كلية الهندسة المعلوماتية
قسم النظم و الشبكات الحاسوبية
مشروع مادة نظم الزمن الحقيقي

برمجة البوابة التفرعية LPT في الحاسب الشخصي

بإشراف الدكتور : حسن الصّالح

إعداد الطالبات:

نسرين جبيل **عبير كلش** فجر باعوث

2005-1-9

الفهرس

1. مقدمة: 4

2. أنماط البوابات التفرعية : 4

3. ميزات استخدام البوابات التفرعية : 5

4. خطوط البوابة التفرعية : 6

5. الطريقة الأولى : الوصول إلى البوابة التفرعية من خلال توابع BIOS : 8

5.1 خطأ انقضاء الزمن المحدد : 9

5.2 توابع مقاطعة الطابعة : 9

5.3 تابع كتابة المحارف: 10

5.4 تابع تبدئة البوابة التفرعية : 10

5.5 تابع قراءة بايت الحالة : 10

6. الطريقة الثانية : البرمجة المباشرة للبوابة التفرعية : 10

6.1 عناوين البوابات : 10

6.2 استخدام الكبل التفرعي العادي باتجاهين : 12

6.3 نقل البيانات عبر الكبل التفرعي وآليته على مستوى البايت بدفعتين : 13

6.4 مشاكل انقضاء الزمن : 15

6.5 التزامن: 16

6.6 إيقاف البرنامج : 16

7. بروتوكول مستوى البلوك BLOCK LEVEL : 17

7.1 قراءة حالة مفتاح Esc : 17

7.2 المستويات الأعلى : 18

7.3 كيفية إدخال بايت كامل: 18

8. الربط بين الحواسيب الشخصية عن طريق البوابات التفرعية : 20

8.1 الخيارات المتاحة لتنفيذ بوابة دخل بثمانية بتات : 21

8.2. بوابة الدخل : 22
8.3. بوابة الدخل بخطي انتخاب :..... 24
8.4. استخدام البتات D4 –D7 : 26
8.5. البتات 3,4,5,7 :..... 28
8.6. زوج من المداخل : 31
8.7. مسك البايتات :..... 33
8.8. زوج من المخارج : 36
9. إرسال واستقبال الملفات وبروتوكول التراسل بين المرسل و المستقبل:..... 38
10. البرنامج التطبيقي لنقل ملف ما على البوابة التفرعية :..... 40

1. مقدمة:

توجد ثلاثة طرق متاحة للوصول إلى البوابات التفرعية :
البرمجة المباشرة للهاردوير , خلال توابع ال Bios أو بواسطة استدعاء توابع DOS , وإن استخدام توابع Bios يقدم ميزات أكثر من توابع DOS لأنها تسمح بتحكم أفضل بحالة الطابعة , أما توابع DOS فإنها تفشل فوراً في حال أن الطابعة تحدث مقاطعة الخطأ الحرجة , في حين أن ال Bios يقدم أعمالاً أخرى .

توجد ثلاثة أنواع لبوابات الحواسيب الشخصية القياسية , والتي تستخدم من أجل وصل الدارات الإلكترونية الإضافية وهي:
البوابة التسلسلية Serial port .
البوابة التفرعية Parallel/printer port .
البوابة التشابهية أو بوابة الألعاب game/analogue port .
تعتبر البوابات التفرعية مفيدة في كثير من التطبيقات والحالات , حيث يتم تزويد جميع الحواسيب الشخصية ببوابة تفرعية واحدة على الأقل , ولكن في حال وجود طابعة موصولة للبوابة التفرعية وعدم وجود بوابة بديلة أخرى , فيمكن عندها تزويد الحاسب بوابة تفرعية ثانية وبكلفة زهيدة وذلك عن طريق إضافة بطاقة خاصة إلى فتحات التوسعة المتاحة , لذلك هذا يعني أن مسألة توفير بوابة تفرعية في الحواسيب الشخصية لأغراض الوصل مع الدارات الإلكترونية الخارجية لا يشكل أدنى مشكلة .
و على الرغم من أن البوابة التفرعية قد تبدو محدودة الأستخدام للوهلة الأولى , ولكنها في الواقع متعددة الإستخدامات , وتعتبر البوابة التفرعية بوابة من أجل الخرج الرقمي (DigitalOutput) ,ولكن لحسن الحظ بالإضافة إلى البتات الثمانية المتوفرة كمخارج رقمية فإنها تمتلك العديد من خطوط المصافحة (handshake lines) , والتب لا يقل عددها عن 9 خطوط منها :خمسة مداخل و أربعة مخارج . ومن الممكن استخدام خطوط المصافحة هذه لأغراض الإدخال والإخراج الرقمي العامة .
إن تشكيل بوابة خرج رقمي ثمانية البتات , وبوبة دخل رقمي ثمانية البتات أيضاً باستخدام البوابة التفرعية سيكون أسهل بكثير من الحصول على نفس النتيجة باستخدام البوابة التسلسلية أو فتحات ممر التوسعة . أن استخدام البوابة التفرعية سيمكن من القراءة و الكتابة بمعدلات أسرع بكثير مما يمكن أن تقدمه البوابة التسلسلية . ولكن ستبقى بعض خطوط المصافحة غير مستخدمة ويمكن استغلالها في الدارات الألكترونية المصممة , حيث يمكن استخدامها لتوفير 8 بتات أخرى من المخارج و المداخل الرقمية .
تمكننا البوابة التفرعية من الحصول على عدد كبي من المداخل والمخارج الرقمية إذا ما استخدمنا تقنية الإنتخاب (multiplexing technique) , ولكن يجب الإنتباه أنه إذا ما تجاوز عدد خطوط الدخل و الخرج الرقمي حداً ما فإنه من المفضل اسخدام فتحات التوسعة .

2. أنماط البوابات التفرعية :

1- SSP (original) :
وهي البوابة التفرعية الأساسية و كذلك النمط القياسي للبوابا ت التفرعية , وفيه تستطيع البوابة إرسال 8 بتات على التوازي و استقبال 5 بتات فقط على بوابة state باعتبار data للإخراج فقط , و بالتالي فلتبادل معطيات 8 بت سنقسم البايت على دفعتين و نرسلهم 4 بت ثم 4 بت .

2- ps/2-type (simple bidirectionnal):

و فيه يصبح مسجل المعطيات للإدخال و الإخراج , ولكن لكل حالة من الحالات (2 حتى 7) , ويوجد مسجلات إضافية داخلية للتحكم .

3- EPP (Enhanced parallel port) :

إن هذا النمط يشبه النمط السابق من حيث مسجل ال data ثنائي الإتجاه , ولكنه يمتاز عن سابقه بسرعته التي تعادل 4 أضعاف سرعة سابقه , ويمتاز أيضاً بسرعة التبديل م ندخل إلى خرج وبالعكس , وبالتالي فإن استخدامه ضروري عند التعامل مع وحدات التخزين الكبيرة .

4- ECP (extende capabilities port) :

يشبه سابقيه (2و3) بالإتجاه الثنائي و يتميز بامتلاكه buffer تجعله مناسباً لنقل المعطيات بالطريقة DMA (direct memory access) , ويستعمل عادة من أجل الطابعات حيث الإرسال و الإستقبال سريع جداً , وكذلك الماسحة الضوئية .

ملاحظة هامة :

يمكن إعداد البوابة التفرعية (ضبط الإعدادات) في بطاقات اللوحات الأم الحديثة لتكون من أي من الأنماط الأربعة السابقة . أي عند التحويل من النمط 4 إلى النمط 3 فإننا نلغي ال buffer . أما عند التحويل من النمط 3 إلى النمط 2 فإننا نلغي عامل السرعة إذا كان switch .

3. ميزات استخدام البوابات التفرعية :

يداخلنا أحياناً سؤال مهم وهو لماذا يفضل استخدام البوابة التفرعية و بوابة الألعاب بالرغم من وجود ممرات التوسعة ؟

يمكن القول بأنه توجد بعض المصاعب العملية في استخدام ممرات التوسعة , مما يمنع استخدامها بشكل واسع من قبل الإلكترونيين , فإن البوابة التفرعية تشكل خياراً سهلاً و مناسباً ولهذا يبقى مصممها بعيداً عن التعرض لمصاعب فعلية في أثناء التنفيذ .

تتم عملية وصل الدارة الإلكترونية المصممة بالحاسب بسهولة باستخدام موصل من النوع D . لذلك فإن سهولة تصميم و تنفيذ دارات المواجهة مع الحاسب عن طريق البوابة التفرعية هو ما يشكل ميزة كافية لاستخدام هذه البوابة و الإستغناء عن فتحات التوسعة في معظم الحالات .

و من الميزات الأخرى التي تدفع لاستخدام بوابا ت الحاسب القياسية بدلاً من فتحات التوسعة هي كون هذه البوابات توفر لنا بعض الكيان الصلب (Hardware) الذي قد نستغله في الدارات الإلكترونية , بينما في حالة ممرات التوسعة فلابد لنا من وضع دارات فك عنونة (address decoding) , وبوابات الدخل و الخرج (Input /Output ports) , وهذا يعني أننا نستخدم العديد من الدارات المتكاملة قبل بداية المشروع نفسه , ولكن استخدام البوابات القياسية يسمح لنا بالإستغناء عن دارات المواجهة المساعدة .

ولكن بالمقابل فإنه توجد ميزة أخرى لإستخدام ممرات التوسعة : وهي أن الدارة في النهاية ستكون موجودة داخل الحاسب و لن نحتاج إلى كتلة إضافية بجانبه , كماهو الحال عند استخدام البوابات القياسية .

إن تطوير البطاقات الموصولة مع الحاسب عبر ممرات التوسعة سيتطلب فك الحاسب , و إجراء التجريب داخله , ا لأمر الذي قد يسبب بعض المشاكل.

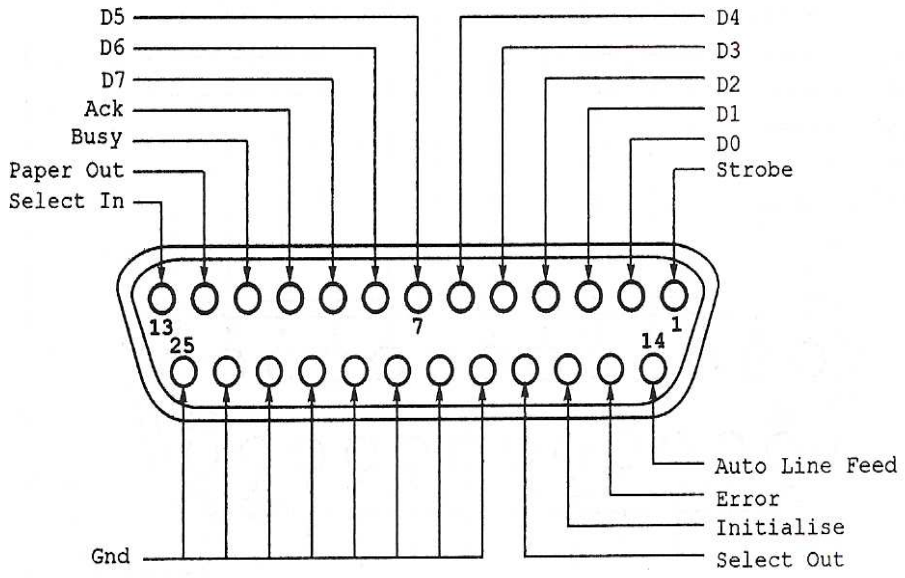
ولكن توجد مشكلتين أساسيتين عند استخدام بوابات الحاسب القياسية :

عدم توفر خطوط في البوابة التفرعية , بينما يتوفر خط التغذية 5 فولط في بوابة الألعاب فقط , وعند استخدام البوابة التفرعية يمكن استخدام خط التغذية 5 فولط من بوابة الألعاب . كما يمكن

استخدام التغذية من ممرات التوسعة , إلا إن هذه الحلول ليست مناسبة بالقدر الكافي . مما يدفع أحياناً إلى وضع دارات تغذية خارجية .
عندما يزداد تعقيد دارة المواجهة المصممة يفضل استخدام ممرا ت التوسعة لأنها في هذه الحالة تشكل الحل النسب .

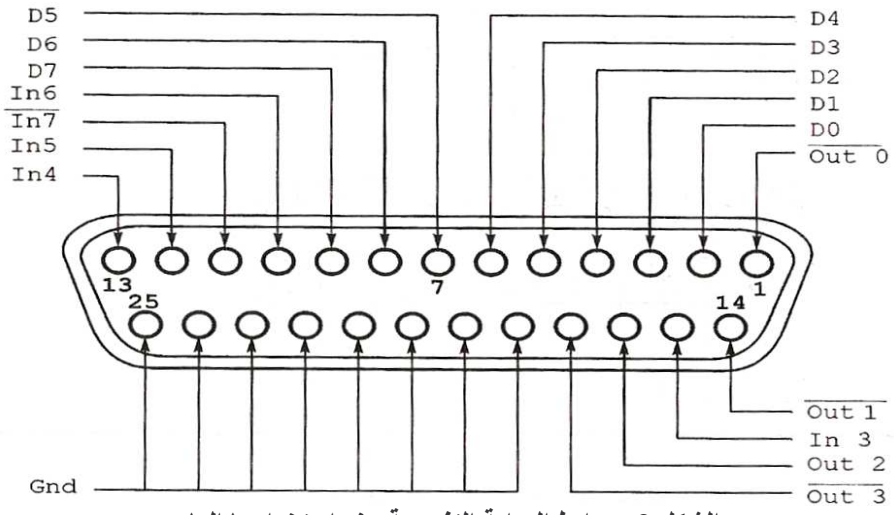
4. خطوط البوابة التفرعية :

البوابة التفرعية في الحاسب عبارة عن وصل مؤنث مؤلف من 25 خطاً من النوع D (female 25 – way D type connector) , لذلك نحتاج إلى موصل مذكر مؤلف من 25 خطاً من النوع D (male 25 – way D type connector) من اجل وصل الدارة مع البوابة التفرعية .
عندما تستخدم البوابة التفرعية لوصل الطابعة فإن خطوط الموصل سيكون لها الشكل التالي, وهو موصل الطابعة عندما ينظر إليه من خارج الحاسب باتجاه الداخل :



الشكل 1: مرابط بوابة الطابعة

إن الشكل السابق يظهر خطوط الموصل عندما ينظر إليه من الخلف , أي عندما نمسك به لنوصله إلى الحاسب .
إن التحكم المباشر بالبوابة التفرعية يسمح بدرجة كبيرة من الحرية , و بمساعدة بعض الدارات المتكاملة الخارجية يمكن الحصول على العديد من خطوط الدخل و الخرج و المصافحة . وحتى في الحالة البسيطة التي نستخدم فيها البوابة كمخرج رقمي ثماني البتات , فقد يكون من الأسهل التحكم مباشرة بالبوابة عوضاً عن استخدام برامج نظام التشغيل أو البرامج الأخرى .
إن التحكم المباشر بالبوابة التفرعية سهل , ولا تعترضه أية صعوبة , وعند اعتماد منحى التحكم المباشر للبوابة التفرعية فيجب ان يكون هذا بشكل عام بغض النظر عن كونها منفذ للطابعة , وفي هذه الحالة تكون الوظائف و الأسماء المحددة لخطوط الطابعة هي المبينة بالشكل:



الشكل 2: مرابط البوابة التفرعية عند استخدامها العام

حيث أنه من D0 حتى D7 : هي ثمانية خطوط خرج تمسك القيم المكتوبة فيها (Latching output data) , وعلى هذه الخطوط تكتب المعطيات عندما ترسل إلى الطابعة .
أما في الحالة العامة , وعند التحكم المباشر بالبوابة تستخدم هذه الخطوط كمخرج رقمي , وذلك بكتابة قيمة ما مؤلفة من بايت واحد على العنوان المحدد لهذه البوابة .
أما بقية الخطوط فتؤخذ على أنها خطوط دخل أو خرج رقمي تتم قراءتها أو الكتابة فيها باستخدام عناوين دخل أو خرج محددة .

يمكن تمثيل البوابة التفرعية على ثلاثة مسجلات :

1-) Data Register : d0 d1 d2 d3 d4 d5 d6 d7
وهو يستخدم قديماً للإخراج فقط , أما في اللوحات الجديدة فيستخدم في الإدخال و الإخراج .

2-) State Register :S0 S1 S2 S3 S4 S5 S6 S7’
حيث أن البتات S0 S1 S2 غير مستخدمة , وليس لبعض البتات مقابلات على الأرجل , وإن هذا المسجل هو مسجل للإدخال فقط , فهو يدل على الحالة .

3-) Control Register : C0’ C1’ C2 C3’ C4 C5 C6 C7
وهذا المسجل لإرسال إشارات التحكم , حيث "أن البتات C4 C5 C6 C7 غير مستخدمة , و هو يستخدم سابقاً كخرج فقط , و إنما في الللوحات الجديدة يمكن استخدامه كدخل أيضاً .

وأن التمثيل السابق ليس تمثيل عشوائي , وإنما يوافق إشارات الطابعة .
ولسوء الحظ فإن بعض خطوط الدخل و الخرج الإضافية تحتوي على عاكس (inverter) داخلي ,وهي المشار إليها بخط فوق الإسم .
إن وجود عاكس على بعض الخطوط يعتبر أمراً غير مرغوب فيه , ولكنه لا يشكل عائقاً كبيراً , إذ أنه يمكن عكس هذه الخطوط باستخدام عاكس خارجي أو إبقاء هذه الخطوط على ما هي عليه , مع الأخذ بعين الإعتبار أنها معكوسة في البرامج التي نكتبها للتحكم بالبوابة .

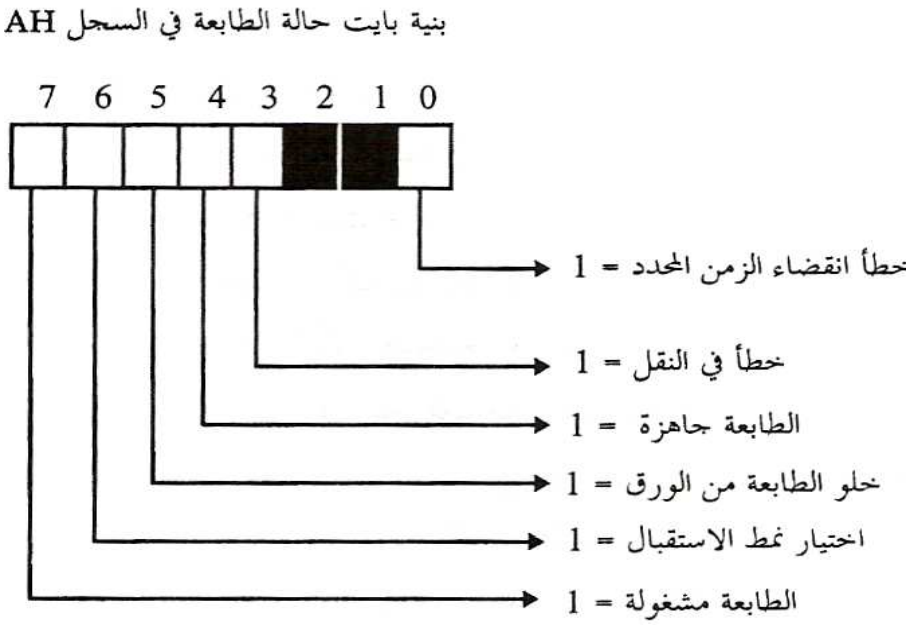
5. الطريقة الأولى : الوصول إلى البوابة التفرعية من خلال توابع Bios :

يقوم Bios بحجز المقاطعة 17H بشكل خاص لإستخدام البوابة التفرعية , و لذلك يطلق على هذه المقاطعة اسم مقاطعة الطابعة (printer interrupt) . تتضمن هذه المقاطعة ثلاثة توابع لأداء مهمات مختلفة للبوابات التفرعية . تتفوق هذه التوابع في مقدرتها على التحكم بالبوابات التفرعية الثلاث التي من الممكن أن تتواجد داخل الحاسب الشخصي , وهذه التوابع معاكسة لتوابع DOS التي لا تستطيع التحكم إلا بالبوابة التفرعية الأولى LPT1 . وتتم عنونة البوابة التفرعية المقصودة بإجراء المقاطعة 17H من خلال وضع أرقام معينة في المسجل DX للدلالة على البوابة التفرعية المطلوبة , وهذه الأرقام هي 0,1,2 وتوضع في المسجل DX و تقابل البوابات LPT3, LPT2, LPT1 على التسلسل .

