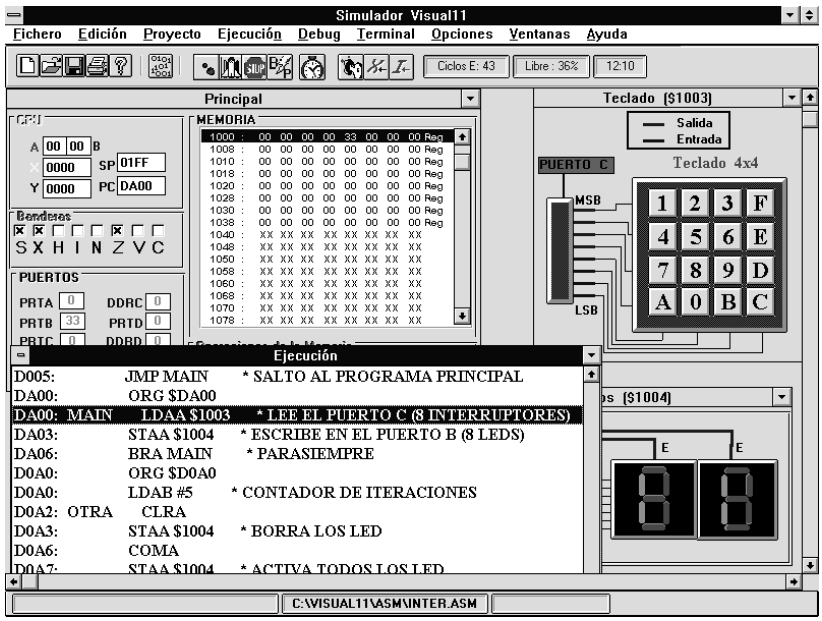
Bonifacio Martín del Brío **Carlos Bernal Ruiz**

Manual del Usuario de

VISUAL11

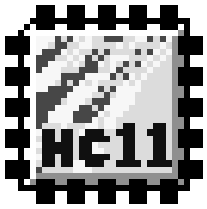
Entorno de aprendizaje de microcontroladores basado en el M68HC11 (versión 0.95)



Area de Tecnología Electrónica

Depto. de Ingeniería Electrónica y Comunicaciones

Universidad de Zaragoza



Manual del programa Visual11 (versión 0.95)

Bonifacio Martín del Brío
Carlos Bernal Ruiz
Dept. Ingeniería Electrónica y Comunicaciones
Universidad de Zaragoza. España

Diciembre 1997

nenet@posta.unizar.es
http://zape.unizar.es

Area de Tecnología Electrónica
Depto. de Ingeniería Electrónica y Comunicaciones
Universidad de Zaragoza. España

Indice.

LEE_ME.TXT

MUY IMPORTANTE.

La presente versión de Visual11 (0.95) se distribuye gratuitamente tal cual, sin garantía de ningún tipo. Su propósito es puramente educativo, y no debe ser empleado con fines lucrativos.

Asimismo, los autores no se responsabilizan de cualquier daño o pérdida causada directa o indirectamente por el uso de este producto.

Tanto el programa como la documentación adjunta puede copiarse y distribuirse libremente tal cual (sin modificaciones), a condición de que se reconozca su procedencia (Bonifacio Martín del Brío y Carlos Bernal Ruiz, de la Universidad de Zaragoza, España), y **siempre con propósitos educativos, nunca lucrativos**. Por favor, envíe una nota a la siguiente dirección si decide su empleo dentro de algún curso, sea éste universitario o no, o si desea realizar alguna sugerencia:

Bonifacio Martín del Brío.
Dept. Ingeniería Electrónica y Comunicaciones.
Universidad de Zaragoza.
Zaragoza, España.

e-mail: nenet@posta.unizar.es

CREDITOS.

Visual11 ha sido programado en Visual Basic 3.0, de Microsoft.

