10000001)_2$

تدريب 5:

(1) $(10111111)_2 = -65$

(2) $(10000000)_2 = -128$

(3) $(01111111)_2 = +127$

(4) $(11111111)_2 = -1$

تدريب 6: (أ) لا تحدث أي مشكلة.

(ب) يحدث Overflow لأن القيمة المراد تخزينها خارج المدى المحدد للعدد القصير الصحيح ذو الإشارة. فيتم تخزين القيمة 56- و ليس القيمة 200.

تدريب 7:

(1) $53.8125 = (110101.1101)_2$

(2) $0.1015625 = (0.0001101)_2$

(3) $0.7 = (0.10110011001100\mathbf{L})_2$

تدريب 8: :

(1) $53.8125 = (110101.1101)_2 = 0.1101011101\times 2^6$

(2) $0.1015625 = (0.0001101)_2 = 0.1101\times 2^{-3}$

(3) $0.7 = (0.10110011001100\mathbf{L})_2 = 0.10110011001100\mathbf{L}\times 2^0$

تدريب 9:

0	00000110	11010111010000000000000	(1)
1	11111101	11010000000000000000000	(2)
0	00000000	10110011001100110011001	(3)

تدريب 10: 7.15×10^{-8}

تدريب 11: **99**

المصادر و المراجع

1. M. Morris Mano, ***Digital Design***, Prentice-Hall, 2002
2. R. F. Tinder, ***Engineering Digital Design***, Academic Press, 2000
3. JohnWakerly, ***Digital Design: Principles and Practices***, Prentice-Hall, 2000
4. Daniel D. Gajski, ***Principles of Digital Design***, Prentice Hall, 1997
5. R. H. Katz, Benjamin/Cummings, ***Contemporary Logic Design***, 1994
6. J. P. Hayes, ***Introduction to Digital Logic Design***, Addison-Wesley, 1993