Function	Task
00H	Display Characters
01H	Initialize Printer
02H	Request Printer status

وتمتلك هذه التوابع شيئاً آخر مشترك وهو أنه بالإضافة إلى وجوب تمرير رقم البوبة لها و لكن كذلك إنها جميعها تخزن حالة الطابعة في المسجل AH بعد أن يتم استدعاؤها . تبين حالة الطابعة عدة أمور تتعلق بحالتها وجاهزيتها للطباعة أو عدم جاهزيتها , فتبين لنا البتات السابقة مثلاً فيما إذا كانت مشغولة أم لا , وهل فرغت من الورق أم لا , كما تبين فيما إذا حدث خطأ خلال استقبالها للمحارف أم لا .

و الشكل التالي يبين و يوضح ما سبق حيث أنه المسجل AH مع معاني بتاته , حيث يبين الشكل توضع البتات على المسجلAH بالإضافة إلى معنى وعمل كل بت و نلاحظ أن البتين 1,2 غير مستخدمين حالياً :



الشكل3 : معاني بتات حالة الطابعة

فيجب إذاً التأكد من حالة الطابعة قبل القيام بأي عملية , أي نتأكد من وجود الورق فيها , أو إذا كانت مطفأة أو شغالة , و كذلك إذا كانت موصولة إلى البوابة التفرعية أو لا . ويتم هذا بواسطة بايت الحالة التابع للطابعة , حيث إذا كان بت معين مساوياً إلى الواحد فهذا يعني أن الشرط المقابل له محقق , وإلا فإنه يكون غير محقق و مساوياً إلى الصفر .
إذا في حالة عدم تحقق شروط معينة أو تحقق أمور معينة (مثل خلو الطابعة من الورق) فلن نستطيع إتمام عملية إرسال المحارف إلى الطابعة :
و البرنامج المصغر التالي يوضح كيف يتم هذا :

```
Pstatus=printerstatus;
If( (( pstatus and 29h )<>0) or
   (  ( pstatus and 80h )=0) or
   ( ( pstatus and 10h )=0))  then
      printerok=false
          else
      printerok=true;
```

5.1. خطأ انقضاء الزمن المحدد :

يحدث هذا الخطأ إذا حاولت توابع Bios إرسال المحارف إلى الطابعة عدداً محدداً من المرات دون أن تنجح في ذلك . يتم تحديد هذا العدد من خلال عداد موجود في الذاكرة . والجدول التالي يوضح مواقع عدادات البوابات التفرعية الموجودة في الذاكرة :

العنوان	المحتوى
0040:4478	عداد انقضاء الزمن المحدد للبوابة التفرعية الاولى
0040:4479	عداد انقضاء الزمن المحدد للبوابة التفرعية الثانية
0040:007A	عداد انقضاء الزمن المحدد للبوابة التفرعية الثالثة

وعادة ما يقوم Bios بوضع القيمة 20 في هذه العدادات عند الإقلاع . ويتم عندئذٍ تكرارا محاولة إرسال المحارف 262140*20 مرة .نلاحظ أن الزمن المقابل لهذا العداد من المحاولات يختلف من حاسب لآخر وذلك حسب سرعته . ويستطيع المبرمج الوصول إلى هذه العدادات لأنها موجودة في ذواكر RAM , وعندها يستطيع المبرمج تغيير قيمتها.

5.2. توابع مقاطعة الطابعة :

نستطيع تنفيذ المقاطعات البرمجية كالمقاطعة 17H إما بلغة التجميع أو من خلال التوابع الموجودة في معظم لغات البرمجة العالية المستوى .
بالنسبة إلى لغات التجميع فإنه يتم تحميل المسجلات بالقيم المناسبة ثم يتم تنفيذ تعليمة المقاطعة INT فيمكن ان نكتب على سبيل المثال التعليمةINT 17H , و يتم بعد ذلك قراءة النتائج من المسجلات المناسبة . أما في اللغات العالية المستوى فيتم استخدام توابع متعددة فمثلاً في لغة Turbo Pascal يتم استخدام أحد التابعين Inter أو MsDos الموجودين في الوحدة Dos , أما بلغة C مثلاً فيتم تنفيذ المقاطعات البرمجية بواسطة أحد التوابع التالية :
intdos (), in86() , int86X() أو intdosx()

5.3. تابع كتابة المحارف:

يقوم هذا التابع بكتابة محرف ما في البوابة التفرعية , ثم يتم تحديد هذا التابع بوضع القيمة 00H في المسجل AH , ويتم وضع الترميز ASCII للمحرف الذي نريد كتابته في المسجل AL , ويتم أخيراً تنفيذ المقاطعة 17H . وبعد تنفيذ هذا التابع سيحتوي المسجل AH على بايت حالة الطابعة .

5.4. تابع تبدئة البوابة التفرعية :

و يقوم هذا التابع بتهيئة البوابة التفرعية و الطابعة إذا كانت موصولة معها . ويتم عادةً استدعاء هذا التابع قبل البدء بإرسال المحارف إلى الطابعة . و يتم هذا التابع بوضع القيمة 10H في المسجل AH , ويتم بعج ذلك تنفيذ المقاطعة 17H . وبعد تنفيذ هذا التابع سيحتوي المسجل AH على بايت حالة الطابعة .

5.5. تابع قراءة بايت الحالة :

ويقوم هذا التابع بقراءة بايت الحالة للطابعة , ويتم تحديد هذا التابع بوضع القيمة 02H في المسجل AH , ثم يتم تنفيذ المقاطعة 17H . وبعد تنفيذ هذا التابع سيتم وضع بايت الحالة المقروء للطابعة في المسجل AH .

6. الطريقة الثانية : البرمجة المباشرة للبوابة التفرعية :

6.1. عناوين البوابات :

إن نظام التشغيل Dos يسمح انا بأن نتتعامل مع بوابتين تفرعيتين , وهما LPT1,LPT2 .
وتشغل كل من هذه البوابات ثلاثة عناوين في خريطة عناوين الدخل والخرج لذلك فإن الكتابة و القراءة من العناوين الخاصة بهذه البوابات يتم بالتعليمتين INP,OUT بلغة BASIC أما بلغة الإسمبلي فتتم بالتعليمتين IN,OUT .
والجداول التالية توضح و تلخص عناوين هذه البوابات :
أولاً : بالنسبة إلى (***LPT1***) :

H378 &		&H379		&H37A	
اسم الخط	رقم البت	اسم الخط	رقم البت	اسم الخط	رقم البت
D0	0	غيرمستخدم	0	معكوس out0	0
D1	1	غيرمستخدم	1	معكوس out1	1
D2	2	غيرمستخدم	2	out 2	2
D3	3	In3	3	معكوس out3	3

4	غيرمستخدم	4	In4	4	D4
5	غيرمستخدم	5	In5	5	D5
6	غيرمستخدم	6	In6	6	D6
7	غيرمستخدم	7	معكوسIn7	7	D7

ثانياً : بالنسبة إلى (***LPT***2) :

&H27A		&H279		H278 &	
رقم البت	اسم الخط	رقم البت	اسم الخط	رقم البت	اسم الخط
0	معكوس out0	0	غيرمستخدم	0	D0
1	معكوس out1	1	غيرمستخدم	1	D1
2	out2	2	غيرمستخدم	2	D2
3	معكوس out3	3	In3	3	D3
4	غيرمستخدم	4	In4	4	D4
5	غيرمستخدم	5	In5	5	D5
6	غيرمستخدم	6	In6	6	D6
7	غيرمستخدم	7	معكوسIn7	7	D7

أن كتابة قيمة ما على خطوط المعطيات D0-D7 لا نحتاج إلى أكثر من إخراج هذه القيمة على العنوان المناسب , وكمثال على هذا إذا أردنا أن نجعل D0-D7 تأخذ القيمة واحد منطقي أي (5) فولط للبوابة LPT2 فيكفي عندها أن نكتب القيمة &HFF أي 255 على العنوان H278 &

في اللغات العالية المستوى مثل BASIC فإن ذلك يتم بواسطة التعليمة التالية :

OUT &H278 , 255

وهنا من المريح و الجيد أن المعطيات سوف تبقى ثابتة بعد كتابتها بسبب وجود ماسك داخلي , وهنا لا حاجة لوضع ماسك (latch) في الدارة الإلكترونية .

ومن الجدير بالذكر أنه على الرغم من أن خطوط المعطيات D0-D7 يمكن أن تكون ثنائية الإتجاه و لكن هذا لا يعني أنها تستعمل كدخل و كخرج في نفس الوقت في البوابات القياسية , فمثلاً بوابات الطابعة القياسية لا يمكن أن تكون إلا خرجاً ,أي نستطيع استخدامها كمخرج للمعطيات الرقمية فقط . يمكن إدخال المعطيات عن طريق البوابة التفرعية باستخدام خطوط الدخل الخمسة IN3-IN7 , ونستطيع الحصول على دخل بعرض ثمانية بتات بمساعدة بعض الدارات المتكاملة .

و كذلك الحال بالنسبة إلى خطوط الخرج للمعطيات D0-D7 , فالخطوط المسماة Out0-Out3 على العناوين &H37A للبوابة LPT1 , و &H27A للبوابة LPT2 هي خطوط خرج حصراً وتبقى القيم المكتوبة ممسوكةً عليها .

إن التعامل مع الخطوط out و In كخطوط للمصافحة سيكون أسهل بكثير فيما إذا استدعنا التعامل مع كل خط من هذه الخطوط بشكل مستقل و منفصل عن بقية الخطوط , ولكن هذا غير ممكن , إذ إن جميع الخطوط out0- out3 لها نفس العنوان , و بالتالي لا يمكن التعامل معها إلا

ككتلة واحدة , وكذلك الأمر بالنسبة لخطوط In3-In7 . لذلك يجب الحذر عند التعامل مع هذه الخطوط .
وعندما نريد تغيير حالة خط واحد من هذه الخطوط يجب أن ننتبه إلى عدم تغيير بقية الخطوط خطأً . ولإجراء تغيير في حالة الخطوط out0--out3 يجب تخزين القيمة المكتوبة على هذه الخطوط بمتحول ضمن البرنامج .
بجب الانتباه إلى أن الخطوط out0---out3 موجودة على البتات الأربعة الدنيا , وأن الخطوط out0-out1-out3 معكوسة , وإن اتصال هذه الخطوط بالبتات الأربعة الدنيا يعني أن القيم التي يمكن أن تأخذها تتراوح بين الصفر و ال15 , وإن أية قيمة فوق 15 لن تكون إلا تكراراً لإحدى القيم من 0 حتى 15 , فمثلاً القيمة 16 تكافئ الصفر , والقيمة 17 تكافئ 1 وهكذا .....
يمكن أن نفهم ذلك بسهولة بكتابة العدد بالشكل الثنائي , فمثلاً يكتب ا لعدد 5 بالشكل الثنائي على بايت واحد كمايلي :5=xxxx0101 , ونلاحظ أن البتات الأربعة العليا مهما كانت قيمتها فلن تغير في ا لأمر شيئاً لأنه لا تأثير لها على الخطوط out0- out3 .
ونلاحظ أن كون البتات out0- out1- out3 معكوسة فهذا يعني أن البت المكتوب على أحد هذه الخطوط سيظهر معكوساً .
و ا لأمر نفسه ينطبق على بتات الدخل In3-In7 , فمثلاً إذا تمت قراءة القيمة , 37=0010 0101 , أو القيمة 32=0010 0000 , فإن هذا يعكس نفس الحالة للخطوط لأن البتات الثلاثة الدنيا لا تأثير لها , وتكون في هذه الحالة القيم الفعلية للخطوط هي :

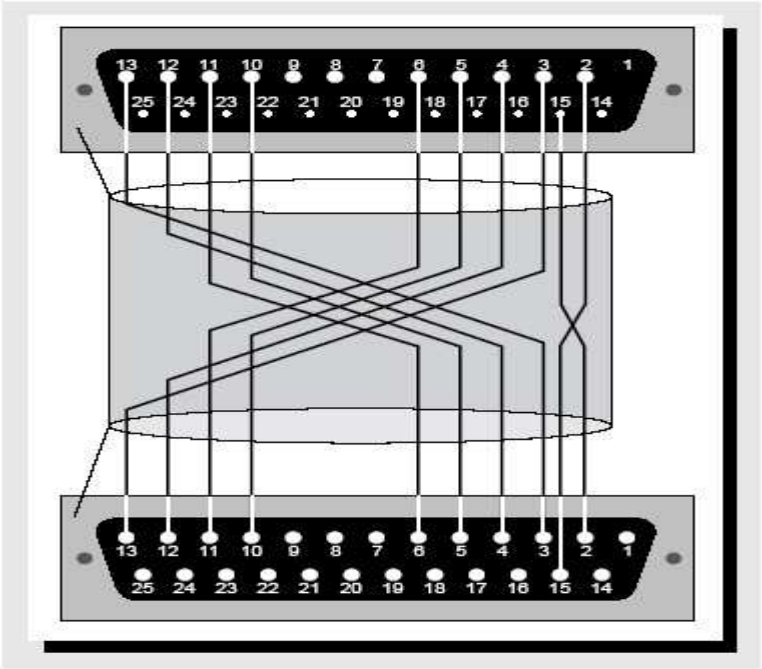
In7	In6	In5	In4	In3
1	0	1	0	0

و نلاحظ أن قيمة البت السابع In7 الفعلية هي 1 وليس 0 , وذلك بسبب وجود العاكس الداخلي.

6.2. استخدام الكبل التفرعي العادي باتجاهين :

إذا احتجت أن تستعمل بوابتك التفرعية بشكل خاطئ (معاكس) لنقل المعلومات بين حاسبين عندها الكبل التفرعي العادي لن يعمل . هذه المشكلة تأتي من كون البوابات التفرعية تملك و صلات متماثلة على كلا الحاسبين لذلك نهاية ما من الكبل التفرعي لن تدخل في الحاسب الثاني .
هناك مشكلة ثانية في الكبل التفرعي العادي و هي أن نقل المعلومات يتم في اتجاه واحد فقط . أي أن حاسب ما يستطيع استخدام خطوط المعطيات من D0 و حتى D7 لإرسال المعطيات , لكن لا يستطيع استخدام نفس الخطوط لاستقبال المعطيات عبرها كما أنه لا يستطيع أي حاسب آخر استعمال تلك الخطوط .
عادة نقل المعطيات بين حاسبين يتطلب وصلات ثنائية الاتجاه. كمثال على ذلك , المستقبل سيعيد نتيجة مجموع المعطيات التي قام باستقبالها لذلك المرسل سيعرف على كل الأحوال إن كان قد تم 0
خطوط الحالة استخدمت من قبل الطابعة لتعيد معلومات الحالة إلى الحاسب الذي يمكن أن يمد بالحل .هذه هي الأخطاء , ACK , PE , SCLT وخطوط BUSY . تلك التي تترافق مع مسجل البوابة التفرعية الثانية . هذه الخطوط يتم وصلها إلى خطوط المعطيات من D0 و حتى D4 و هذا يعني أن المستقبل يقرأ الخرج من المرسل عبر خطوط الحالة و التي تكون مفهرسة بشكل واضح . بشكل معاكس , خطوط المعطيات D0 و حتى D4 الخاصة بالمستقبل يتم وصلها لخطوط الحالة الخاصة بالمرسل حيث يمكنها أن تقوم بنقل المعلومات باتجاهين .
لذلك بشكل أساسي نقوم بمقاطعة خطوط المعطيات مع خطوط الحالة . هذه القواعد تطبق على كل من المرسل و المستقبل . أي معطيات مرسلة بواسطة الخانات الخمسة الأولى من مسجل البوابة التفرعية الأولى سيتم استقبالها من قبل الخانات الخمسة الأخيرة في المسجل الثاني

الخاص بالشريك المتصل الآخر . وهذا لا يهم أي نهاية من الكبل سيتم وصلها إلى المرسل و أي نهاية سيتم وصلها إلى المستقبل .
الرسم التوضيحي التالي يظهر أي الأرجل من كل نهاية سيتم وصلها لتشكيل النقل التفرعي :

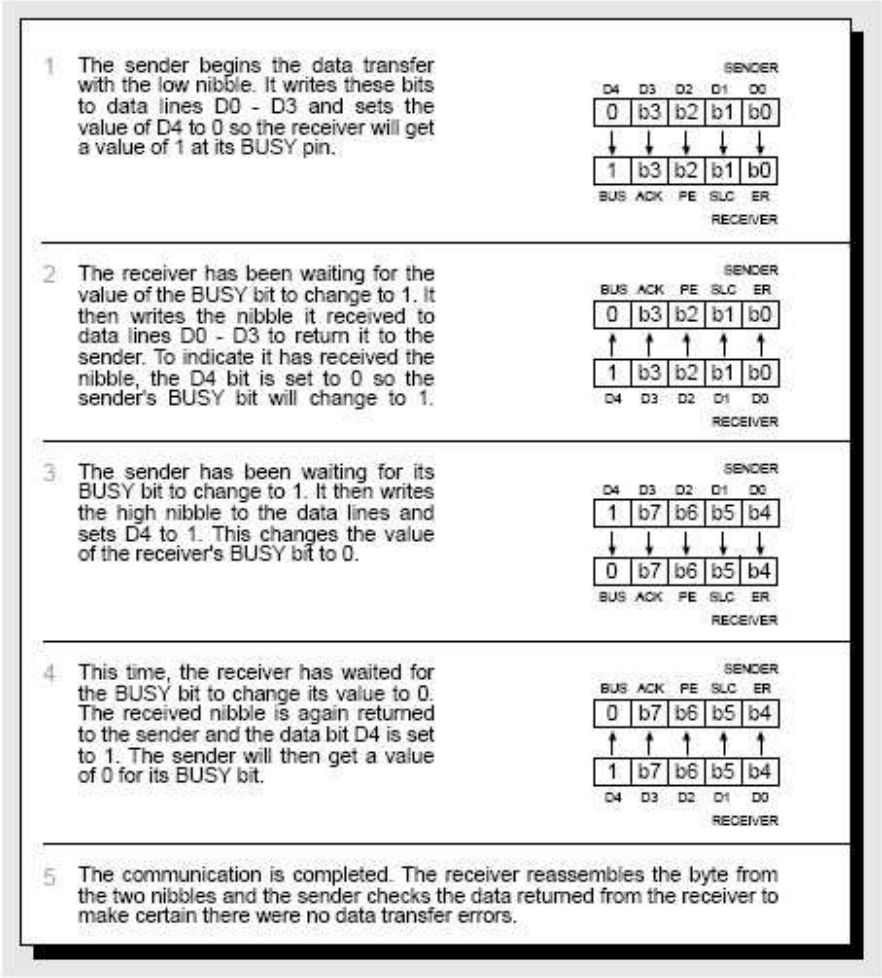


سنحتاج إلى وصلتين مذكرتين من نوع DB-25 و الكابلات التفرعية الأطول من هذا تسبب مشاكل في نقل البيانات .
هذا النوع من الكابلات يمكن أن يستخدم مع برامج نقل البيانات التي لها علاقة بالتجارة . و هذه البرامج تعمل عادة مع النوع نفسه من الكابلات . و إذا كان الكبل لا يعمل فيجب أن نقوم بفحص الأرجل , و هذا أيضاً ممكن حيث أن البرنامج يفترض أن المعطيات و خطوط الحالة تتصل بترتيب مختلف .
كبل النقل التفرعي يمكن أن يستخدم لوصل جهازين ونقل المعطيات بينها و يمكن أيضاً أن يستخدم هذا النوع من الكابلات للتحكم بجهاز Client من قبل جهاز Server .