Visual11 hace uso del compilador de dominio público AS11, de Motorola.

El manual se ha escrito con el editor Word 6.0, de Microsoft.

Algunos iconos se han realizado con Resource Workshop 1.02, y la ayuda se ha compilado con HC31.EXE. ambos de la compañía Borland.

Crescent Software ha programado el control TRUEGRID.VBX (dominio público), que ha sido empleado en Visual11.

Si observa que se ha empleado algún recurso software no referenciado aquí, por favor, indíquenoslo para incluirlo en esta lista en próximas versiones.

AGRADECIMIENTOS.

A MOTOROLA, por su familia de microprocesadores y microcontroladores de 8 bits, completa, potente y muy educativa. Por la información que hemos tomado de sus libros de datos y página *web* (http://www.motorola.com), así como por el software que han establecido como dominio público. Consideramos a Motorola una compañía ejemplar en muchos sentidos.

A la empresa SELCO (Madrid), distribuidora de Motorola en España, y especialmente a Chelo López y José Pérez, por su constante atención, apoyo y material suministrado. SELCO es consciente de que facilitar el acceso de sus productos en el entorno universitario revertirá favorablemente en sus ventas a medio-largo plazo, cuando nuestros alumnos trabajen como ingenieros en compañías diversas. Desde nuestra experiencia particular pensamos que otras distribuidoras o delegaciones en España aún no son plenamente conscientes de que un ingeniero tiene cierta tendencia a usar en su vida profesional aquello que ha empleado durante su formación, y que facilitando el acceso de los docentes a sus productos pueden estar sembrando las ventas del futuro.

A los autores del programa Micro [Varela 94], sobretodo a Jose Manuel Gómez (Universidad de Castilla la Mancha, Toledo) y Manuel Castro (UNED, Madrid). Dicho simulador MS-DOS nos sirvió en una primera época (ya lejana) de inspiración.

A los alumnos de la asignatura *Microprocesadores e Instrumentación Electrónica* durante estos primeros años de andadura de la especialidad de Electrónica Industrial; de manera especial a Miguel Angel Moya, Antonio Bono, Juan José Lanuza, Jesús Bernal Ruiz, Sergio Puértolas, Carlos Mairal, Fidel Vicioso, y otros muchos alumnos que han realizado importantes aportaciones a la asignatura, en la forma de programas, placas, montajes, o simplemente consejos.

Carlos Bernal Ruiz merece un capítulo aparte; él es la persona que más horas y entusiasmo ha dedicado a la programación de Visual11, él es su verdadero autor.

Finalmente, agradecer las muestras de ánimo y apoyo prestado por compañeros de la Universidad de Zaragoza, especialmente Nicolás Medrano y Tomás Pollán Santamaría.

Dr. Bonifacio Martín del Brío

Profesor Titular de la E.U. de Ingeniería
Técnica Industrial de Zaragoza

1. Acerca de .... Visual 11.

Una vez hayamos resaltado la enorme importancia de los microprocesadores y microcontroladores en la actualidad, exponemos en este capítulo qué es Visual11 y hacia quién va especialmente dirigido.

1.1 Microprocesadores y Microcontroladores.

Aunque el término **microprocesador** evoca en el público general la CPU (*Central Processing Unit*) o 'corazón' de un computador (como al Pentium de un PC), el especialista en electrónica sabe que sistemas electrónicos de todo tipo incluyen en la actualidad uno o más microprocesadores [Malone 96]. Todo equipo, maquinaria o instalación que incluya una parte electrónica, con toda seguridad contendrá también microprocesadores. De hecho, buena parte de los microprocesadores vendidos en el mundo acaban incorporados en equipos o máquinas como lavadoras y hornos microondas, cámaras de vídeo y fotográficas, tarjetas monedero y mandos a distancia. Se dice que en un automóvil moderno puede haber del orden de 40 microprocesadores desempeñando calladamente tareas diversas (sistemas de frenado ABS, inyección electrónica, suspensión activa, sistemas de seguridad, computador de a bordo, etc.).