6.3. نقل البيانات عبر الكبل التفرعي وآليته على مستوى البايت بدفعتين :

باستخدام الكبل التفرعي الذي تم وصفه سابقاً يمكن إرسال خمس خانات بشكل تزامني عبر الخطوط من D0 و حتى D4 . يمكن إرسال المعطيات في كلا الاتجاهين و بشكل تزامني .
و في بعض الأحيان نحتاج إلى ما يشابه خط Strobe و ذلك لحفظ أثر تقدم نقل البيانات . واحد من خطوط البيانات الخمسة يجب أن يستخدم لهذا الهدف .
يتألف البايت من ثمانية بتات و نحن لا نستطيع إرسال إلا أربع بتات في اتجاه واحد في نفس الوقت . لذلك يتم تقسيم كل بايت إلى قسمين يسمى كل منهما Nibble و كل مرة يرسل Nibble واحد فقط .
أولاً , يكتب الNibble الأول على البتات 0 حتى 3 لمسجل البوابة التفرعية الأول . و يتم إرسالها خارجاً على خطوط المعطيات D0-D3 بينما يهيئ البت الخارج على D4 بالقيمة 0

فيستقبل المستقبل القيمة 1 على BUSY . و هذا يخبر المستقبل بأن الNibble الأول من البايت التالي جاهز للقراءة , و هذا يعني أن المستقبل ينتظر و يقرأ خط الحالة حتى يحتوي البت BUSY على القيمة 1 .
بعدئذ لن يكون البت BUSY كثر إفادة لأن المستقبل يباشر بواسطة قراءة الNibble من البتات الموافقة في مسجل البوابة الثانية . تخزن محتويات تلك البتات الأربعة و يعاد إرسالها إلى المسل عبر خطوط المعطيات . يهيئ البت الخارج على D4 بالقيمة 0 لذلك يعاد استقبال القيمة 1 على الطرف الآخر و هذا يشير أن ال Nibble المرجع تم قراءته و حفظه .
وعندما يعود كلا الNibbles بنفس الطريقة , يستطيع المرسل تحديد فيما إذا تم نقل البايت بدقة ,إذاً يتحقق الاتصال على مستوى البايت ,بكل الأحوال هذه ليست الإجرائية المعتادة . بما أنه يأخذ وقت طويل لفحص كل بايت بشكل مستقل فإن هذه الطريقة من التحقق تنجز فقط في مستوى البلوك . أما إذا استخدم خط BUSY لإرسال إشارة STROBE معادة للمرسل لكل Nibble فإن ذلك لن يأخذ أي وقت لإرجاع ال Nibble الصحيح .و يتم إرسال ال Nibble الثاني بنفس الأسلوب ما عدا أن قيمة بت الحالة تغير إلى 1 و عندها سيستقبل المستقبل القيمة 1 على خط BUSY .
يعود بعدئذ الأربع بتات إلى المرسل و يصفر بت المشغولية للمرسل BUSY = 0 . بعدئذ يمكن لزوج الأربع بتات أن يندمجوا لتشكيل البايت الواحد .و يكتمل النقل من نقطة الاستقبال .يدمج المرسل الزوج الأربع بتات المعادة من المستقبل و يفحص البايت الكامل لمعرفة الأخطاء . و تنبه إجرائيات برنامج الإرسال العملية المستدعية من أجل وقوع أي خطأ حتى يقوم بالعمل الموافق و يمكن إعادة إرسال بلوك المعطيات إذا كان ضرورياً ذلك . يمكن أن تكشف معظم أخطاء نقل البيانات بهذه الطريقة: تتضمن الأخطاء أنماط غير مألوفة من تشويش الكابلات .
كما رأينا في الاتصالات الطبيعية للطابعة ,يتطلب الاتصال الصحيح بين حاسبين التبادل المناسب لإشارة STROBEعلى خط BUSY . و تذكر أنه لأن الخطوط متصالبة في الكابلات فإن نتعامل مع خطين منفصلين لBUSY . و يطبق نفس الشيء لكلا المرسل و المستقبل : في الخرج إن بت BUSY هو بت المعطيات الرابع D4 . أما تستقبل الإشارة على خط حالة BUSY في الطرف الآخر .



يستخدم كلاً من المرسل و المستقبل خطوط المعطيات من D0-D4 لإرسال المعلومات و يكون لكل منهما خط STROBE منفصل .

6.4. مشاكل انقضاء الزمن :

تعمل بروتوكولات الاتصالات عادةً بدون أخطاء طالما لم ينقطع التيار الكهربائي .و إذا حدث أي خطأ فإن كلاً من المرسل و المستقبل سيبقى في انتظار الطرف الآخر للاستجابة على الرسالة الأخيرة . و لمنع الحالة التي يتم فيها انتظار استجابة من أحد أطراف الاتصال للأبد ,استجابة قد لا تأتي أبداً , حصل انقضاء الزمن . تحدد هذه القيمة الفترة التي يمكن لنظام ما أن ينتظر لجواب الشريك قبل انهاء الاتصال .

لقد أشرنا في القسم الأول أن BIOS يستخدم أيضاً عداد لزمن الانقضاء للاتصال مع المنفذ التفرعي . يقاس مجال زمن الانقضاء بتنفيذ حلقة قراءة لعدد محدد من المرات .هذه العملية ليست سهلة الإدارة , من أجل برنامج سيعمل على مختلف أنظمة الحواسيب .يطلب هذا الزمن لمعالجة حلقة القراءة الذي يمكن أن يختلف حسب سرعة المعالج .

هنا مثال عن كيفية عمل عداد زمن الانقضاء . سوف نفترض أن المرسل قد أرسل فقط الأربع بتات الدنيا من البايت . و أن عداد انقضاء الزمن قد وضع على القيمة الأعظمية و بعدها انتظر حتى يجعل المستقبل BUSY=1 .ستبدأ حلقة القراءة بالتنفيذ .و سيستمر العمل حتى تتغير قيمة

بت BUSY إلى 1 أو يصل متحول عداد انقضاء الزمن إلى القيمة 0 .و يستمر العداد بالتنازل ما لم يصل المتحول إلى القيمة 0 .
يمكن أن يظهر ذلك بالكود التالي :

```
TimeOutCount = MAXVALUE
WHILE ( BUSY-Bit = 0 ) AND ( TimeOutCount > 0 ) DO
  BEGIN
  END

IF TimeOutCount = 0 THEN
  error
ELSE
  o.k.
END
```

يبقى برتوكول الاتصالات مفعلاً عل طول قراءة زمن الانقضاء في برامج الdemo مع الإجرائيات التي تدعى SendAByte , ReceiveAByte

6.5. التزامن:

إذا بروتوكول الاتصالات مفعّل,سوف يعمل بدون مشاكل . بعض الأحيان في البداية قد تعطي مشاكل . قبل بدء الاتصالات ,يجب أن يملك كلا المرسل و المستخدم نفس القيمة 0 في BUSY bit , و إذا لم يكن كذلك فإن المستقبل سيفترض مباشرة أن أربع بتات قد أرسلت و سيحاول قراءتها, مع أن المرسل لم يرسل شيئاً ما بعد.
يجب أن يتزامن المرسل و المستقبل قبل بداية الاتصالات . التحديد عندما يبدأ المرسل أو المستقبل أولاً أمر صعب.ففي برامج demo هنا سنستخدم المستقبل كنقطة بداية .
من أجل التهيئة ,ينتظر المستقبل حتى يعطي المرسل القيمة 0 لBUSY bit ,وبعدها يقوم المرسل بهذا . ويكمل التزامن عندما كل الطرفين يضعان القيمة 0 لBUSY bit . في برامج demo يتم هذا ضمن التابع PortInit.تستخدم إجرائية التهيئة الوقت المحدد time out limit ,إنها تعمل بما يوافق التوصيف الرئيسي السابق . بالإضافة إلى ذلك تمكّن البرامج المستخدم من الخروج من البرنامج في أي لحظة و ذلك بضغط مفتاح Esc . ومن ناحية أخرى ربما عليك انتظار عدة دقائق لمجال timeout ليمكن الامتداد فيها .

6.6. إيقاف البرنامج :

يتضمن كلا البرنامجين مقبض لمقاطعة لوحة المفاتيح و الذي يفعّل بضغط مفتاح Esc.مع مقبض مقاطعة المؤقت يستخدم البرنامج أيضاً متحول للاتصال مع مقبض مقاطعة لوحة المفاتيح ,إنه متحول منطقي يصبح له القيمة Trueعندما نضغط مفتاح Esc .
سوف يستجيب البرنامج ببساطة بالرغم من ذلك يكون قد امتدّ على المجال لtimeout .يمكنّك التفحص السريع للمتحول السابق المتعلق بـEsc من تحديد مصدر المقاطعة .

7. بروتوكول مستوى البلوك Block level :

إن مستوى البلوكBlock level أعلى من مستوى البايت . و كما يوحي الاسم ,فإن هذا المستوى يستخدم لنقل كامل بلوكات المعطيات من المرسل إلى المستقبل .أنه بروتوكول للبرمجيات بشكل كامل .إنه مستقلّ عن الكيان الصلب لأنه يثق بإجرائيات الإرسال و الاستقبال من مستوى البايت .
إنهم إجرائيات SendABlock, ReceiveABlock في برامجنا ال demo .
يحتوي البلوك عادة على المعلومات التالية :الرمز الذي يميزه بالإضافة لمحتوياته , طول البلوك و البلوك نفسه .
يستخدم الرمز ليقوم المستقبل بالتنظيم المباشر للذي أرسل دون قراءة معطيات البلوك . و بما يوافق هذا تقوم إجرائية SendABlock بانتظار لتمرير الرمز و عدد البايتات و مؤشر على البلوك نفسه .إنّ كلا الرمز و عدد البايتات المحمّلة هم جزء من ال ترويسة و تحفظ بشكل منفصل عن بلوك المعطيات الفعلي . يجب على المستقبل أولاً أن يستقبل الترويسة بشكل صحيح و ذلك قبل استقبال البايت الأول من معطيات البلوك . تخيّل ماذا سيحدث إذا أراد المرسل إرسال بلوك معطيات من 120 bytes و لكن المستقبل استقبل بلوك من 200 bytes .سوف يعد المستقبل ال80bytes التالية كجزء من بلوك المعطيات الأول , و سيصبح الاتصال متشابك بشكل مستحيل .
تذكر أنه عند مستوى البايت ,يرسل المستقبل كل بايت استقبله إلى المرسل .لذا سوف يعرف بروتوكول مستوى البلوك بشكل مباشر فيما إذا تم نقل الترويسة بدقة . و لسوء الحظ هذا لا يدع المستقبل يعرف فيما إذا استقبل الترويسة بشكل صحيح . لذلك يقوم المرسل بتنبيه المستقبل بإرسال ميزة نظامية .لذا لا يجب على المستقبل أن يفترض عدم وجود أخطاء في الترويسة .
و كتغذية عكسية ,يرسل المرسل الميزة ACK إذا تم النقل بنجاح ,أو NACK إذا حصل خطأ .
تمثّل ميزة ACK أوNACK بالشيفرة 00H و FFH في برامج demo ,و لكن باستطاعتك استخدام أي شيفرة . تحتل هذه الميزات أيضاً جزء في بروتوكول الاتصالات حتى أنهم يُفحصوا في مستوى البايت من أجل نجاح عملية النقل .
عندما يستلم المستقبل الترويسة و ميزةACK بدون أخطاء ,يستطيع المرسل البدء بنقل بلوك المعطيات الفعلي . و إذا حدثت مشكلة يعيد المرسل نقل الترويسة .سيعرف المستقبل إذا التريسة أعيد إرسالها لأنه كان قد استقبل ميزة NACK من المحاولة السابقة .فإذا استلم المستقبل الترويسة و ميزة ACK ,يستطيع عندئذ التركيز على استقبال بلوك المعطيات .و كلما زاد طول العينات من البتات التي تعبر عن الميزتانACK ,NACK ,يمكن أن تتحول ميزة ACK إلى الميزةNACK لخطأ في نقل البيانات .
إذا تابع في مواجهة الأخطاء ,سوف لن يحافظ المرسل على محاولة إرسال الترويسة للأبد .لقد وضع الثابت MAXTRY ليقول للمرسل على عدد الأخطاء لتعدها قبل تجاهل محاولة الإرسال لبلوك المعطيات الحالي . الميزتانACK ,NACK تستخدمان أيضاً للتأكيد على استلام بلوك المعطيات . ترسل ميزة التغذية العكسية Feedback للمستقبل بعد إرسال كامل بلوك المعطيات .
و بهذه الطريقة يمكن لكل أنواع أخطاء الاتصالات أن تُكتشف . يزيل البروتوكول الحاجة التفحصات التي هي طريقة عامة لفحص أخطاء النقل في برمجيات الاتصالات .

7.1. قراءة حالة مفتاح Esc :

إذا نُقل بلوك من المعطيات , يرسل البايت الأخير لإتمام العملية . ولكن يرسل هذا البايت من المستقبل إلى المرسل .و يعطي أيضاً للمستقبل إمكانية الاتصال بإشارة ESCAPE مع المرسل .
على أي حال إنها ليست ضرورية حقاً لأن المستقبل يستطيع بسهولة إنهاء برمجية الاتصالات مع ESCAPE وثم ينتظر المرسل حتى وقوع خطأ انقضاء الوقت .

بالرغم من ذلك ,فإنه ليس الحل الأفضل ينتظر المرسل في البروتوكول من أجل "escape byte " من المستقبل و ذلك بعد أن يرسل بلوك المعطيات بشكل كامل .إذا استقبل المرسل قيمةtrue فإنه ينهي البرنامج بالرسالة الموافقة . و من ناحية أخرى يستمر المرسل بتنفيذ البرنامج بشكل طبيعي .
الاتصال بهذا النوع من الرسائل يجب أن يكون متاحاً في كلا الاتجاهين ,حتى المرسل يمكن أن يقرر أيضاً إنهاء الاتصال في أي وقت ,و هنا تختلف الإجرائية عن المستقبل . عندما تبدأ, تحدد إجرائية SendABlock فبما إذا تم ضغط المفتاح Esc ,وإذا تم ذلك ترسل تأشيرة ESCAPE خاصة بدلاً من ترويسة بلوك المعطيات الفعلي . تكون إشارة ESCAPE الخاصة معروفة للمستقبل .عندما يتعرف المستقبل على هذه التأشيرة ,يعتبرها كإشارة لإنهاء البرنامج .
ببناء هذه الآلية لESCAPE ضمن بروتوكول البلوك ,نستطيع صنع استعلام دائم عن ESCAPE في أعلى مستوى.توزّع برامج demo أيضاً مع مستوى الملفات الأعلى من مستوى البلوك .يستخدم مستوى الملفات بروتوكول مستوى البلوك لنقل كامل الملفات جزء تلو الآخر .لن نذهب في تفاصيل مستوى الملفات .فيما مضى , سماعات البرنامج هي مدعمة جداً بالوثائق في نهاية هذا القسم .
تذكر أنه يمكن استخدام الإجرئيات في مستوي البلوك و البايت لنقل أي معطيات بين المرسل و المستقبل ,كما أنه يمكن للمرسل و المستقبل أن يقوموا بإجهاض الاتصال في أي وقت و في أي نقطة . تستطيع هذه الإجرائيات تخديم ما يلزمك كأساس لبرامجك الخاصة لنقل المعطيات ,كما أنها تزاحم الرزم التجارية مثل LapLink .يمكن ألا تكون بنفس السرعة تماماً , و ذلك لأن هذا يتطلب كتابة كل إجرائيات مستوى البايت بلغة الاسمبلي .

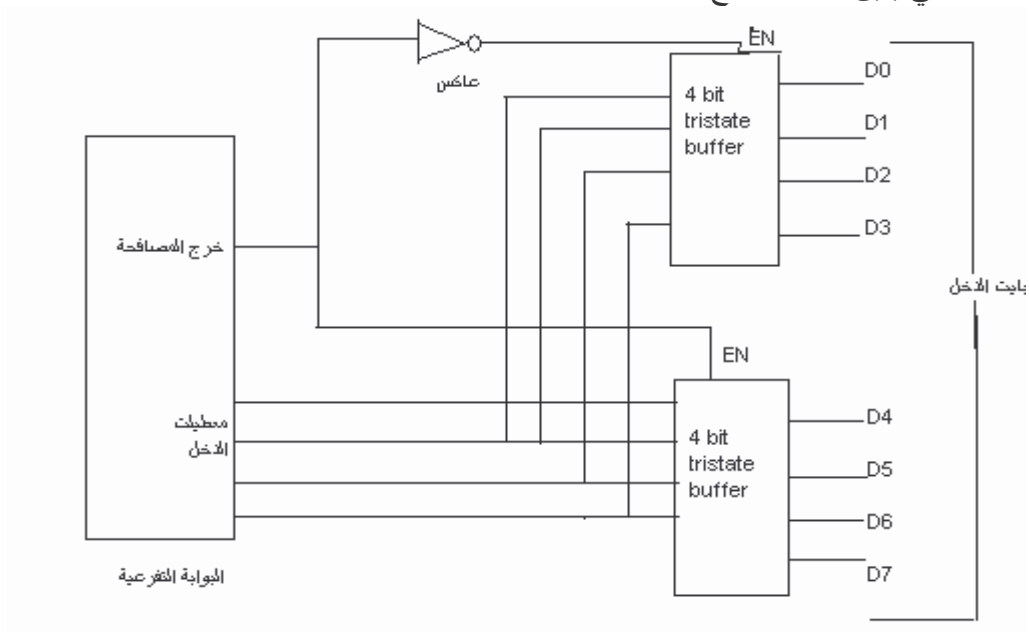
7.2. المستويات الأعلى :

إذا حصل خطأ ما ,مثل خطأ انقضاء الوقت في مستوى البايت أو تأشيرة ESCAPE في مستوى البلوك ,فإنه يمكن لهذا الخطأ أن ينتقل إلى أعلى مستوى بأقصى سرعة ممكنة . المستويات المختلفة في برامج الdemo (البايت , البلوك , الملف)تستخدم الإجرائيات و التوابع للاتصال فيما بينها . عندما ينتهي أي إجراء بحدوث خطأ فإنه يعاد للإجراء المستدعي بحيث يعود بطريقه من مستوى إلى المستوى الأعلى فالأعلى .

7.3. كيفية إدخال بايت كامل:

من الواضح أننا لا نستطيع إدخال بايت كامل دفعة واحدة عبر خطوط الدخل in3---in7 لأن عددها أقل من ثمانية . إن استخدام هذه الخطوط من أجل إدخال بايت يدفعنا إلى الاستعانة ببعض الدارات المتكاملة البسيطة . حيث يمكننا على سبيل المثال استخدام أحد خطوط الخرج OUT X كخط انتخاب(multiplexing line) , حيث أنه يتم انتخاب و إدخال البتات الأربعة الدنيا عندما يكون خط الانتخاب مساوياً للصفر على سبيل المثال , ثم يتم في المرحلة الثانية انتخاب و إدخال البتات الأربعة العليا و ذلك بجعل خط الانتخاب مساوياً للواحد .
حيث نضع دارتي عازل ثلاثي الحالات (tristate Buffer) , وكل منهما بعرض 4 بتات , ونصل مخارج هذين العازلين إلى خطوط الدخل للبوابة التفرعية .
يقوم أحد خطوط الخرج out x بقيادة مدخل التأهيل(Enable) لأحد العازلين , بينما تتم قيادة مدخل تأهيل العازل الآخر عبر نفس خط الخرج و لكن بعد مروره بعاكس (enverter) . وهكذا نضمن أنه لن يتم تأهيل إلا أحد العازلين في نفس اللحظة , ويمكن اختيار تأهيل أحدهما أو الآخر بإخراج القيمة المناسبة على خطوط الخرج المستخدم .

وهكذا يتم إدخال البايت الكامل بإدخال البتات الأربعة الدنيا D0-D3 عبر العازل الأول , والبتات الأربعة العليا D4-D7 عبر العازل الثاني.
والشكل التالي يبين المبدأ المتبع :



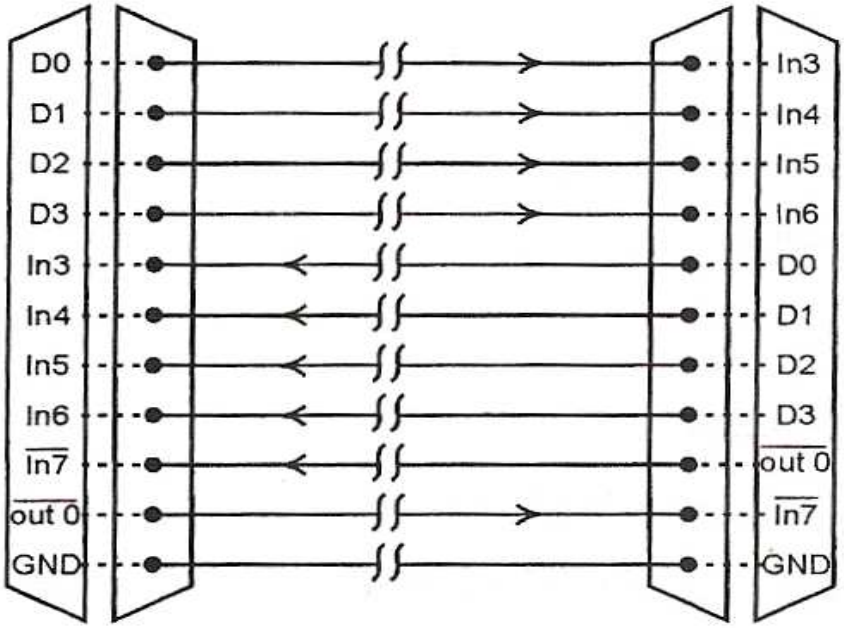
الشكل 4 : استخدام البوابة التفرعية كبوابة دخل رقمية ثمانية البتات

إذاً لإدخال بايت كامل فلا بد من قراءة العازل الأول ثم قراءة العازل الثاني , ومن ثم يقوم البرنامج المكتوب بتشكيل القيمة الصحيحة للبايت الكامل D0-D7 ابتداءً من القيمتين المقروءتين .

من الواضح أن هذه الطريقة ليست مباشرة و سريعة كما لو كنا نقرأ البايت دفعة واحدة , ولكن حتى مع الحواسيب البطيئة نسبياً فإن معدلات سرعة النقل بهذه الطريقة يمكن أن تبلغ عدة مئات الكيلو بايتات في الثانية . وهذه الطريقة حتماً أسرع من استخدام البوابة التسلسلية التي لن يزيد معدل سرعة النقل فيها عن عدة كيلو بايتات في الثانية .
ولكن هنا تواجهنا مشكلة بسيطة : وهي احتمال تغير قيمة المعطيات في الفترة الواقعة بين قراءة النصف الأول و قراءة النصف الثاني . وهذه المشكلة ستكون موجودة كلما كان عرض بوابة الدخل أقل من عرض المعطيات التي نود قراءتها .
في كثير من التطبيقات يمكن تجاوز هذه المشكلة دون إضافة أية دارات متكاملة إضافية , وذلك عندما تكون سرعة تبدل المعطيات قليلة نسبياً , ولكن مع ذلك يفضل عادة اتخاذ بعض التدابير لضمان صحة قراءة المعطيات , فمثلاً يمكن اللجوء إلى تنفيذ القراءة مرتين أو أكثر , ومقارنة هذه القراءات , فإذا كانت النتيجة هي نفسها في القراءات المتتالية فهذا يعني أن المعطيات لم تتغير أثناء إجراء هذه القراءات .
أما إذا إردنا ضمان صحة القراءة 100% فلا بد من استخدام ماسك (Latch) للبايت بكامله قبل قراءته , فبواسطة أحد خطوط الخرج out x نقو مبمسك البايت كاملاً , ومن ثم نقوم بقراءة البايت الممسوك على دفعتين من خرج الماسك.
إن ضمان صحة القراءة 100% سيكون على حساب الكلفة الزائدة ثمناً للماسك , وعلى حساب انخفاض معدل سرعة نقل المعطيات قليلاً بسبب الحاجة لتنفيذ تعليمتين لإضافيتين : تعليمة مسك المعطيات قبل إجراء القراءة , وتعليمة العودة لمتابعة تغير المعطيات بعد إجراء القراءة .

8. الربط بين الحواسيب الشخصية عن طريق البوابات التفرعية :

سوف نعالج إمكانية وصل حاسبين مع بعضهما البعض عن طريق البوابات التفرعية وهذا لا يحتاج من ناحية المعدات إلا كبل توصيل بين البوابتين التفرعيتين . و إن البساطة الشديدة في الاحتياجات العتادية يقابلها إلى حد ما تعقيد في البرامج اللازمة لاستثمار لتطبيق .
يمكن تطبيق مفهوما لبوابات التفرعية في تطبيقين مهمين في الحياة العملية هما :
أولاً: تطبيق لإرسال و استقبال الملفت عبر البوابة التفرعية .
ثانياً: إنشاء محادثة عبر الحاسبين المرتبطين بالبوابة التفرعية .
أما من ناحية الكيان الصلب اللازم لهذه العملية أو لهذا التطبيق فهو بسيط للغاية حيث نحتاج إلى كبل إضافة إلى موصلين(connectors) من النوع D ب 25 نقطة حيث يتم الوصل بينهما كما في الشكل:



الشكل 5 : كبل التوصيل للربط بين الحاسبين

إذاً يتم كما هو واضح من الشكل السابق أنه يتم وصل القسم الأدنى من بتات المعطيات D0 حتى D2 وهي بتات الخرج من الحاسب الأول إلى بتات الدخل In3 إلى In6 في الحاسب الثاني , وكذلك بتات المعطيات في الحاسب الثاني يتم وصلها ببتات الدخل في الحاسب الأول . أن التوصيل بالشكل السابق يعني أننا لن نكون قادرين على إرسال أكثر من نصف بايت دفعة واحدة , و أن الطرف المرسل يجب أن يرسل هذا النصف على البتات الدنيا (أي D3 D2 D1 D0) .
اما الطرف المستقبل فيقوم في كل مرة باستقبال نصف بايت , ويتم هذا الإستقبال على البتات In3 In4 In5 In6 .
وبما أنه لا يمكننا إرسال أكثر من نصف بايت دفعة واحدة , ولذلك عندما نريد إرسال بايت كامل B فيجب تقسيمه إلى جزئين L,H و إرسالهما على التتالي .
لنفترض أن البايت B الذي نريد إرساله يكتب على الشكل التالي:

B=b7b6b5b4 b3b2b1b0
H L

لإرسال النصف الأدنى L= b3b2b1b0 فيكفي جعل L تأخذ القيمة B ومن ثم القيام بإخراج القيمة L على البوابة 378h (وفي هذه الحالة سيتم إخراج البتات

b3b2b1b0 على الخطوط المقابلة لها D3 D2 D1 D0) . أما لإرسال النصف الأعلى H=b7 b6 b5 b4 فلا بد لنا من إزاحة هذه البتات نحو النصف الأدنى (لأنه يتم الإرسال على الخطوط الدنيا D3 D2 D1 D0 وليس على الخطوط العليا D7 D6 D5 D4).و ولذلك يجب جعل H=xxxxb7b6b5b4 قبل إخراجها على البوابة 378h . وتتم هذه الإزاحةعلى سبيل المثال بتنفيذ التعليمة التي تسند القيمة B div 16 إلى H .

وكذلك الأمر من جهة المستقبل فإنه يجب استقبال البايت B على دفعتين متتاليتين , أولاً يتم استقبال النصف الأدنى L ثم يتم استقبال النصف الأعلى H ليتم لاحقاً تشكيل البايت B . يتم استقبال كل نصف بقراءة بوابة الدخل 379h . ونلاحظ من طريقة الوصل المعتمدة أن البتات النفيدة هي البتات المقابلة للمداخل In3 In4 In5 In6 .ولذلك فإنه يتم تشكيل النصف الأدنى L بإزاحة البايت الذي يتم استقباله مثل R على سبيل المثال نقوم بإزاحته ثلاثة بتات نحو اليمين , وتصفير بقية البتات أي أننا نسند القيمة (port[$379]div8)and $0F)) نسندها إلى L . أما النصف الأعلى H فيتم تشكيله بإزاحة البايت الذي يتم استقباله وهوR بتاً واحداً نحو اليمين وتصفير البتات الدنيا , أي أن (port[$379]*2)and $F0))=Hأ . وأخيرا يتم تشكيل كامل البايت B بجمع L و H , أي أن B=L+H .

يتم في التوصيل السابق استخدام الخطين In7 المعكوس داخلياً و out0 المعكوس داخليا ص أيضاً من أجل تحقيق بروتوكول الإتصال بين المرسل و المستقبل . يتم وصل المخرج out0 في الطرف الأول إلى المدخل In7 في الطرف الثاني , وكذلك out0 في الطرف الثاني إلى In7 في الطرف الأول . إن هذا الوصل يلغي تأثير العواكس الداخلية لنه عندما يتم إخراج قيم ةما على المخرج out0 فيظهر عكس هذه القيمة على هذا الخط , وعندما يتم إدخالها في الطرف الأخر عن طريق المدخل In7 فيتم أيضاً عكس هذه القيمة).

ويجب الإنتباه عند توصيل الكبل في الطرفين , لأن أي خطأ في عملية التوصيل قد تؤدي إلى ضرر في الحاسب , حيث أن مواضع الخطوط في موصل الطابعة (الموصل المؤنث الموجود في الحاسب) تنعكس عند التعامل مع الموصل الموجود في نهاية الكبل (الموصل المذكر), أي أن الطرف اليميني للموصل سيصبح يسارياً في الطرف اليساري للموصل المذكر , والعكس بالعكس.

8.1. الخيارات المتاحة لتنفيذ بوابة دخل بثمانية بتات :

بما أنه توجد 5 خطوط دخل IN3-IN7 , فإنه توجد أكثر من طريقة لتنفيذ بوابة دخل بثمانية بتات , تتمثل الطريقة الأولى بربط مخارج كل من العازلين إلى المداخل الأربعة العليا IN4-IN7 . يتم بداية انتخاب العازل المربوط على بتات الدخل الدنيا D0-D3 وذلك بإخراج القيمة المناسبة على خرج المصافحة الذي يؤهل هذا العازل , ثم تتم عملية القراءة من البوابة و يتم تخزين القيمة الناتجة في متحول (variable) . يتم في المرحلة الثانية انتخاب العازل الآخر المربوط على بتات الدخل العليا D4-D7 , وذلك بإخراج القيمة المناسبة على خرج المصافحة, ثم تتم عملية قراءة ثانية و تخزين النتيجة في متحول ثانٍ . إن القيمة الناتجة عن قراءة البتات العليا D4-D7 صحيحة و لا تحتاج لأي تصحيح لأنها موصولة على المداخل العليا IN4-IN7 وهذا الأمر لا ينطبق على البتات الدنياD0-D3 لأنها تقرأ على المداخل العليا IN4-IN7 , لذلك لا بد من إجراء تصحيح للقيمة الناتجة , يتمثل هذا التصحيح بتقسيم القيمة المقروءة على 16 (وهذا مكافئ لإجراء إزاحة بمقدار 4 بتات نحو اليمين). و للحصول على قيمة البايت كاملاً تتم إضافة ناتج القسمة إلى القيمة المقروءة من العازل ذي البتات العليا . فلو افترضنا أنه يتم تخزين القراءة الأولى D0-D3 في المتحول v1 , وأنه يتم تخزين القراءة الثانية D4-D7 في المتحول v2 و عندها تحسب قيمة البايت كاملاً كمايلي :

B=v1/16+v2

وهنا توجد مشكلة صغيرة , وهي كون المدخل الأخير in7 معكوساً , ويمكن التغلب على هذه المشكلة ببساطة بوضع عاكس للبت الأخير D7 , كما يمكن الإستغناء عن هذا العاكس و إجراء التصحيح المناسب بواسطة البرنامج . يمكن الإستغناء عن هذا العاكس و إجراء التصحيح المناسب بواسطة البرنامج , ويمكن الإستغناء عن حالة الربط مع البت المعكوس بإجراء الربط عبر الداخل IN3-IN6. وتتم عملية قراءة المعطيات بهذه الطريقة بشكل مشابه للطريقة السابقة : حيث يتم أولاً انتخاب و تأهيل العازل ذو البتات الدنيا , ثم تتم عملية القراءة و تخزين النتيجة في متحول v1 . يتم بعد ذلك انتخاب و تأهيل العازل ذو البتات العليا , ثم تتم عملية القراءة و تخزين النتيجة في متحول ثان v2 .

يجب أأن يتم في هذه الطريقة تصحيح كلتا القراءتين : يتم تصحيح القراءة الأولى بتقسيمها على 8 (يكافئ هذا عملية إزاحة بمقدار ثلاثة بتات نحو اليمين), كما أن القراءة الثانية يتم ضربها ب 2 (يكافئ هذا عملية إزاحة بمقدار بت واحد نحو اليسار) . و بإضافة القيمتين المصححتين نحصل على البايت B كاملاً كمايلي :

B=v1/18+v2*2

وليس مهماً إياً من مخارج المصافحة الأربعة سيتم استخدامه كخط انتخاب للعازلين . وإذا وجد خطا خرج مصافحة غير مستخدمين فيمكن الإستغناء عن العاكس المستخدم في حالة وجود خط انتخاب وحيد . و في هذه الحالة يتم انتخاب العازل الأول باستخدام أحد خطي الإنتخاب , فيما يتم انتخاب العازل الثاني بواسطة الخط الآخر . تقع في هذه الحالة المسؤولية على عاتق البرنامج لضمان عدم تأهيل و انتخاب العازلين معاً .

و بالطبع في حالة استخدام خطي انتخاب منفصلين يجب توخي الحذر الشديد في البرنامج لضمان عدم تأهيل العازلين معاً , الأمر الذي يسبب بعطب العازلين في حال حدوثه . حتى في الحالة التي يكون فيها البرنامج صحيحاً فإن النتيجة غير مضمونة , لأن حالة المخارج عند تشغيل (switch on) الحاسوب وقبل تنفيذ البرنامج غير معروفة , لذلك يفضل اللجوء إلى استخدام خط انتخاب وحيد مع عاكس .

و بهذا الشكل نستطيع القول أن البوابة التفرعية توفر لنا بوابة خرج ثمانية البتات وبوابة دخل ثمانية البتات . وثلاثة مخارج للمصافحة , ودخل وحيد للمصافحة .

ويعتبر هذا كاف لكثير من التطبيقات , علماً أننا نستطيع بنفس الطريقة (طريقة الإنتخاب) مضاعفة عدد المداخل و المخارج . فباستخدام خطي انتخاب يمكن انتخاب واحد من أربعة عوازل ثلاثية الحالة و كلاً منها بعرض 4 بتات , وبالتالي نحصل على بوابة دخل بعرض 16 بتاً . وباستخدام 3 خطوط للإنتخاب يمكن الحصول على بوابة دخل بعرض 32 بتاً .

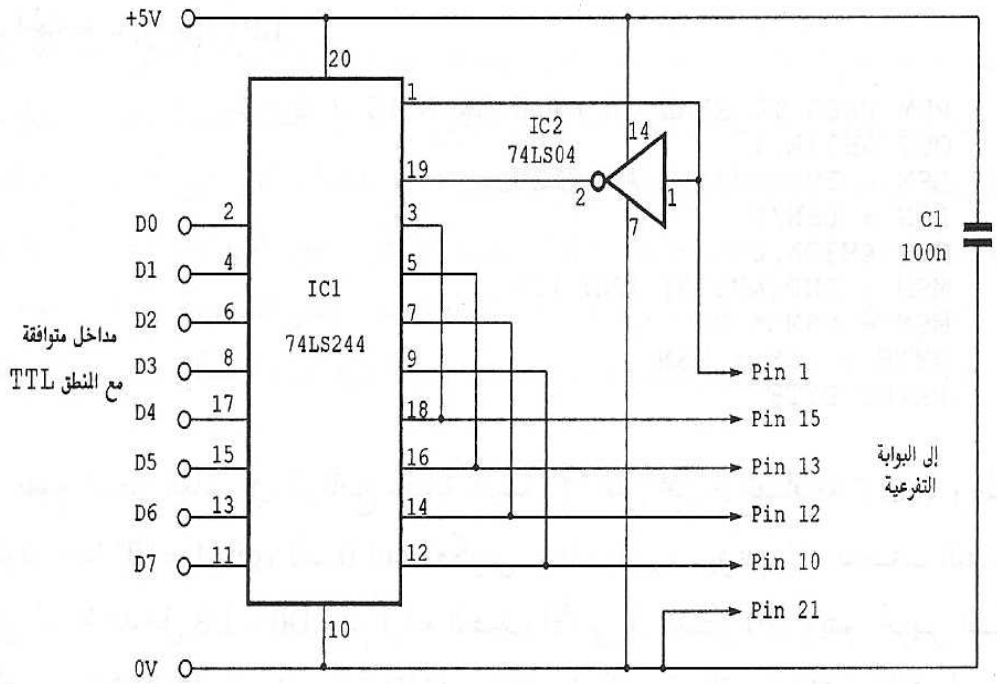
إن توسيع بوابة الدخل لأكثر من 16 بتاً يصبح معقداً بعض الشيء , وفي هذه الحالة يمكن اللجوء إلى ممرات التوسعة , مع أن هذا قد يكون أكثر كلفة و أكثر و ليس أقل تعقيداً , ولكنه سيكون أكثر فاعلية . ويمكن بهذه الطريقة تطبيق تقنية الإنتخاب على بوابة الخرج للحصول على بوابة خرج بعرض 16 بتاً .

8.2. بوابة الدخل :

بعد الإطلاع على المبدأ الأساسي (Basic Principle) لتشكيل بوابة دخل و خرج باستخدام البوابة التفرعية , سنعرض فيما يلي بعض الدارات العملية لتشكيل بوابات الدخل و الخرج . كما رأينا سابقاً توجد العديد من طرق الانتحاب , و سيتم التعرض لها تباعاً , و الطريقة الأفضل ستكون بحسب التطبيق الفعلي . و لكن في معظم الأحيان ستكون الحلول متكافئة و يبقى الخيار للمستثمر بحسب ما يجده مناسباً .

تشكل الدارة في الشكل 7 بوابة دخل ثمانية البتات . يتم في هذه الدارة استخدام البتات

In3 – In6 , الأمر الذي يجنبنا التعامل مع المدخل In7 المعكوس .



الشكل 7 : دارة بوابة الدخل الرقمية ثمانية البتات

يوجد العديد من العوازل ثلاثية الحالة و التي يمكن استخدامها في هذا التطبيق , تم هنا استخدام العازل الثماني (Octal Buffer) الأكثر شهرة : 74 LS 244 . رغم أن هذه الشريحة توصف بأنها عازل ثماني , إلا أنها في الحقيقة عبارة عن عازلين كل منهما رباعي البتات و لكل منهما خط تأهيل منفصل , و هذا يجعلها الشريحة المثالية لتطبيقنا الحالي , لقد تم في هذه الدارة استخدام خط مصافحة وحيد مع عاكس لتأمين الانتخاب المتعاكس , أي أن انتخاب أحد النصفين سيؤدي حتماً لتعطيل انتخاب النصف الآخر . من الممكن استخدام أي خط خرج لتأمين عملية الانتخاب , و قد تم هنا اختيار الخط Out0 لتبسيط الأمور .

يتم انتخاب النصف الأدنى D0 – D3 عندما يكون Pin1 في حالة الصفر المنطقي , فيما يتم انتخاب النصف الأعلى D4 –D7 عندما يكون في حالة الواحد المنطقي .

يجب الانتباه إلى أن Pin1 معكوس داخلياً , لذلك فإن كتابة القيمة "1" عليه يعني ظهور الصفر"0" و كتابة الصفر "0" يعني ظهور الواحد "1" .

يقوم البرنامج التالي بقراءة القيمة الموجودة على بوابة الدخل , و من ثم يقوم بطباعة النتيجة على الشاشة . يستخدم هذا البرنامج بوابة الطابعة الأولى LTP1 , أما إذا كانت البوابة المستخدمة هي البوابة LTP2 فيلزم فقط تغيير العناوين , و استخدم تلك الخاصة بالبوابة LTP2 .