Debido a que el usuario no es consciente de que al apretar el pedal del freno o presionar un botón del mando a distancia está manejando un microprocesador, se suele hablar en estos casos de **computación oculta**. No obstante, estos microprocesadores que calladamente, y sin que el usuario apenas sea consciente de ello, desempeñan eficientemente su labor, son algo diferentes a las potentes CPU incluidas en los computadores. Por un lado, su potencia de cálculo suele ser reducida (alrededor del 80% de las unidades vendidas son de 4 u 8 bits [Bursky 95]); por otro, dentro del mismo encapsulado junto a la CPU integran más elementos, como cierta cantidad de memoria (ROM, RAM, EPROM, EEPROM, *flash*), y diversos elementos del subsistema de E/S, como puertos y dispositivos periféricos (contadores, temporizadores, conversores A/D, etc.). Este tipo especial de microprocesador, que integra en una única pastilla prácticamente todos los elementos del sistema de control, se denomina **microcontrolador** (algunas casas lo llaman microcomputador), resultando un dispositivo de bajo coste, y tamaño y consumo reducidos.

Por lo tanto, ya que la mayor parte de los sistemas electrónicos actuales incluyen uno o más microcontroladores, el estudio del diseño de sistemas basados en ellos es un aspecto relevante que debe ser tratado en carreras técnicas como Ingeniería Industrial, Ingeniería Electrónica o Electrónica Industrial.

Tras varios años de experiencia en la enseñanza de sistemas basados en microprocesadores y microcontroladores en la especialidad de Electrónica Industrial de la E.U. de Ingeniería Técnica Industrial de Zaragoza (EUITIZ), decidimos emprender la compleja tarea de desarrollar un entorno software que permitiera al alumno practicar este importante asunto. Las razones que nos movieron fueron varias. Por un lado, los simuladores entonces disponibles no nos satisfacían en demasía (normalmente simulaban tan solo la CPU, por lo que solamente servían para practicar la

programación en ensamblador); por otro, queríamos disponer de una herramienta cuyo funcionamiento al alumno no lo costase demasiado tiempo aprender y además pudiese copiar libremente. Con estas ideas en mente desarrollamos en primer lugar un simulador del microprocesador M6800 de Motorola para sistema operativo MS-DOS [Martín del Brío 96, 97c], y, con la experiencia obtenida, abordamos la enorme tarea de desarrollar un simulador para el microcontrolador M68HC11 (también de Motorola), esta vez para entorno MS-Windows. El resultado final de varios años de trabajo es **Visual11**.