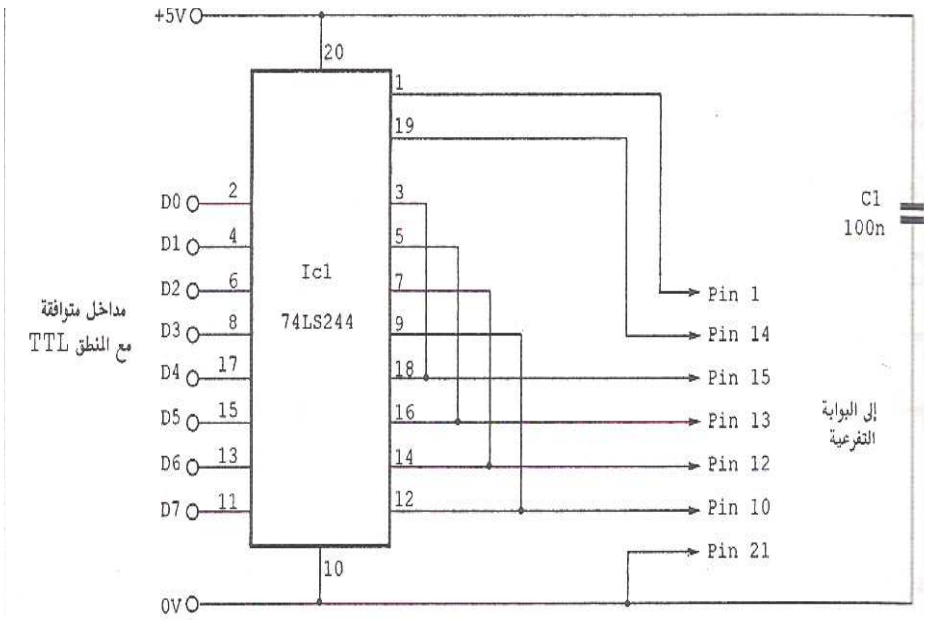
```
REM PROG TO READ IN BYTE ON BITS 3 TO 6
 OUT &H37A , 1
LSN = INP ( &H379 ) AND 120
LSN = LSN/8
OUT &H37A , 0
50  MSN = INP ( &H379) AND 120
60  MSN = MSN * 2
```

```
70  BYTE = LSN + MSN
80  PRINT BYTE
```

يقوم السطر العاشر من البرنامج بكتابة القيمة "1" على مخارج المصافحة out0 , و بالتالي سيكون لدينا "1"=Pin1 , لأن out0 معكوس . و هذا بدوره يؤدي إلى انتخاب النصف الأدنى لبوابة الدخل D0 – D0 . تتم قراءة النصف الأدنى في السطر 20 , و يتم تخزين الناتج في المتحول "LSN" يتم إجراء عملية AND بيت القيمة المقروءة و القيمة 120 , و ذلك لحجب البتات الغير موصولة فعلياً على المداخل In3 –In6 . فالقيمة 120 تكتب على الشكل الثنائي كما يلي : 120=&b0111 1000 . نلاحظ أن البتات D3 –D6 تأخذ القيمة "1" في العدد 120 , بينما بقية البتات تأخذ القيمة "0" . عند إجراء عملية AND فإن جميع البتات غير البتات In3 – In6 ستكون أصفاراً في النتيجة .
القيمة المخزنة في المتحول "LSN" غير صحيحة و تحتاج إلى تصحيح , لأنها لم تقرأ على الخطوط المقابلة (الخطوط In3 – In6 بدلاً من الخطوط In0 – In3) . عند البرمجة بلغة التجميع يمكن إجراء التصحيح بتنفيذ تعليمة دوران أو عملية إزاحة لوضع البتات في مواضعها الصحيحة . أما عند استخدام لغات البرمجة عالية المستوى فالأسهل إجراء التصحيح عن طريق الضرب أو التقسيم . يتم تصحيح القيمة في هذا البرنامج في السطر 30 , و ذلك بتقسيم ناتج القراءة على القيمة 8 و تخزين النتيجة في نفس المتحول "LSN" . يتم في السطر 40 انتخاب النصف الأعلى D4 –D7 , و تتم في السطر 50 عملية القراءة و تخزين الناتج في المتحول "MSN" . يتم تصحيح القراءة بضربها بالقيمة 2 في السطر 60 , ويتم تخزين النتيجة في نفس المتحول "MSN" . يتم في السطر 70 جمع محتوى المتحولين "LSN" و "MSN" للحصول على قيمة البايت كاملاً , ويتم تخزين النتيجة في المتحول Byte , ليتم لاحقاً طباعته على الشاشة في السطر 80 .

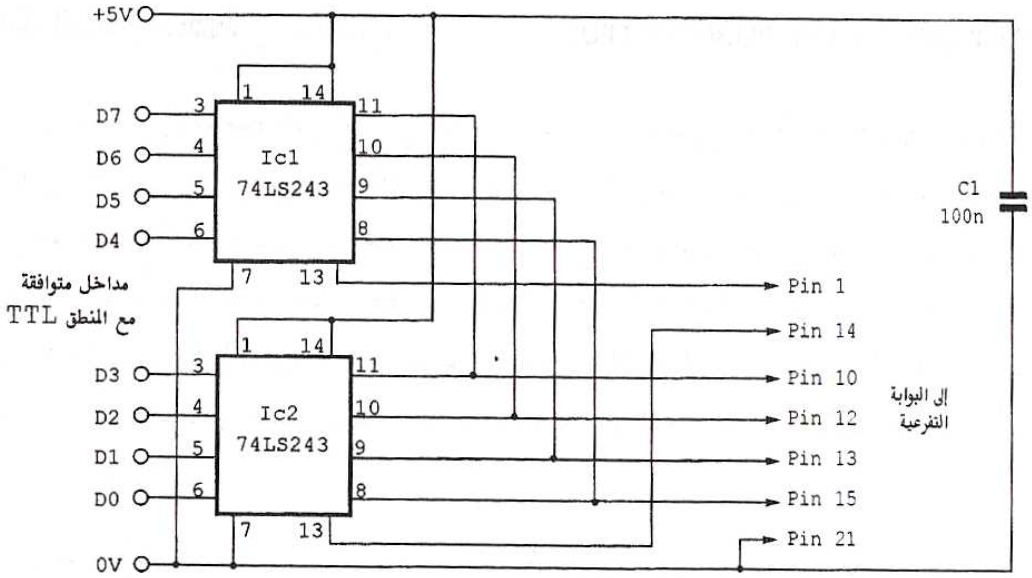
8.3. بوابة الدخل بخطي انتخاب :

من الممكن الاستغناء عن العاكس إذا ما استخدمنا خطي مصافحة خرج للانتخاب بدلاً من خط واحد . من الواضح أن استخدام هذه الطريقة سيؤدي إلى استهلاك خطي خرج مصافحة , أي أننا سنخسر أحد خطوط خرج المصافحة مقابل توفير عاكس . قد يكون هذا مفيداً إذا كنا لا نحتاج خطوط المصافحة هذه لأهداف أخرى . و لكن يجب أن لا ننسى التحذير السابق الذكر , إذا من الممكن تأهيل كلا العازلين في نفس اللحظة بسبب خطأ ما في البرنامج , أو عند تشغيل الحاسب و اختيار المخارج التي تؤدي إلى تأهيل العازل عند الإقلاع .
هنا الشكل 8 يبين استخدم الشريحة 47LS244 مع خطي مصافحة هرج للانتخاب هما :
Pin1=out0 و Pin14=out1 .



الشكل 8 : دارة بوابة دخل رقمية ثمانية البتات ذات خطي انتخاب

الشكل التالي يبين الدارة المكافئة للدارة السابقة حيث تستخدم زوجاً من الشريحة 47LS243 :



الشكل 9 : دارة أخرى لبوابة دخل رقمية ثمانية البتات ذات خطي انتخاب

من الواضح أن هذا التغيير في الكيان الصلب للنظام سيتبعه تغيير في الكيان اللين المرافق . إن التغيير في البرنامج سينحصر في تعليمات الإخراج out , والتي بواسطتها يتم انتخاب أحد نصفي المداخل . لانتخاب أحد نصفي المداخل يجب إخراج القيمة المناسبة على خط خرج المصافحة المقابل لهذا النصف . فانتخاب النصف الأدنى يتم بجعل "out0=Pin1="0 و جعل "out1=Pin14="1 , أما انتخاب النصف الأعلى فيتم بإخراج القيم المعاكسة أي "out0=Pin1="1 و "out1=Pin14="0 .

يجب ملاحظة أن خرجي المصافحة out0 و out1 مزودين بعاكسين داخليين , و لهذا فإن إخراج الصفر على أحدهما يتم بكتابة القيمة "1" في البت المقابل , لذلك فإن القيمة

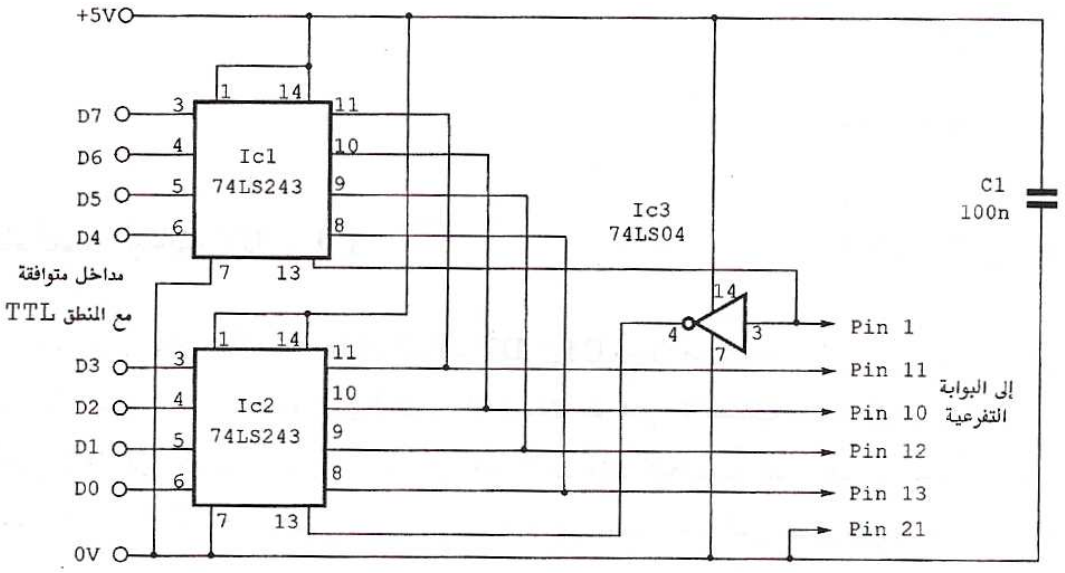
&B00000001 ستنتخب النصف الأدنى للمداخل بينما القيمة &B00000010 ستنتخب النصف الأعلى و بذلك يكون البرنامج بعد كما يلي :

```
REM PROG TO READ IN BYTE ON BITS 3 TO 6
OUT &H37A , 1
LSN = INP ( &H379) AND 120
LSN = LSN/8
OUT &H37A , 2
50  MSN = INP ( &H379) AND 120
60  MSN = MSN * 2
70  BYTE = LSN + MSN
80  PRINT BYTE
```

8.4. استخدام البتات D4 –D7 :

قد يقول قائل بأن استخدام البتات D4 – D7 بدلاً من البتات In3 – In6 يشكل الحل الأنسب . إن هذا الحل سيكون هو الأنسب فيما لو كان المدخل الأخير In7 غير معكوس داخلياً . و لكن مع ذلك فإن التغلب على هذه المشكلة أمر بسيط : فإما أن نقوم بعكس المدخل الأخير بواسطة عاكس خارجي , أو أن نقوم بتصحيح القيمة المقروءة عن طريق البرنامج لها سيئة و حيدة , و هي أننا سنكون بحاجة لبعض التعليمات الإضافية , الأمر الذي سيتسبب في إبطاء معدل سرعة النقل التي يحتاجها التطبيق , فإذا كانت سرعة النقل مع كون التصحيح يجري عن طريق البرنامج كافية فلا داعي لاستخدام عاكس خارجي , و إن لم تكن هذه السرعة كافية فعندها نلجأ إلى العاكس الخارجي .

الشكل التالي يبين الدارة باستخدام البتات In4 – In7 , وباستخدام شريحتي 47LS243 حيث نستطيع استخدام نفس الطريقة مع الشريحة 47LS244 .



الشكل 10 : بوابة دخل رقمية ثمانية البتات تستخدم البتات 4 حتى 7 مع عكس البت 7 برمجياً

هذه الدارة لا تحتوي على عاكس خارجي , و لذلك لا بد من إجراء تصحيح للقيمة المقروءة .
البرنامج التالي يبين كيفية قراءة البوابة :

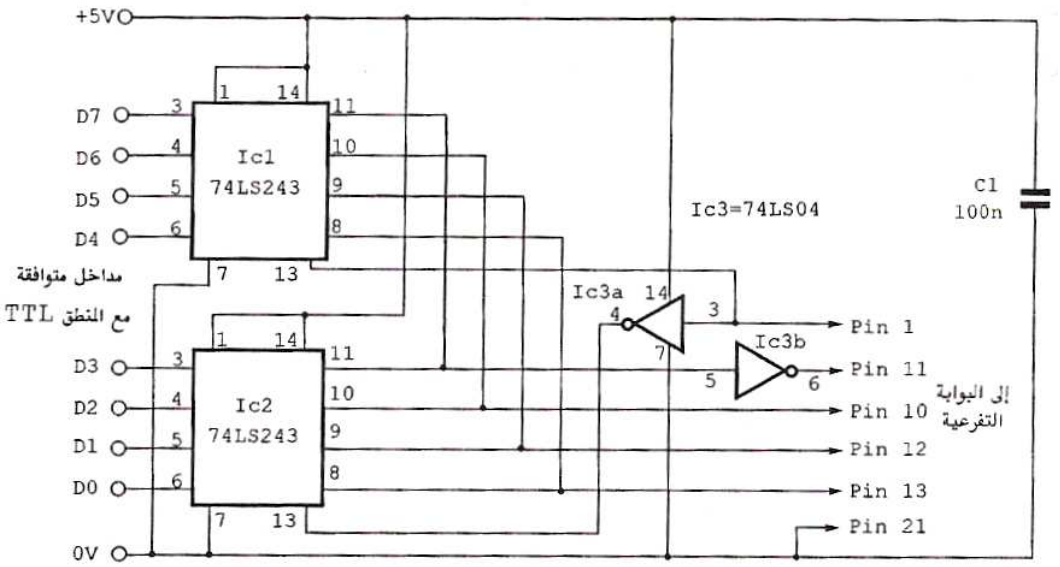
```
REM PROG TO READ IN BYTE ON BITS 4 TO 7
OUT &H37A , 1
LSN = INP ( &H379) AND 240
IF  LSN >127 Then LSN=LSN-128 Else LSN=LSN+128
40  LSN=LSN/16
50  OUT &H37A , 0
60  MSN = INP ( &H379) AND 240
70  IF MSN>127 Then MSN=MSN-128 Else MSN=MSN=MSN+128
80  BYTE = LSN + MSN
90  PRINT BYTE
```

يشابه هذا البرنامج البرامج السابقة , و يكمن الاختلاف الوحيد في طريقة معالجة القيمة المقروءة . في السطرين 20 و 60 نستعمل قناعاً بالقيمة 240=&B1111 0000 . حيث نلاحظ أن هذا القناع يؤدي إلى تصفير البتات الأربعة الدنيا , بينما يبقي البتات الأربعة العليا على ما عليه , و هذا ما نحتاجه لأننا نستخدم الخطوط العليا In4 – In7 . بعد إجراء القراءة لا بد من عكس البت الأخير , وهذا يجب أن يتم سواء عند قراءة المداخل الأربعة D4 –D7 . يتم خذا التصحيح في السطرين 30 (من أجل النصف الأدنى) و 70 (من أجل النصف الأعلى) . تتم عملية التصحيح بفحص قيمة البت الأخير , فإن كانت قيمته 1 أي أن القيمة المقروءة أكبر من 127 فيجب طرح القيمة 128 , أما إذا كانت القيمة المقروءة أصغر و تساوي 127 فهذا يعني أن البت الأخير يساوي 0 , و في هذه الحالة يجب إضافة القيمة 128 . إلى هنا نكون قد صححنا الخطأ الناتج عن العاكس الداخلي للخط الأخير In7 , و لكن يبقى أن نصحح الخطأ الناتج عن قراءة النصف الأدنى للمداخل D0 – D3 على الخطوط العليا In4 – In7 , و يتم هذا التصحيح بإزاحة البتات المقابلة أربع خانات نحو اليمين :

(D3D2D1D0 0000) → (0000 D3D2D1D0)

و عملية الإزاحة هذه تكافئ التقسيم على 16 , و هذا ما نفعله في السطر 40 .

يبين الشكل التالي نفس الدارة السابقة , ولكن هذه المرة يتم عكس المدخل الأخير عن طريق عاكس خارجي . إن خضوع المدخل الموصول مع الخط In7 إلى عمليتي عكس (أولاً عن طريق العاكس الخارجي ثم عن طريق العاكس الداخلي) سيؤدي إلى تأخير زمني بسيط على الإشارة المطبقة على هذا المدخل مقارنة مع بقية المداخل . إن هذا التأخير الزمني أقل من أن يسبب أية مشكلة في التطبيقات الاعتيادية .



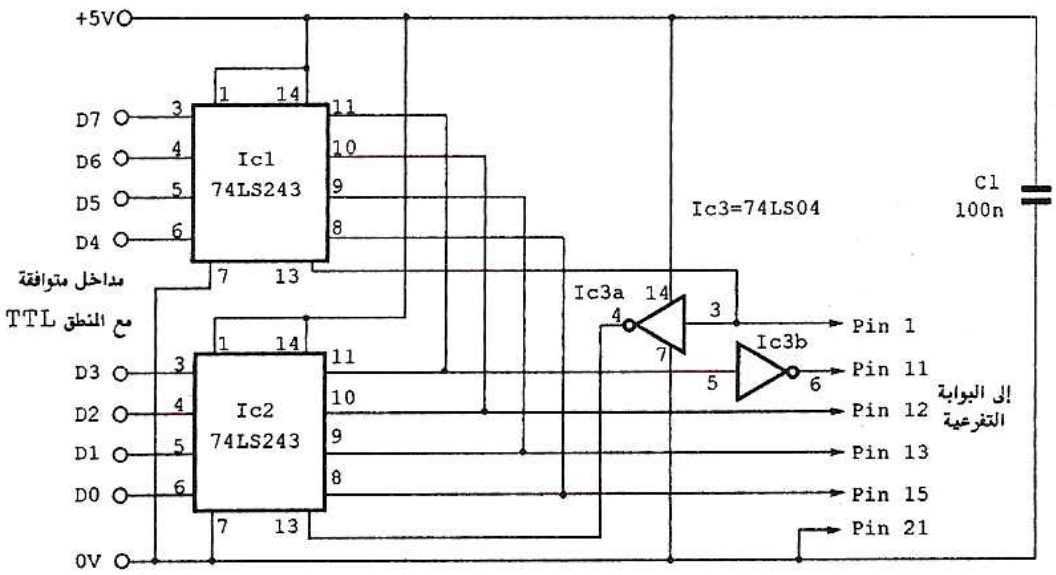
الشكل 11 : بوابة دخل رقمية ثمانية البتات تستخدم البتات 4 حتى 7 مع عكس البت 7 عتادياً

إن البرنامج الذي يقوم بقراءة المعطيات باستخدام هذه الدارة بسيط جداً , لأننا لا نحتاج تصحيح البت الأخير المعكوس , يتم فقط تصحيح النصف الأدنى من المعطيات بسبب الإزاحة بالتقسيم على 16 . و البرنامج هو التالي :

```
   REM PROG TO READ IN BYTE ON BITS 3 TO 6
   ( Hardware Inversion)
   10  OUT &H37A , 1
   20  LSN = INP ( &H379) AND 240
30  LSN = LSN/16
40  OUT &H37A , 0
50  MSN = INP ( &H379) AND 240
60  BYTE = LSN + MSN
70  PRINT BYTE
```

8.5. البتات 3,4,5,7 :

إن استخدم In3,In4,In5,In7 لتنفيذ بوابة الدخل يعتبر تعقيداً لا لزوم له , و لكن مع ذلك فإن البعض يفضل استخدام هذه الخطوط و ترك الخط السادس ليتسنى لهم استغلاله كخط مقاطعة (interrupt Line) كما تفعل ذلك الطابعة . يبين الشكل التالي دارة المواجهة في هذه الحالة حيث تستخدم البوابة في هذه الحالة بشكل مشابه للطابعة .



الشكل **12 : بوابة دخل رقمية ثمانية البتات مع تجنب استخدام البت 6**

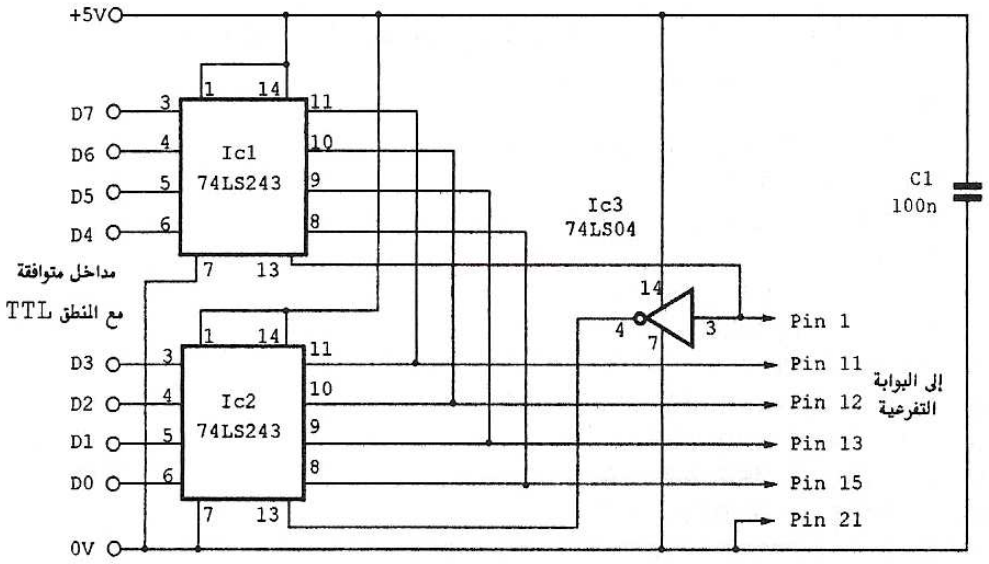
يتم في هذه الدارة عكس المدخل السابع بواسطة عاكس خارجي , الأمر الذي يوفر علينا بعض التعليمات . يقوم البرنامج التالي بقراءة البوابة و إجراء التصحيحات اللازمة , ومن ثم يطبع النتيجة على الشاشة :

```
10  REM PROG TO READ IN BYTE ON BITS 3 TO 6
 ( Hardware Inversion)
 20  OUT &H37A , 1
 30  LSN = INP ( &H379) AND 184
40  D7 = LSN AND 56
50  D3TOD5 = LSN AND 56
60  D7 =D7/16
70  D3TOD5 = D3TOD5 / 8
80  LSN= D3TOD5 + D7
90  OUT &H37A , 0
100 MSN = INP ( &H379) AND 184
110 D7 = MSN AND 128
120 D3TOD5 = MSN  AND 56
130 D3TOD5 = D3TOD5 * 2
140 MSN = D3TOD5 +D7
150 BYTE = LSN + MSN
160 PRINT BYTE
```

يتم في السطر20 اختيار النصف الأدنى للمداخل D3 – D0 , ومن ثم تتم قراءة هذا النصف بعد حجب البتات الغير لازمة في السطر 30 (184=&B1011 1000) البتات الغير محجوبة هي التي تقابل الـ 1 و بالتالي هي D7,D5,D4,D3 و هي البتات التي تقابل المداخل المستخدمة) يتم في السطر التالي حساب قيمة البت السابع و تخزينه في المتحول D7 . يتم نفس العمل بالنسبة للبتات الثالث و الرابع و الخامس و يتم تخزين الناتج في المتحول "D3 TO D5" , ثم يتم

تصحيح القيم بالنسبة لكلا المتحولين السابقين و ذلك بتقسيم D7 على 16 و تقسيم D3 TO D5 على 8 . بإضافة ناتج العمليتين السابقتين نحصل على قيمة النصف الأدنى للمداخل و يتم تخزين الناتج في المتحول "LSN" . بنفس الطريقة يتم اختيار النصف الأعلى للمداخل في السطر 90 , ثم تتم قراءة هذا النصف بعد حجب البتات غير اللازمة . يتم بعد ذلك حساب قيمة البت الأخير و الذي لا يحتاج في هذه الحالة لأي تصحيح ثم يتم حساب قيمة البتات "D3 TO D5" و يتم تصحيح قيمة هذه البتات بضربها بـ 2 , و بإضافة قيمة D7 و القيمة الصحيحة لـ D3 TO D5 نحصل على قيمة النصف الأعلى للمداخل . تتم في السطر 150 إضافة قيمة النصف الأدنى إلى النصف الأعلى و تخزين النتيجة في المتحول "Byte" , ثم يتم بعد ذلك إظهار هذه القيمة على الشاشة .

و إذا أردنا استخدام البتات الثالث و الرابع و الخامس و السابع , و لكن دون تزويد الدارة بعاكس خارجي عندها نحصل على الدارة التالية :



الشكل 13 : بوابة دخل رقمية ثمانية البتات مع تجنب استخدام عاكس خارجي للبت 7

هذه الدارة تشابه تماماً الدرة السابقة , والاختلاف الوحيد هو أن الدارة لا تحتوي على عاكس المدخل Pin11=In7 . و في هذه الحالة يجب إجراء تصحيح على قيمة البت السابع . ويبين البرنامج التالي كيفية قراءة و حساب المداخل :

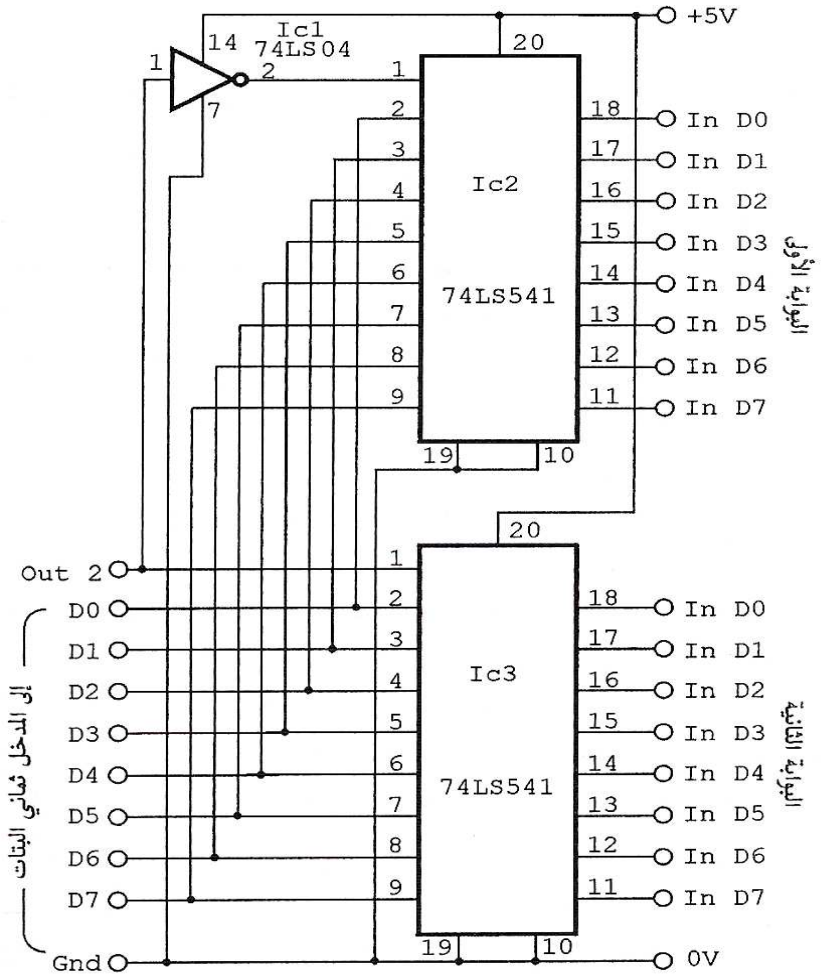
```
REM PROG TO READ IN BYTE ON BITS 3,4,5 nd 7
( Hardware Inversion)
20  OUT &H37A , 1
30  LSN = INP ( &H379) AND 184

40  D7 = LSN AND 128
50  IF D7 > 127 Then D7=0 Else D7=128
60  D3TOD5 = LSN AND 56
70  D7 =D7/16
80  D3TOD5 = D3TOD5 / 8
90  LSN= D3TOD5 + D7
100  OUT &H37A , 0
```

```
110 MSN = INP ( &H379) AND 184
120 D7 = MSN AND 128
130 If D7 > 127 Then D7=0 Else D7=128
140 D3TOD5 = MSN  AND 56
150 D3TOD5 = D3TOD5 * 2
160 MSN = D3TOD5 +D7
170 BYTE = LSN + MSN
180 PRINT BYTE
```

يتشابه هذا البرنامج إلى حد كبير مع البرنامج السابق و يكمن الاختلاف الوحيد في السطرين 50 و 130 , حيث يتم في هذين السطرين عكس قيمة البت السابع .

8.6. زوج من المداخل :

ليس صعباً تحقيق زوج من المداخل كل منها بعرض 8 بتات باستخدام خطوط المصافحة للبوابة التفرعية ,يحتاج الأمر فقط استخدام خرج مصافحة إضافي لكي يقوم بعملية الانتخاب .يبين الشكل (14) إحدى الطرق الممكنة لتحقيق عملية الانتخاب الإضافية .و لا يمكن وصل الدارة مباشرة مع البوابة التفرعية ,بل يجب استخدام إحدى بوابات الدخل المنفذة سابقاً .



الشكل (14):بوابة دخل رقمية بعرض بايتين

ترتكز هذه الدارة على زوج من الشريحة 74LS541 و التي هي عبارة عن عازل ثلاثي الحالات بثمانية بتات .يتم إدخال البايت الأول عن طريق مداخل الشريحة IC2 (port 1) و البايت الثاني عن طريق مداخل الشريحةIC3 (port 2) , و يقوم خرج كل منهما بقيادة مداخل إحدى بوابات الدخل السابقة التنفيذ .يتم تحقيق انتخاب إحدى الشريحتين باستخدام out 2 فعندما تكون بقيمة 0 فإننا نؤهل الشريحة IC3 و بالتالي تنتخب البوابة الثانية port 2 و عندما تكون بقيمة 1 فإننا نؤهل الشريحة IC2 و بالتالي تنتخب البوابة الأولى port 1 .أن وجود العاكس في هذه الدارة يحقق لنا مبدأ الانتخاب العاكس.أي أن انتخاب إحدى البوابتين سيؤدي حتماً إلى تعطيل البوابة الأخرى .

تحتوي الشريحة المستخدمة في داخلها على بوابة NOR بمدخلين ,هنا المربطان pin 1,pin 2 و يتم تأهيل هذه الشريحة عندما يوصل المربطين السابقين إلى الصفر المنطقي و لذلك يمكن التحكم بانتخاب هذه الشريحة بواسطة إشارتين . و لهذه الدارة خط واحد للتحكم بتأهيل أو تعطيل الشريحة , و لذلك يتم وصل أحد مربطي التحكم إلى الأرضي بشكل دائم , بينما يتم التحكم بتأهيل الشريحة عن طريق المربط pin 1 . و من الممكن عكس الأدوار ببساطة .

من الواضح أن طريقة كتابة البرنامج الذي يقوم بالقراءة من إحدى البوابتين (port 1 أو port 2) تعتمد على بوابة الدخل المستخدمة . لنفرض مثلاً أننا نستخدم الدارة المبينة في الشكل (1-12). في هذه الحالة البرنامج التالي لقراءة البوابة الأولى port 1 و طباعة النتيجة على الشاشة .

```
5     REM PROG TO READ PORT 1
10    OUT &H37A , 5
20    LSN = INP(&H379) AND 240
30    LSN = LSN/16
40    OUT &H379 , 4
50    MSN = INP(&H379) AND 240
60    BYTE = MSN + LSN
70    PRINT BYTE
```

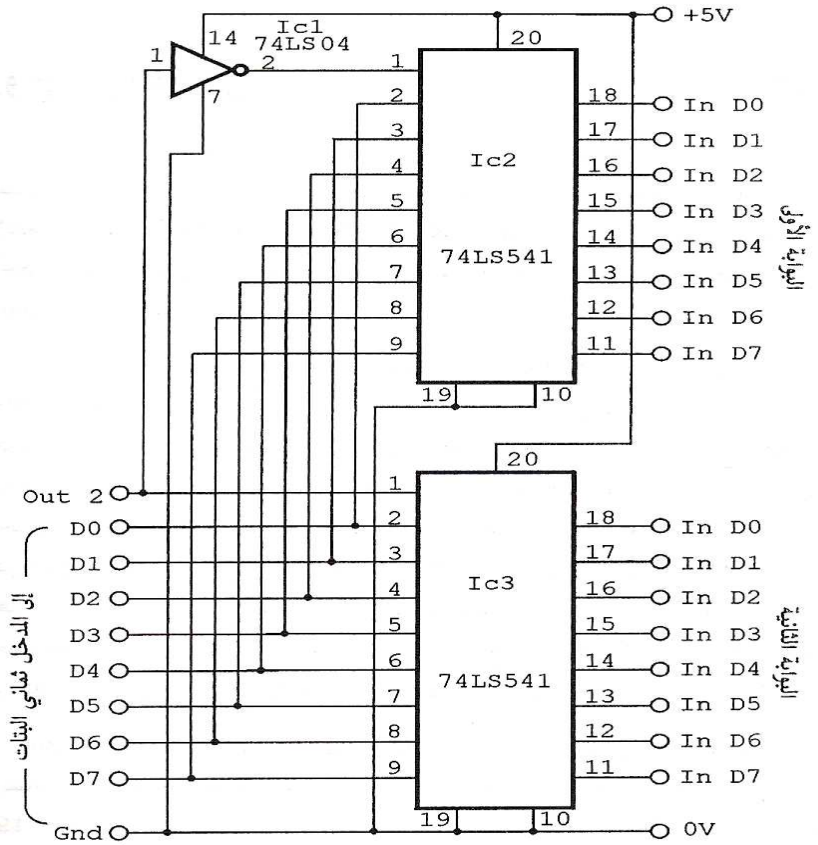
هذا البرنامج هو نفسه البرنامج الذي يقوم بقراءة بوابة الدخل الأساسية مع فارق وحيد في السطرين 10,40 حيث يتم إخراج قيم تجعل المخرج out 2=1 و بهذا الشكل نختار البوابة الأولى و ليس الثانية .

و إذا أردنا قراءة البوابة الثانية فعلينا إخراج القيمة 0 على الخط out 2 و البرنامج التالي يقوم بذلك .

```
5     REM PROG TO READ PORT 2
10    OUT &H37A , 1
20    LSN = INP(&H379) AND 240
30    LSN = LSN/16
40    OUT &H379 , 0
50    MSN = INP(&H379) AND 240
60    BYTE = MSN + LSN
70    PRINT BYTE
```

يبين الشكل (15) دارة لتحقيق زوج من بوابات الدخل باستخدام الشريحة 74LS245 و هي رخيصة و سهلة الاستخدام و تستخدم كما لو كانت عازلاً ثلاثي الحالة , ويتم استخدامها لتمرير

المعطيات في اتجاه وحيد .و يتم التحكم بتأهيل كل عازل عن طريق المربط pin 19 . و هي تكافئ وظيفياً الدارة السابقة و بالتالي تعمل معها البرامج السابقة نفسها.



الشكل (15): بوابة دخل رقمية أخرى بعرض بايتين

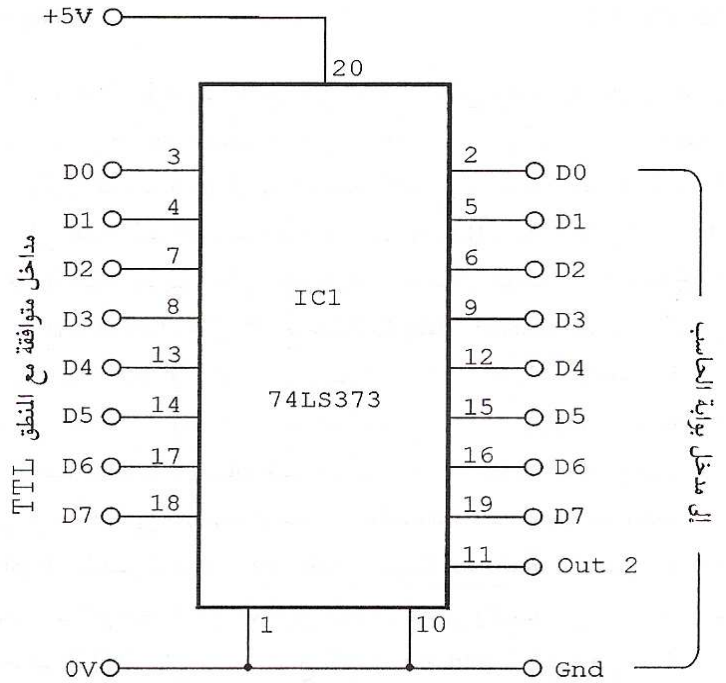
8.7. مسك البايتات :

إن قراءة البايت الكامل على دفعتين متتاليتين قد يسبب مشكلة إذا كانت سرعة تغيير المعطيات كبيرة نوعاً ما .إذا طرأ تغيير على قيمة الدخل بعد قراءة النصف الأول للبايت و قبل قراءة النصف الثاني فإن النتيجة ا. يمكن التغلب على هذه المشكلة باستخدام ماسك latch لمسك كامل البايت في لحظة ما , و من ثم تتم قراءة هذا البايت على دفعتين . إن مسك المعطيات لا يحتاج إلا إلى ماسك بثمانية بتات octal latch و إلى مخرج مصافحة إضافي للتحكم به .يبين الشكل (16) إحدى الطرق لاستخدام الماسك . يتم في هذه الدارة استخدام الدارة المتكاملة 74LS373 و هي عبارة عن ماسك شفاف ثماني البتات . عندما يكون مربط التحكم pin11 لهذا الماسك على المنطق المرتفع فإن مخارج الماسك تعكس ما هو موجود على مداخله أما إذا مربط التحكم pin11 لهذا الماسك على المنطق المنخفض فإن الجبهة الهابطة تتسبب بتجميد المعطيات الموجودة على مداخل الماسك لتظهر و تثبت على مخارجه .و يتم في نفس الشكل التحكم بالماسك عن طريق خرج المصافحة out2 كما يمكن أن تتم عملية التحكم باستخدام أي من مخارج المصافحة الأخرى .

يقوم البرنامج التالي بمسك المعطيات و من ثم تتم قراءتها ,بافتراض أن دارة الشكل (1-12) هي المستخدمة لإنجاز بوابة الدخل .

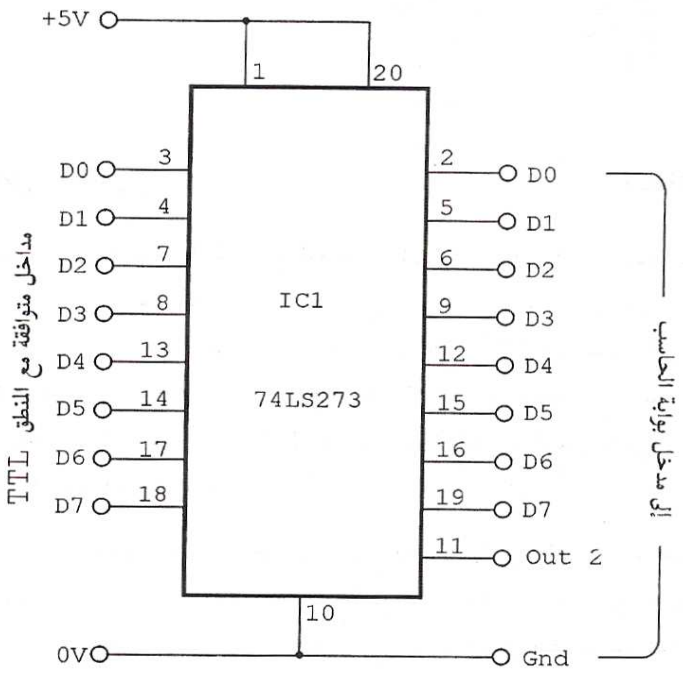
```
10    REM PROG TO READ BYTE VIA DATA LATCH(74LS373)
20    OUT &H37A , 0
```

```
30      OUT &H37A , 4
40      OUT &H37A , 0
45      FOR DELAY = 1 TO 30000 : NEXT DELAY
50      OUT &H37A , 1
60      LSN = INP(&H379) AND 240
70      LSN = LSN/16
80      OUT &H37A , 0
90      MSN = INP(&H379) AND 240
100     BYTE = MSN + LSN
110     PRINT BYTE
```



الشكل(16): استخدام الشريحة 74LS373 كماسك للمعطيات

يتمثل التغيير الوحيد في هذا البرنامج في الأسطر 20 حتى 40 و التي يتم من خلالها إرسال نبضة موجبة على المخرج out 2 , و هذه النبضة هي التي تقوم بعملية المسك عند انتهائها . و في الحقيقة السطر 40 هو الذي يتسبب في نقل حالة الخرج out2 من المنطق المرتفع إلى المنطق المنخفض و بالتالي يتم مسك المعطيات في هذه اللحظة . و يعطي السطر 45 تأخيراً زمنياً و لا يقدم هذا السطر فائدة في البرامج الحقيقية , و تم وضعه هنا ليتسنى لنا تغير قيمة الدخل قبل القيام بعملية القراءة و بالتالي رؤية أثر الماسك .



الشكل (17): استخدام الشريحة 74LS273 كماسك للمعطيات

يبين الشكل (17) دارة بديلة لاستخدام الماسك .ترتكز هذه الدارة على الشريحة 74LS273 و التي هي عبارة عن قلاب ثماني البتات من النوع D . تتم عملية نقل المعطيات من المداخل إلى المخارج و مسكها بإرسال جبهة صاعدة على المربط pin11 . و لا يمكن أن تكون هذه الشريحة شفافة و إنما يمكن فقط نقل المعطيات من المداخل إلى المخارج و تثبيتها على المخارج كلما أر سلنا جبهة صاعدة .