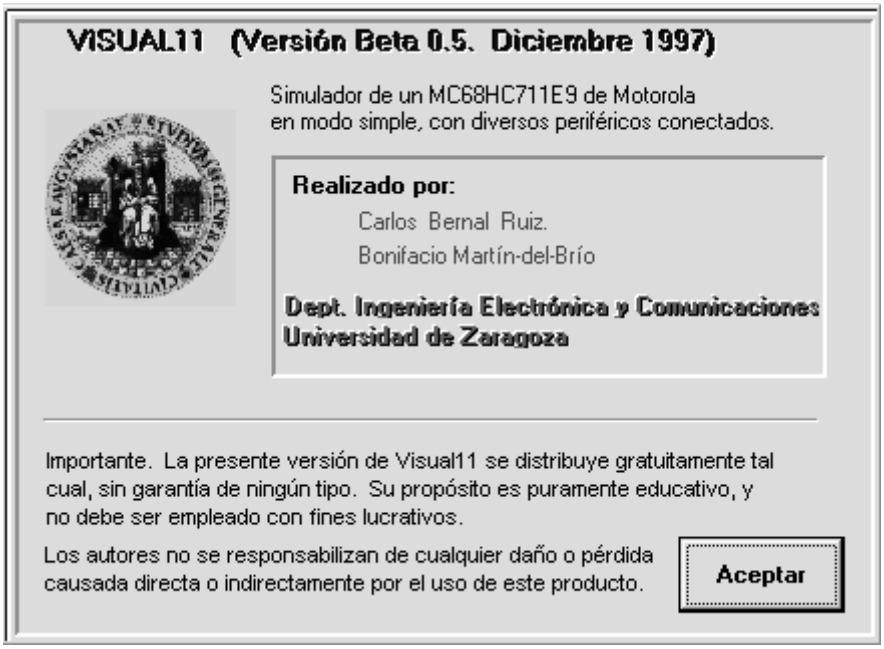


Figura 1. *Ventana ACERCA_DE de Visual11.*

1.2 ¿Qué es Visual 11?

Visual11 es una aplicación Windows desarrollada para el aprendizaje de sistemas basados en microcontroladores, construido alrededor de un M68HC11 de Motorola.

Este programa no es tan solo un simulador, sino más bien todo un entorno software para computador PC compatible con sistema operativo Microsoft Windows 3.11 (o Windows 95), que permite:

- Editar programas ensamblador.
- Compilar.
- Simular el núcleo procesador del 68HC11 (CPU-11).
- Simular algunos recursos de entrada/salida básicos
- Cargar y ejecutar el programa ya comprobado en un 68HC11 real (está pensado para trabajar con la conocida tarjeta de evaluación 68HC11 EVBU de Motorola).

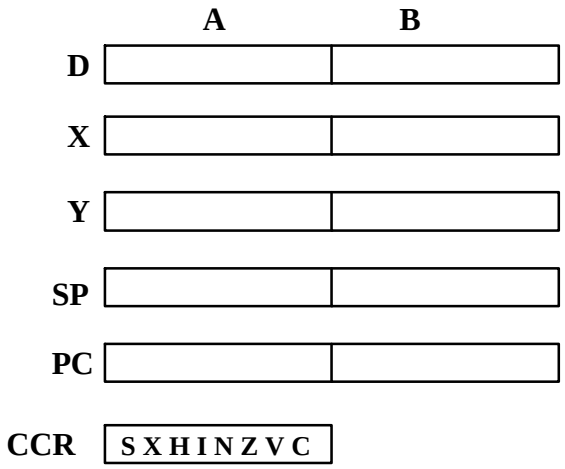


Figura 2. *Registros del núcleo procesador del 68HC11.*

Al ejecutar el icono de Visual11 desde Windows se despliega un interfaz de usuario convencional, de fácil e intuitivo manejo. El programa lo hemos **enfocado hacia la primera etapa en la enseñanza de microprocesadores**, en la que el alumno aprende la arquitectura y organización de la CPU, practica con el juego de instrucciones, construye unos primeros programas básicos, y aprende los fundamentos de las entradas/salidas. En esta primera etapa conviene que el alumno no vea su atención distraída del verdadero objetivo docente, por lo que resulta interesante que disponga de un entorno sencillo, básico, sin complejas opciones que puedan resultar de interés para un profesional, pero que él no acabará de comprender.

La presente versión de Visual11 simula un MC68HC711E9 operando en modo simple, con diversos periféricos conectados a sus puertos de propósito general B y C. Además del núcleo procesador del 68HC11 (Fig. 2), el programa simula:

- Sus bloques de memoria integrados: RAM (512 bytes), EEPROM (512) y EPROM (12 KB).
- Los puertos de propósito general B (8 bits, salida) y C (8 bits, entrada/salida).
- Los siguientes periféricos básicos que pueden ser conectados a los puertos B y C: grupo de 8 diodos LED, 8 interruptores, dos visualizadores de 7 segmentos multiplexados, y un teclado hexadecimal no codificado.
- Las interrupciones XIRQ (no enmascarable) e IRQ (enmascarable).
- La patilla y estado de RESET.