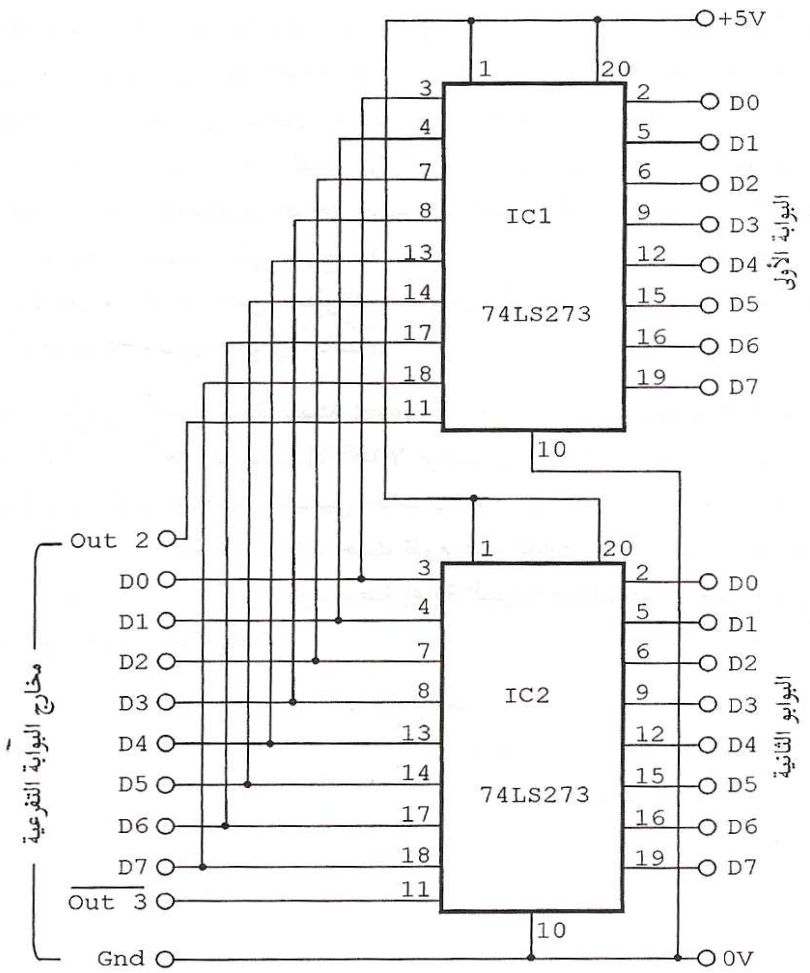
يقوم البرنامج التالي بمسك ثم قراءة المعطيات الممسوكة .يطابق هذا البرنامج تماماً سابقه و لكن يكمن الاختلاف الوحيد أن النبضة المرسلة على المخرج out 2 هي سالبة و ليست موجبة .

```
10    REM PROG TO READ BYTE VIA DATA LATCH(74LS273)
20    OUT &H37A , 4
30    OUT &H37A , 0
40    OUT &H37A , 4
45    FOR DELAY = 1 TO 30000 : NEXT DELAY
50    OUT &H37A , 1
60    LSN = INP(&H379) AND 240
70    LSN = LSN/16
80    OUT &H37A , 4
90    MSN = INP(&H379) AND 240
100   BYTE = MSN + LSN
110   PRINT BYTE
```

8.8. زوج من المخارج :

إن وجود ثمانية خطوط لمخارج المعطيات (D0-D7) في البوابة التفرعية يشكل بوابة خرج جاهزة .يتم استخدام بوابة الخرج هذه بكتابة القيمة المراد إخراجها على هذه الخطوط أي بكتابة القيمة على أحد العناوين 378 &H أو 278 &H (أي LPT 1,LPT 2) .بعد كتابة القيمة ستعكس حالة خطوط خرج المعطيات D0-D7 القيمة المكتوبة , و هذه الخطوط متوافقة مع المنطق TTL .

إن استخدام خطوط خرج المعطيات D0-D7 لتشكيل بوابتي خرج يعتبر أمراً غاية في السهولة و لا يحتاج إلا إلى استخدام دارتي ماسك ,حيث يتم التحكم بكل منهما بواسطة مخرج مصافحة مستقل .



الشكل (18) :بوابة خرج يعرض بايتين

يبين الشكل (18) إحدى الطرق الممكنة لإنجاز ذلك .تستخدم هذه الدارة شريحتي 74LS273 و التي هي عبارة عن قلاب D يستخدم كماسك .

لكتابة بايت على البوابة الأولى على خرج Port1 مثلاً تتم أولاً كتابة هذا البايت على العنوان المحدد للبوابة التفرعية المناسبة . يتم بعد ذلك إخراج القيمة صفر على خط التحكم الخاص ,ثم يتم إخراج القيمة واحد على هذا الخط .إن انتقال خط التحكم هذا من الصفر إلى الواحد يتسبب

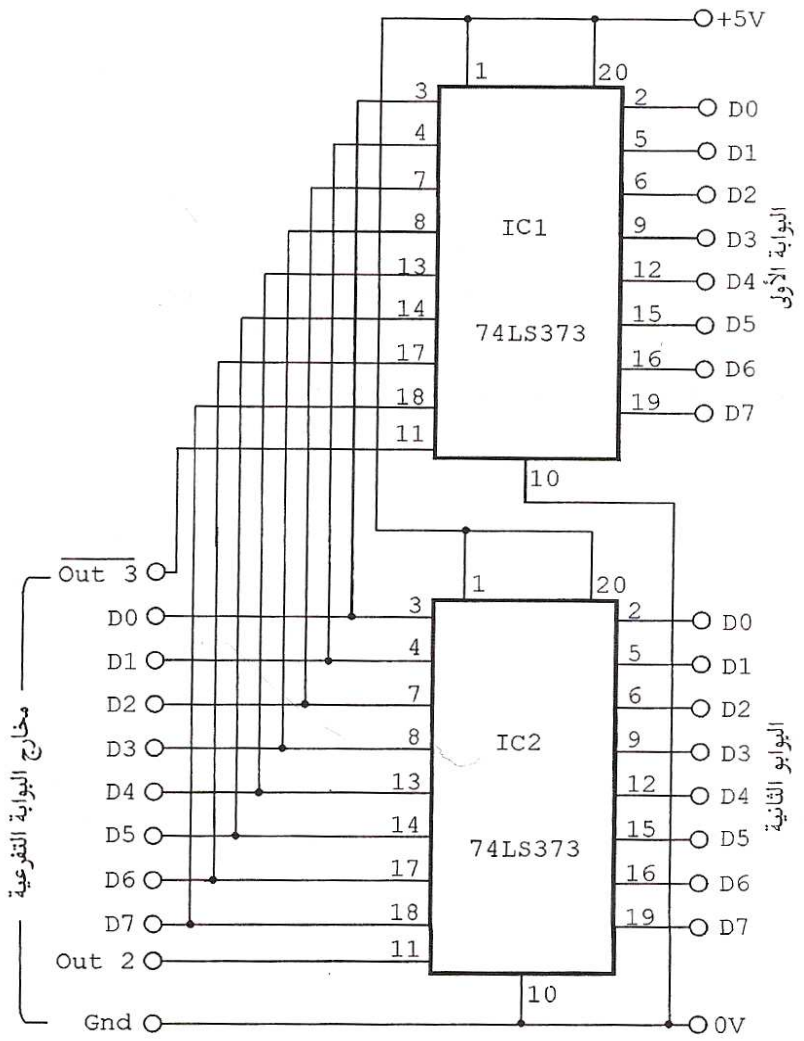
بنقل المعطيات من دخل الماسك إلى خرجه و تثبيتها , و بذلك تكون القيمة التي كتبت على البوابة التفرعية قد ظهرت على خرج الماسك و ثبتت .لكتابة بايت آخر على البوابة الثانية يتم القيام بنفس الخطوات السابقة و لكن في هذه المرة يتم استخدام مخرج مصافحة مختلف .
لقد تم وصل مدخل إعادة التهيئة لكل من الشريحتين IC 1&IC 2 مع التغذية 5V+ و بالتالي فقد ألغي عملهما , و يجب استخدام مداخل إعادة التهيئة لعدم رؤية قيم عشوائية و يتم ذلك بواسطة دارة R-C حيث تقوم هذه الدارة بتوليد نبضة سالبة عند بدء تشغيل الحاسب .قد نحتاج لأن تكون نبضة إعادة التهيئة طويلة نوعاً ما ,حتى ينتهي الحاسب من الإقلاع .
إن برنامج كتابة قيمة ما في إحدى البوابتين سهل جداً,فالبرنامج التالي على سبيل المثال سيقوم بكتابة القيمة 123 في البوابة الأولى

```
10      REM PROG TO WRITE  BYTE OF DATA TO PORT 1 of
(74LS273)
20      OUT &H37A , 4
30      OUT &H378 , 123
40      OUT &H37A , 0
50      OUT &H37A , 4
```

يقوم السطر 20 بإسناد القيم الأولية المناسبة لمخارج التحكم ,أي جعل المخرجين out 2, out 3على المنطق المرتفع . مع العلم أن out 3 معكوس داخلياً و بالتالي يجب كتابة الصفر على هذا المخرج لنحصل على الواحد . في السطر 30 تتم كتابة القيمة المراد إخراجها و هي 123 . يولد السطران التاليان نبضة سالبة على المخرج out 2 ,بينما يتم إبقاء out 3 على حالته الابتدائية .
و عند انتقال حالة الخط out 2 من الصفر إلى الواحد فسيظهر بايت المعطيات على خرج البوابة port 1 .إن كتابة قيمة ما في بوابة الخرج الثانية port 2 يشبه تماماً الكتابة في البوابة الأولى port 1 .يقوم البرنامج التالي بكتابة القيمة 231 في بوابة الخرج الثانية port 2 .

```
10      REM PROG TO WRITE  BYTE OF DATA TO PORT 2
20      OUT &H37A , 4
30      OUT &H378 , 231
40      OUT &H37A , 12
50      OUT &H37A , 4
```

يكمن الاختلاف الوحيد بين هذا البرنامج و سابقه في السطرين 40,50 , حيث يتم توليد نبضة سالبة على المخرج out 3 بينما يتم المحافظة على المخرج out 2 في الحالة الابتدائية .
يمكن أن يكون الشكل (19) بديلاً عن دارة الشكل (18) فهي تشابهها تماماً. إلا أنه يتم في هذه الدارة يتم استخدام الماسك 74LS373 (الماسك الشفاف ثماني البتات) . و الفرق الأساسي بين الدارتين أن خطوط التحكم يجب أن تكون في حالة المنطق المنخفض في الأحوال الطبيعية , و عندما نريد مسك المعطيات يتم تغيير حالة خط التحكم إلى المنطق المرتفع .



الشكل (19) : بوابة خرج أخرى بعرض بايتين

من الطبيعي أننا سنكون قادرين على إنجاز ثلاث أو أربع بوابات خرج ثمانية البتات إذا توفرت لنا مخارج المصافحة الإضافية, و هذا ما يجعل من الصعب إنجاز بوابات الدخل بسبب عدم توفر مخارج المصالحة الكافية .يفضل في الحالات العملية اللجوء إلى استخدام ممرات التوسعة في حال وجود عدد كبير من المخارج .

9. إرسال واستقبال الملفات وبروتوكول التراسل بين المرسل و المستقبل:

إذا أردنا كتابة تطبيق معين من أجل تراسل الملفات بين حاسبين فيجب مراعاة عدة أمور , ولابد من القول أن هذان البرنامجان يعملان مع بعضهما البعض , و أن طريقة عمل أحدهما تعتمد على طريقة عمل الآخر . إن الجزء الأهم في برنامج الاستقبال هو إجرائية استقبال بايت وحيد receive1byte , وكذلك الأمر فإن الجزء الأهم في برنامج الإرسال هو إجرائية إرسال بايت واحد فقط send1byte .

يتكون برنامج الاستقبال بشكل أساسي من حلقة تكرارية يتم فيها استدعاء إجرائية استقبال البايتات بشكل متكرر , وكذلك الأمر بالنسبة لبرنامج الإرسال فهو أيضاً يتكون من حلقة تكرارية يتم فيها استدعاء إجرائية إرسال البايتات بشكل متكرر.

يستعمل برنامج ***الإرسال*** الإجرائيات والتوابع التالية :

procedure senderisready:

تقوم هذه الإجرائية بتفعيل خط المصافحة out0 , أشارة إلى أن المرسل جاهز لإرسال قيم ما .

Procedure senderisnotready:

تقوم هذه الإجرائية بتعطيل خط المصافحة out0 , أشارة إلى أن المرسل غير جاهز لإرسال أية قيمة .

Function receiverisready:Boolean

يقوم هذا التابع بقراءة و تفحص حالة خط الدخل in7 , فإذا كانت فعالة فيعيد القيمة true , وهذا بدوره يشير إلى أن المستقبل أصبح جاهزاً لاستقبال قيمة ما , وإن كانت حالة هذا الخط غير فعالة فإنه يعيد القيمة false , وهذا ما يشير إلى أن المستقبل غير جاهز .

Function receiverisnotready:Boolean

يقوم هذا التابع بقراءة و تفحص حالة الخط للدخل in7 فإن كانت غير فعالة فيعيد القيمة true , وإن كانت فعالة فيعيد القيمة false , يعمل هذا التابع بشكل معاكس للتابع السابق .

يستعمل برنامج ***الاستقبال*** من جهته الإجرائيات و التوابع التالية :

Procedure receiverisready:

تقوم هذه الإجرائية بتفعيل خط المصافحة out0 , أشارة إلى أن المستقبل جاهز لاستقبال قيمة ما .

Procedure receiverisnotready:

تقوم هذه الإجرائية بتعطيل خط المصافحة out 0 , إشارة إلى أن المستقبل غير جاهز لاستقبال أية قيمة .

Function senderisready:

يقوم هذا التابع بقراءة و تفحص حالة الخط للدخل in7 فإن كانت فعالة فيعيد القيمة true ,وهذا بدوره يشير إلى أن المرسل أصبح جاهزاً لإرسال قيمة ما ,

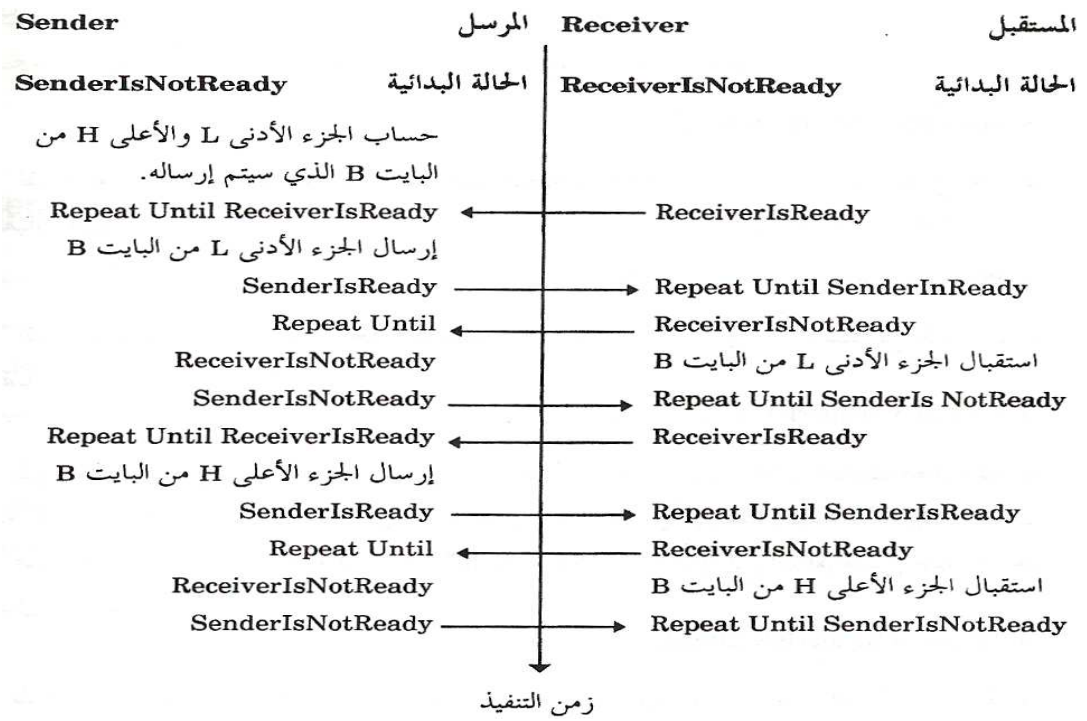
وإن كانت حالة هذا الخط غير فعالة فيعيد القيمة false ,و هذا يشير إلى أن المرسل غير جاهز لإرسال أية قيمة .

Function senderisnotready:

يقوم هذا التابع بقراءة و تفحص حالة الخط للدخل in7 , إن كانت حالة هذا الخط غير فعالة فيعيد القيمة true ,و إن كانت فعالة فيعيد القيمة false , يعمل هذا التابع بشكل معاكس للتابع السابق .

ومن الممكن أن نلاحظ بسهولة التشابه و التناظر في التسميات المعتمدة لكلا البرنامجين ,وهذا في الحقيقة أمر معتمد , حيث نلاحظ مثلاً أن الاسم receiverisready هو إجرائية في برنامج الإستقبال بينما هو تابع في برنامج الإرسال . فبالطبع يختلف الدور الذي يلعبه هذا البرنامج الجزئي في كلا البرنامجين, ففي برنامج الاستقبال يقوم بتفعيل خط المصافحة out0 للإشارة إلى جاهزيته للاستقبال , فيما يقوم في برنامج الإرسال بقراءة و تفحص حالة الخط للدخل In7 لمعرفة مدى جاهزية المستقبل .

والشكل التالي هو خوارزمية كل من إجرائيتي الإرسال و الإستقبال , وهي تعبر بشكل فعلي عن بروتوكول التراسل المعتمد.



الشكل 20 : بروتوكول التراسل و التزامن بين المرسل و المستقبل

10. البرنامج التطبيقي لنقل ملف ما على البوابة التفرعية :

لإرسال الملفات فيمكن أن نتبع التقنية الأخرى التالية :
أولاً يجب إرسال الملف كما هو فيجب أن يصل بنفس الإسم .
يتم إرسال عدد أحرف الإسم فيعطي إشارة للجهاز الآخر بأنه سيرسل له عدد أحرف اسم الملف .
أرسال الإسم حرفاً تلو الآخر على أساس حلقة من عدد أحرف الإسم حيث كل حرف يرسل على جزئين (نرسل كل 4 بتات سوية) .
إن عملية التزامن مع الطرف آخر هي أمر مهم جداً يجب أخذه بعين الإعتبار .
بعدها يتم أرسال عدد بايتات الملف إي حجم الملف كبايت , على أربع بايتات . فمثلاً إذا كان حجم الملف 3400 بايت , فسنرسلها على أربع بايتات
ندخل بحلقتين كي يقوم الطرفين بالخلقة (على حجم إرسال الحروف للملف) , فنرسل d0-d3 إلى المرسل .
إما المستقبل فسيستقبل على state من s3 إلى s6 , باعتبار عدم وجود قلب في البتات .
بالإضافة إلى كل ذلك يلزمني موضوع التزامن كإشارة متعارف عليها من المرسل و المستقبل (خرج مع دخل) , فنستخدم مثلاً في الإرسال (d5) و في الإستقبال (s7') من أجل التزامن .

وهكذا يكون موضوع التزامن هو الأمر الأهم حتى تتم العملية بشكلها الصحيح , فعندما يرسل الطرف الأول إشارة تزامن فيعرف عندها الطرف الثاني بوجود شيء ما فيقوم بقراءته .

this program Transfers files over the parallel interface:

```
/*== Add include files ==============================================*/
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <setjmp.h>
#include <string.h>
#ifdef __TURBOC__ /* Turbo C compiler? */
#include <dir.h> /* Include directory functions */
#include <ctype.h> /* for toupper() */
#endif
/*== Type definitions ===============================================*/
typedef unsigned char BYTE; /* Define a byte */
typedef unsigned int WORD; /* Create a WORD */
typedef struct { /* Block transfer header */
BYTE Token;
unsigned int Len;
} BHEADER;
/*== Assembler module functions =====================================*/
extern void IntrInstall( int far * escape_flag,
WORD far * timeout_count );
extern void IntrRemove( void );
extern void EscapeDirect( int disconnect );
/*== Constants ======================================================*/
#define ONESEC 18 /* One second */
#define TENSEC 182 /* Ten seconds */
#define TO_DEFAULT TENSEC /* Time out default value */
#define TRUE ( 0 == 0 )
#define FALSE ( 0 == 1 )
#define MAXBLOCK 4096 /* 4K maximum block size */
/*-- Constants for transfer protocol -------------------------------*/
#define ACK 0x00 /* Acknowledge */
#define NAK 0xFF /* Non-Acknowledge */
#define MAX_TRY 5 /* Maximum number of tries */
/*-- Tokens for communication between sender and receiver -----------*/
#define TOK_DATSTART 0 /* Start of file data */
#define TOK_DATNEXT 1 /* Next file data block */
#define TOK_DATEND 2 /* End file data transfer */
#define TOK_ENDIT 3 /* End program */
#define TOK_ESCAPE 4 /* <Esc> pressed on remote computer */
/*-- Codes for LongJump call ----------------------------------------*/
#define LJ_OKSENDER 1 /* All data sent successfully */
#define LJ_OKRECD 2 /* All data received successfully */
#define LJ_TIMEOUT 3 /* Time out: No response */
#define LJ_ESCAPE 4 /* <Esc> pressed on local computer */
#define LJ_REMESCAPE 5 /* <Esc> pressed on remote computer */
#define LJ_DATA 6 /* Communication error */
#define LJ_NOLINK 7 /* No link */
#define LJ_NOPAR 8 /* No interface */
#define LJ_PARA 9 /* Invalid call parameters */
/*== Macros =========================================================*/
/*-- The lower three bits of the input register are not allocated ---*/
/*-- and can be set to 1 or 0 by the computer, so they are masked ---*/
/*-- by GetB(). ---*/
#ifdef __TURBOC__ /* Turbo C compiler? */
#define GetB() ( inportb( InpPort ) & 0xF8 )
#define PutB( Was ) outportb( OutPort, Was )
#define DIRSTRUCT struct ffblk
#define FINDFIRST( path, buf, attr ) findfirst( path, buf, attr )
```

```
#define FINDNEXT( buf ) findnext( buf )
#define DFILENAME ff_name
#else /* No --> Microsoft C */
#define GetB() ( inp( InpPort ) & 0xF8 )
#define PutB( Was ) outp( OutPort, Was )
#define DIRSTRUCT struct find_t
#define FINDFIRST( path, buf, attr ) _dos_findfirst(path, attr, buf)
#define FINDNEXT( buf ) _dos_findnext( buf )
#define DFILENAME name
#endif
#ifdef MK_FP /* Macro MK_FP already defined? */
#undef MK_FP /* Yes --> Undefine macro */
#endif
#define MK_FP(seg,ofs) ((void far *) ((unsigned long) (seg)<<16|(ofs)))
/*== Global variables ==============================================*/
int InpPort; /* Input port address */
int OutPort; /* Output port address */
int Escape = 0; /* <Esc> not pressed */
WORD Timeout = TO_DEFAULT; /* Selected time out value */
WORD TO_Count; /* Counter for time out */
jmp_buf ReturnToEnd; /* Return address for end */
BYTE *BlockBuf; /* Buffer for passing a block */
FILE *FiVar = NULL; /* File variable for file being processed */
/*********************************************************************/
/* GetPortAdr: Initializes a parallel interface's port addresses in */
/* the global variables INPPORT and OUTPORT. */
/* Input : LPTNUM = Parallel interface number (1-4) */
/* Output : TRUE if interface is valid */
/* Global vars. : InpPort/W, OutPort/W */
/* Info : The base addresses of up to 4 parallel interfaces */
/* lie in the four memory words starting at */
/* 0040:0008H. */
/*********************************************************************/
int GetPortAdr( int LptNum )
{ /* Read port addresses from BIOS variable segment */
OutPort = *( WORD far * ) MK_FP( 0x0040, 6 + LptNum * 2 );
if ( OutPort != 0 ) /* Interface available? */
{ /* Yes */
InpPort = OutPort + 1; /* Input register address */
return TRUE; /* End error-free */
}
else
return FALSE; /* Error: Interface not available */
}
/*********************************************************************/
/* Port_Init : Initializes the registers needed for transfer. */
/* Input : SENDER = TRUE if sender, FALSE if receiver */
/* Output : TRUE if register initializes successfully */
/* Global vars. : InpPort/R, OutPort/R */
/* Info : The asymmetry (send 00010000, wait for 00000000) */
/* occurs because of signal inversion. Normally the */
/* input and output registers contain the desired */
/* values, but initialization is needed after an */
/* aborted transfer, or when restarting. */
/*********************************************************************/
int Port_Init( int Sender )
{
EscapeDirect( TRUE ); /* Release through Escape time out */
if ( Sender ) /* Device = Sender? */
{
TO_Count = Timeout * 5; /* Start time out counter */
PutB( 0x10 ); /* Send: 00010000b */
while ( ( GetB() != 0x00 ) && TO_Count ) /* Wait for 00000000b */
```

```
;
}
else /* Device = Receiver? */
{
TO_Count = Timeout * 5; /* Start time out counter */
while ( ( GetB() != 0x00 ) && TO_Count ) /* Wait for 00000000b */
;
PutB( 0x10 ); /* Send: 00010000b */
}
EscapeDirect( FALSE ); /* No time out released on Escape */
return ( TO_Count != 0 ); /* End initialization */
}
/*********************************************************************/
/* SendAByte : Sends a byte to the remote computer in two parts, */
/* then checks the result. */
/* Input : B2Send = Byte to be sent */
/* Output : Transfer successful? (0 = error, -1 = O.K.) */
/* Global vars. : Timeout/R, InpPort/R, OutPort/R (in macros) */
/*********************************************************************/
int SendAByte( BYTE B2Send )
{
BYTE RcvdB; /* Received byte */
/*-- Send lower nibble ----------------------------------------------*/
TO_Count = Timeout; /* Initialize time out counter */
PutB( B2Send & 0x0F ); /* Sending, clear BUSY */
while ( ( ( GetB() & 128 ) == 0 ) && TO_Count ) /* Wait for message */
;
if ( TO_Count == 0 ) /* Time out error? */
longjmp( ReturnToEnd, LJ_TIMEOUT ); /* Cancel transfer */
RcvdB = ( GetB() >> 3 ) & 0x0F; /* Bits 3-6 in 0-3 */
/*-- Send upper nibble ----------------------------------------------*/
TO_Count = Timeout; /* Initialize time out counter */
PutB( ( B2Send >> 4 ) | 0x10 ); /* Sending, set BUSY */
while ( ( ( GetB() & 128 ) != 0 ) && TO_Count ) /* Wait for message */
;
if ( TO_Count == 0 ) /* Time out error? */
longjmp( ReturnToEnd, LJ_TIMEOUT ); /* Cancel transfer */
RcvdB = RcvdB | ( ( GetB() << 1 ) & 0xF0 ); /* Bits 3-6 in 4-7 */
return ( B2Send == RcvdB ); /* Byte sent correctly? */
}
/*********************************************************************/
/* ReceiveAByte : Receives a two part byte from remote computer, and */
/* sends returned parts for testing. */
/* Input : None */
/* Output : Received byte */
/* Global vars. : Timeout/R, InpPort/R, OutPort/R (in macros) */
/*********************************************************************/
BYTE ReceiveAByte( void )
{
BYTE LoNib, HiNib; /* Received nibbles */
/*-- Receive and re-send lowest nibble ------------------------------*/
TO_Count = Timeout; /* Initialize time out counter */
while ( ( ( GetB() & 128 ) == 0 ) && TO_Count )/* Wait until BUSY 1 */
;
if ( TO_Count == 0 ) /* Time out error? */
longjmp( ReturnToEnd, LJ_TIMEOUT ); /* Cancel transfer */
LoNib = ( GetB() >> 3 ) & 0x0F; /* Bits 3-6 in 0-3 */
PutB( LoNib ); /* Re-send */
/*-- Receive and re-send highest nibble -----------------------------*/
TO_Count = Timeout; /* Initialize time out counter */
while ( ( ( GetB() & 128 ) != 0 ) && TO_Count )/* Wait until BUSY 0 */
;
if ( TO_Count == 0 ) /* Time out error? */
```

```
longjmp( ReturnToEnd, LJ_TIMEOUT ); /* Cancel transfer */
HiNib = ( GetB() << 1 ) & 0xF0; /* Bits 3-6 in 4-7 */
PutB( ( HiNib >> 4 ) | 0x10 ); /* Re-sending, set BUSY */
return( LoNib | HiNib ); /* Received byte */
}
/*********************************************************************/
/* SendABlock: Sends a data block. */
/* Input : TOKEN = Token for receiver */
/* TRANUM = Number of bytes to be transferred */
/* DPTR = Pointer to buffer containing data */
/* Output : None, jumps immediately to an error handler if */
/* an error occurs. */
/*********************************************************************/
void SendABlock( BYTE Token, int TraNum, void *DPtr )
{
BHEADER header; /* Header for placing tokens and numbers */
BYTE *bptr, /* Pointer to current byte to be sent */
RcvrEscape; /* <Esc> pressed on remote? */
int ok, /* Error flag */
i, /* Loop counter */
try; /* Remaining number of tries */
if ( Escape ) /* <Esc> pressed on local computer? */
{
Token = TOK_ESCAPE; /* Yes --> Send Escape token */
TraNum = 0;
}
/*-- Send header first ---------------------------------------------*/
header.Token = Token; /* Create header */
header.Len = TraNum;
for ( try = MAX_TRY; try; --try ) /* Make MAX_TRY access attempts */
{
ok = TRUE; /* Send on error-free transfer */
for ( bptr = (BYTE *) &header, i = sizeof( header); i; --i )
ok = ok & SendAByte( *bptr++ ); /* Send a byte */
ok = ok & SendAByte( (BYTE) (ok ? ACK : NAK) ); /* Confirmation */
if ( ok ) /* Everything transfer successfully? */
break; /* Yes --> No need to try again */
}
if ( try == 0 ) /* Could the header be sent successfully? */
longjmp( ReturnToEnd, LJ_DATA ); /* No --> Cancel transfer */
if ( Token == TOK_ESCAPE ) /* Was an Escape message sent? */
longjmp( ReturnToEnd, LJ_ESCAPE ); /* Yes --> Cancel transfer */
/*-- Send the data block itself ------------------------------------*/
if ( TraNum ) /* Length > 0? */
{
for ( try = MAX_TRY; try; -- try ) /* Make MAX_TRY attempts */
{
ok = TRUE; /* Send on error-free transfer */
for ( bptr = (BYTE *) DPtr, i = TraNum; i; --i )
ok = ok & SendAByte( *bptr++ ); /* Send byte and read status */
ok = ok & SendAByte( (BYTE) (ok ? ACK : NAK) ); /* Confirmation */
if ( ok ) /* Everything transfer successfully? */
break; /* Yes --> No need to try again */
}
if ( try == 0 ) /* Could data block be sent successfully? */
longjmp( ReturnToEnd, LJ_DATA ); /* No --> Cancel transfer */
}
/*-- Read Escape byte from receiver --------------------------------*/
for ( try = MAX_TRY; try; -- try ) /* Enable tries */
{
RcvrEscape = ReceiveAByte(); /* Read remove Escape status */
if ( RcvrEscape == (BYTE) TRUE || RcvrEscape == (BYTE) FALSE )
break; /* Receive Escape status */
```

```
}
if ( try == 0 ) /* Was the Escape status received? */
longjmp( ReturnToEnd, LJ_DATA ); /* No --> Cancel transfer */
if ( RcvrEscape ) /* <Esc> from remote computer? */
longjmp( ReturnToEnd, LJ_REMESCAPE ); /* Yes --> Cancel transfer */
}
/*********************************************************************/
/* ReceiveABlock: Receives a data block. */
/* Input : TOKEN = Pointer to variable containing tokens */
/* LEN = Pointer to variable containing length */
/* DPTR = Pointer to buffer containing data */
/* Output : None, jumps immediately to an error handler if */
/* an error occurs. */
/* Info : Buffer must be allocated using MAXBLOCK, since block */
/* length cannot usually be anticipated. */
/*********************************************************************/
void ReceiveABlock( BYTE *Token, int *Len, void *DPtr )
{
BHEADER header; /* Header for storing tokens and numbers */
BYTE *bptr; /* Floating pointer for receive buffer */
int ok, /* Error flag */
i, /* Loop counter */
try, /* Remaining number of tries */
EscapeStatus; /* For storing current Escape status */
/*-- Receive header first ----------------------------------------*/
for ( try = MAX_TRY; try; -- try ) /* Make MAX_TRY access attempts */
{
for ( bptr = (BYTE *) &header, i = sizeof(header); i; --i )
*bptr++ = ReceiveAByte( );
if ( ReceiveAByte() == ACK ); /* All bytes received successfully? */
break; /* Yes --> No need to try again */
}
if ( try == 0 ) /* Header received successfully? */
longjmp( ReturnToEnd, LJ_DATA ); /* No --> Cancel transfer */
if ( ( *Token = header.Token ) == TOK_ESCAPE ) /* Sender Escape? */
longjmp( ReturnToEnd, LJ_REMESCAPE ); /* Yes --> Cancel transfer */
/*-- Header was O.K., now receive the data block itself ------------*/
if ( ( *Len = header.Len ) != 0 ) /* No data block? */
{ /* No */
for ( try = MAX_TRY; try; -- try )/* Make MAX_TRY access attempts */
{
for ( bptr = (BYTE *) DPtr, i = header.Len; i; --i )
*bptr++ = ReceiveAByte( );
if ( ReceiveAByte() == ACK ); /* Bytes received successfully? */
break; /* Yes --> No need to try again */
}
if ( try == 0 ) /* Block received successfully? */
longjmp( ReturnToEnd, LJ_DATA ); /* No --> Cancel transfer */
}
/*-- Send current Escape status to the remote computer -------------*/
EscapeStatus = Escape; /* Note status */
for ( try = MAX_TRY; try; -- try ) /* Enable access attempts */
{
if ( SendAByte( (BYTE) (EscapeStatus != 0) ) ) /* Sent? */
break; /* Yes --> No need to try again */
}
if ( try == 0 ) /* Could Escape status be sent? */
longjmp( ReturnToEnd, LJ_DATA ); /* No --> Cancel transfer */
if ( EscapeStatus ) /* <Esc> pressed on this computer? */
longjmp( ReturnToEnd, LJ_ESCAPE ); /* Yes --> Cancel transfer */
}
/*********************************************************************/
/* SendAFile : Sends a file. */
```

```
/* Input : NAME = Pointer to buffer containing filenames */
/* Output : None */
/*********************************************************************/
void SendAFile( char *Name )
{
int Status; /* Send status */
WORD WdsRead; /* Number of bytes read */
unsigned long BytSent; /* Number of bytes sent */
printf( "%-13s", Name );
FiVar = fopen( Name, "rb" ); /* Open file */
SendABlock( TOK_DATSTART, strlen(Name)+1, Name ); /* Send filename */
/*-- Transfer file contents ----------------------------------------*/
BytSent = 0;
do
{
WdsRead = fread( BlockBuf, 1, MAXBLOCK, FiVar ); /* Read block */
if ( WdsRead > 0 ) /* End of block not reached? */
{ /* No */
SendABlock( TOK_DATNEXT, WdsRead, BlockBuf ); /* Send block */
BytSent += WdsRead;
printf( "\r%-13s (%ld)", Name, BytSent );
}
}
while ( WdsRead > 0 );
printf( "\n" );
SendABlock( TOK_DATEND, 0, NULL ); /* End transfer */
fclose( FiVar ); /* Close file */
FiVar = NULL; /* File is closed */
}
/*********************************************************************/
/* ReceiveAFile: Receives a file. */
/* Input : None */
/* Output : The last token received */
/*********************************************************************/
int ReceiveAFile( void )
{
int Status; /* Send status */
WORD LastBlkSize; /* Size of last block */
unsigned long BytSent;
BYTE Token; /* Token received */
int Len; /* Block length received */
char Name[13]; /* Filename */
ReceiveABlock( &Token, &Len, BlockBuf );
if ( Token == TOK_DATSTART )
{
strcpy( Name, BlockBuf );
FiVar = fopen( Name, "wb" ); /* Open (create) file */
printf( "%-13s", Name );
/*-- Receive file contents ----------------------------------------*/
BytSent = 0;
do
{
ReceiveABlock( &Token, &Len, BlockBuf ); /* Receive block */
if ( Token == TOK_DATNEXT ) /* Next data block? */
{ /* Yes */
fwrite( BlockBuf, 1, Len, FiVar ); /* Write block */
BytSent += Len;
printf( "\r%-13s (%ld)", Name, BytSent );
}
}
while ( Token == TOK_DATNEXT );
fclose( FiVar ); /* Close file */
FiVar = NULL; /* File is closed */
```

```
printf( "\n" );
}
return Token; /* Return error status */
}
/*********************************************************************/
/* M A I N P R O G R A M */
/*********************************************************************/
void main( int argc, char *argv[] )
{
DIRSTRUCT SRec; /* Structure for directory search */
BYTE Sender; /* Transfer mode (Send, Receive) */
int sjStatus, /* Longjump code */
LptNum, /* Interface number */
i, /* Loop counter */
DirFound; /* For directory search */
static char *ScnMesg[ 9 ] =
{ "DONE: All files sent successfully.",
"DONE: All files received successfully.",
"ERROR: Time out, remote system not responding.",
"DONE: <Esc> key pressed.",
"DONE: <Esc> key pressed on remote computer.",
"ERROR: Hardware problem (check cable and interface).",
"ERROR: No contact with remote computer.",
"ERROR: Interface you requested not found.",
"ERROR: Invalid or unknown parameters." };
printf( "\n\n  Parallel Interface File Transfer Program\n " );
printf( "Designed by :  Nisrine Jbel, Abir Kalash , Fajr Bahous\n");
printf( "-------------------------------------------------------\n");
if ( strcmp( argv[ 1 ], "/?" ) == 0 ) /* Display syntax only? */
{ /* Yes */
printf( "Syntax: plinkc [/Pn] [/Tnn] [filename]\n" );
printf( " | |\n");
printf( " Interface Time out ");
exit ( 0 );
}
sjStatus = setjmp( ReturnToEnd ); /* Return address to end */
if ( sjStatus ) /* Longjmp called? */
{ /* Yes */
IntrRemove( ); /* Disable interrupt handler */
if ( FiVar ) /* Any files still open? */
fclose( FiVar );
free( BlockBuf ); /* Release allocated buffer memory */
printf( "\n\n%s\n", ScnMesg[ sjStatus - 1 ] );
exit( 0 );
}
BlockBuf = malloc( MAXBLOCK ); /* Generate data buffer */
IntrInstall( &Escape, &TO_Count ); /* Interrupt handler init */
/*-- Set default values and interpret command line ----------------*/
Sender = FALSE; /* Default is Receiver */
LptNum = 1; /* Default is LPT1 */
for ( i = 1; i < argc; i++ )
{
if ( argv[i][0] == '/' ) /* Parameter */
{
switch ( toupper( argv[i][1] ) )
{
case 'T' : Timeout = (atol( &argv[i][2] ) * TENSEC) / 10;
if ( Timeout == 0 )
longjmp( ReturnToEnd, LJ_PARA ); /* Invalid */
break;
case 'P' : LptNum = argv[i][2] - 48; /* Interface */
if ( LptNum == 0 || LptNum > 4 )
longjmp( ReturnToEnd, LJ_PARA ); /* Invalid */
```

```
break;
default : longjmp( ReturnToEnd, LJ_PARA ); /* Unknown params */
break;
}
argv[i][0] = '\0'; /* Clear argument */
}
else /* No parameters, must be a filename */
Sender = TRUE; /* Sender */
}
/*-- Start transfer ------------------------------------------------*/
if ( GetPortAdr(LptNum) == FALSE ) /* Does the interface exist? */
longjmp( ReturnToEnd, LJ_NOPAR ); /* No --> Error */
if ( Port_Init( Sender ) == FALSE ) /* Create link */
longjmp( ReturnToEnd, LJ_NOLINK ); /* Impossible, error */
if ( Sender ) /* Sender? */
{
printf( "Sending over LPT%d:\n\n", LptNum );
/*-- Transfer all data ------------------------------------------*/
for ( i = 1; i < argc; i++ ) /* Execute command line */
{
if ( argv[i][0] != '\0' ) /* Filename? */
{ /* Yes */
DirFound = FINDFIRST( argv[i], &SRec, 0);
while ( !DirFound )
{
if ( SRec.DFILENAME[0] != '.' )
SendAFile( SRec.DFILENAME ); /* Transfer file */
DirFound = FINDNEXT( &SRec );
}
}
}
SendABlock( TOK_ENDIT, 0 , NULL ); /* All files sent? */
longjmp( ReturnToEnd, LJ_OKSENDER );
}
else /* No --> Receiver */
{
printf( "Receiving over LPT%d:\n\n", LptNum );
while ( ReceiveAFile() != TOK_ENDIT ) /* Receive data until */
; /* ENDIT token */
longjmp( ReturnToEnd, LJ_OKRECD );
}
}
```