Por lo tanto, Visual11 permite no solamente practicar la programación en ensamblador, sino además estudiar la arquitectura y organización de un microcontrolador, y practicar los aspectos básicos de las entradas y salidas, incluidas las interrupciones del sistema. El alumno se ve forzado a trabajar desde el comienzo con un microcontrolador de características casi reales, con determinado mapa de memoria y con recursos concretos, de la misma manera que si trabajase con un microcontrolador real. Gracias al simulador podrá practicar con este microcontrolador 'quasi-real' en casa o en una sala de usuarios, en cualquier momento y con coste cero.

El lector puede preguntarse porqué utilizar el **M68HC11 de Motorola**. Por un lado, el 68HC11 no es tan solo un potente microcontrolador (y uno de los más conocidos), sino que

también resulta una excelente herramienta educativa que permite enseñar una enorme variedad de conceptos gracias a los numerosos periféricos que integra (recomendamos al usuario no familiarizado con el microcontrolador 68HC11 de Motorola la consulta del **Apéndice A**, donde se proporciona una breve introducción al mismo).

Por otro lado, para las etapas posteriores del aprendizaje del desarrollo de sistemas basados en microcontroladores el alumno debe trabajar con hardware real. Existen numerosas tarjetas para el 68HC11 disponibles a precios razonables (se suelen denominar tarjetas de evaluación), aunque también el alumno puede construirse de una manera sencilla (y barata) su propia tarjeta (Fig. 3), siguiendo las indicaciones dadas por la propia Motorola. Dicha tarjeta puede controlarse desde un computador PC mediante el programa de domino público PCbug11, también de Motorola, disponible en internet[1], pudiendo así grabar el software de aplicación en su EEPROM interna desde un PC convencional sin necesidad de hardware adicional.

En definitiva, debido a su potencia, precio, muy amplio uso, facilidad de realización de montajes, sencillez de la arquitectura Motorola, libros y documentación disponible, el 68HC11 es uno de los microcontroladores más empleados a nivel mundial en entornos educativos.

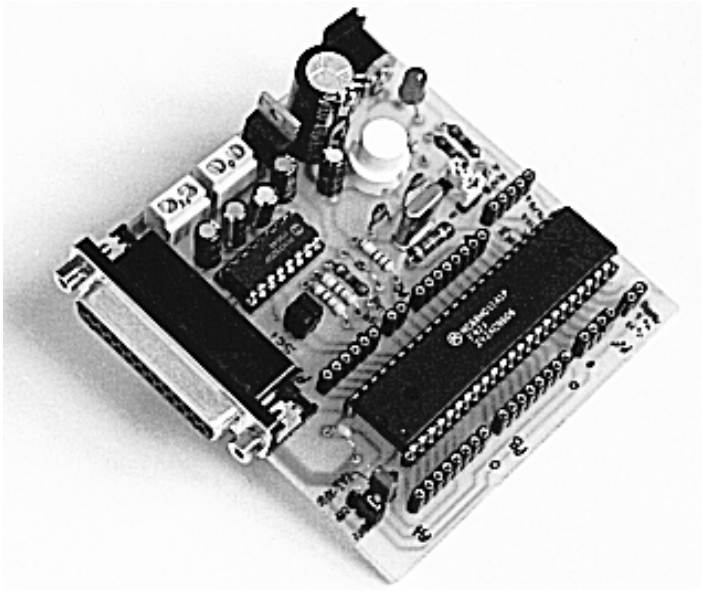


Figura 3. *Fotografía de una placa construida siguiendo las indicaciones de Motorola.*

[1] Aunque toda la información está disponible, para facilitar al alumno su acceso hemos generado en nuestro Departamento una breve publicación titulada 'Kit de iniciación al 68HC11' [Martín del Brío 97b], donde le indicamos cómo construirse una placa sencilla (por unas 4.000 pts.) y manejarla desde un computador PC.

2. Empezando a Trabajar.

Desarrollaremos en este capítulo todas aquellas cuestiones relacionadas con el proceso que debe seguir el usuario desde que tiene en su poder el disquette de instalación de Visual11 (uno tan solo), hasta que lo ha conseguido instalar en el disco duro de su PC, listo ya para ser utilizado.

2.1 Requisitos computacionales.

Para trabajar con Visual11 no se precisan requisitos computacionales demasiado grandes, especialmente teniendo en cuenta los ordenadores PC actualmente disponibles y lo exigente de las últimas versiones de programas convencionales, como MS-Word.

Visual11 requiere un ordenador PC compatible con:

- Procesador 486DX50 o superior (aconsejable Pentium).
- Tarjeta gráfica VGA (aconsejable SVGA).
- Disco duro con unos 6MB libres.
- 8MB de memoria RAM (aconsejable 16MB).
- Sistema operativo MS-Windows 3.1 o superior.

Visual11 se desarrolló, en principio, para Windows 3.11 (es una aplicación de 16 bits), aunque opera sin problemas en Windows 95 (salvo detalles que ocasionalmente se han observado, comentados en el capítulo 8).

2.2 Instalación.

Describimos a continuación el proceso que el usuario debe seguir para instalar Visual11 en su ordenador:

1. Encender el ordenador.
2. Entrar en Windows 3.11 o Windows 95; mantener el resto de las aplicaciones cerradas.
3. Introducir el único disquette de instalación en la disquettera de 3¼.
4. Leer el fichero LEEME.DOC.
5. Ejecutar desde Windows A:\SETUP.EXE, y seguir sus indicaciones.
6. Si se nos presenta algún mensaje del tipo ‘XXXX.DLL está en uso’, indicar siempre ‘ACEPTAR, ‘OK’ o ‘IGNORAR’, dependiendo de las opciones que nos presente (el

programa de instalación simplemente nos indica que la DLL que iba a instalar existe ya y que Windows la está utilizando (lo cual en realidad nos da lo mismo).

7. Si no hay ningún problema o circunstancia extraña, al final de la instalación podrá ejecutar ya Visual11, acudiendo al directorio VISUAL11 y ejecutando VISUAL11.EXE (actúe con el ratón sobre su icono), o bien desde la barra de tareas de Windows95.

NOTA. *El proceso de instalación se ha probado en numerosos ordenadores PC compatibles (Windows 3.11 y Windows95), no habiendo encontrado problema de instalación alguno. Al ejecutar Visual11 tan solo en una ocasión, en un PC concreto (con Windows95), nos hemos topado con un problema: tras instalar con aparente éxito Visual11, al ejecutarlo nos presenta el mensaje* 'Se ha producido un error de inicialización del programa'*, con lo que no hemos podido acceder al entorno Visual11. Hemos detectado que dicho error se produce justo cuando Visual11 trata de abrir la base de datos de las instrucciones del 68HC11: en este caso concreto no podía acceder a ella, probablemente debido a algún problema de configuración o incompatibilidad con el software instalado en dicho ordenador (pensamos que podría tratarse un problema relacionado con la dependencia de dicho PC de una red de área local, o quizás por incompatibilidad con Visual Basic 4.0 para 32 bits). En estos momentos no sabemos a ciencia cierta porqué Visual11 no puede ejecutarse en dicho ordenador concreto, pero pensamos que se trata de algo circunstancial, pues en el resto de los PC donde ha sido probado (para Windows 3.11 o Windows 95) no ha presentado ningún problema.*

2.3 Estructura del directorio Visual11.

Tras instalar Visual11, el usuario encontrará en el directorio VISUAL11 el ejecutable VISUAL11.EXE, el documento LEEME.DOC, y el fichero de ayuda, VISUAL11.HLP (que puede leer desde la ayuda de Windows).

De VISUAL11 surgen dos subdirectorios, VISUAL11\DATA y VISUAL11\ASM. VISUAL11\DATA simplemente contiene la base de datos que incluye las instrucciones del 68HC11.

Más interesante es VISUAL11\ASM. En este directorio está el programa ensamblador AS11.EXE de Motorola (dominio público), que el usuario puede ejecutar si lo desea desde la línea de comandos del MS-DOS; también el fichero ENS.PIF, que Windows requiere para ejecutar AS11. En este directorio el usuario encontrará además algunos ejemplos de programas ensamblador (XXX.ASM). Recomendamos editar y compilar programas ensamblador siempre dentro de este directorio VISUAL11/ASM (que Visual11 toma por defecto como directorio de trabajo), pues el program AS11 de Motorola plantea problemas ocasionales cuando se le pide la compilación de un fichero XXX.ASM que se encuentra en un directorio diferente al suyo propio.

3. Aspectos Básicos de Visual11: Menús y Ventanas.

Recordemos que Visual11 no es tan solo un simulador, sino un entorno software para el aprendizaje de microcontroladores, que permite:

- Editar programas ensamblador para el microcontrolador M68HC11 de Motorola.
- Compilar dichos programas.
- Simular el núcleo procesador del M68HC11 (CPU-11).
- Simular algunos recursos básicos de entrada/salida.
- Cargar un programa compilado en una tarjeta basada en un 68HC11 ejecutando el programa monitor BUFFALO, como pueda ser la conocida tarjeta de evaluación 68HC11EVBU de Motorola, y trabajar desde Visual11 con dicha tarjeta.

En este capítulo realizaremos una introducción a las cuestiones básicas de Visual11. En particular, describiremos las distintas opciones disponibles en sus menús, y la configuración de las distintas ventanas de trabajo. Se recomienda al usuario la lectura del **Apéndice A**, donde se realiza una introducción al 68HC11.

3.1 Configuración general del entorno.

Desde el entorno Microsoft Windows (3.11 o 95), puede ejecutarse Visual11 pinchando con el ratón sobre su icono; puede observarse cómo en primer lugar el programa inicializa tanto sus variables internas como las específicas del microcontrolador simulado.

La apariencia general del entorno en una sesión típica de trabajo puede observarse en la Figura 4. En ella pueden apreciarse, por ejemplo, los diversos **submenús** que componen el menú principal (FICHERO, EDICIÓN, PROYECTO, EJECUCIÓN, DEBUG, TERMINAL, OPCIONES, VENTANAS y AYUDA). Para facilitar el acceso a las opciones más importantes disponibles dentro de cada submenú, éstas tienen asignadas combinaciones de teclas y/o botones, como los que pueden verse en la **barra de herramientas**, bajo el menú principal.

En la misma Figura aparecen distintas ventanas, como la **ventana PRINCIPAL**, que visualiza el estado de los registros y memoria del microcontrolador, permitiendo además su fácil modificación: basta situar el ratón sobre el registro deseado y pulsar su botón izquierdo. En el caso de los **registros de la CPU**, al teclear los **dígitos hexadecimales** del nuevo valor se van superponiendo sobre los del anterior, por lo que no es necesario borrar previamente el valor viejo.

En el caso de la memoria, al pulsar el botón izquierdo surge una nueva ventana, que se explicará más adelante.

Visual11 trabaja con **proyectos**, denominación con la que designamos un programa ensamblador (NOMBRE.ASM). Un proyecto o programa puede editarse directamente en la **ventana EDICION** (que en la Figura 4 aparece replegada en la parte inferior izquierda, la cual puede desplegarse actuando con el ratón); al ser compilado (NOMBRE.S19) el código máquina resultante es cargado en la memoria (simulada) del microcontrolador para su ejecución, apareciendo entonces una nueva ventana denominada **ventana EJECUCION** (ventana situada en la derecha de la Fig. 4), desde la que puede seguirse la ejecución del programa.

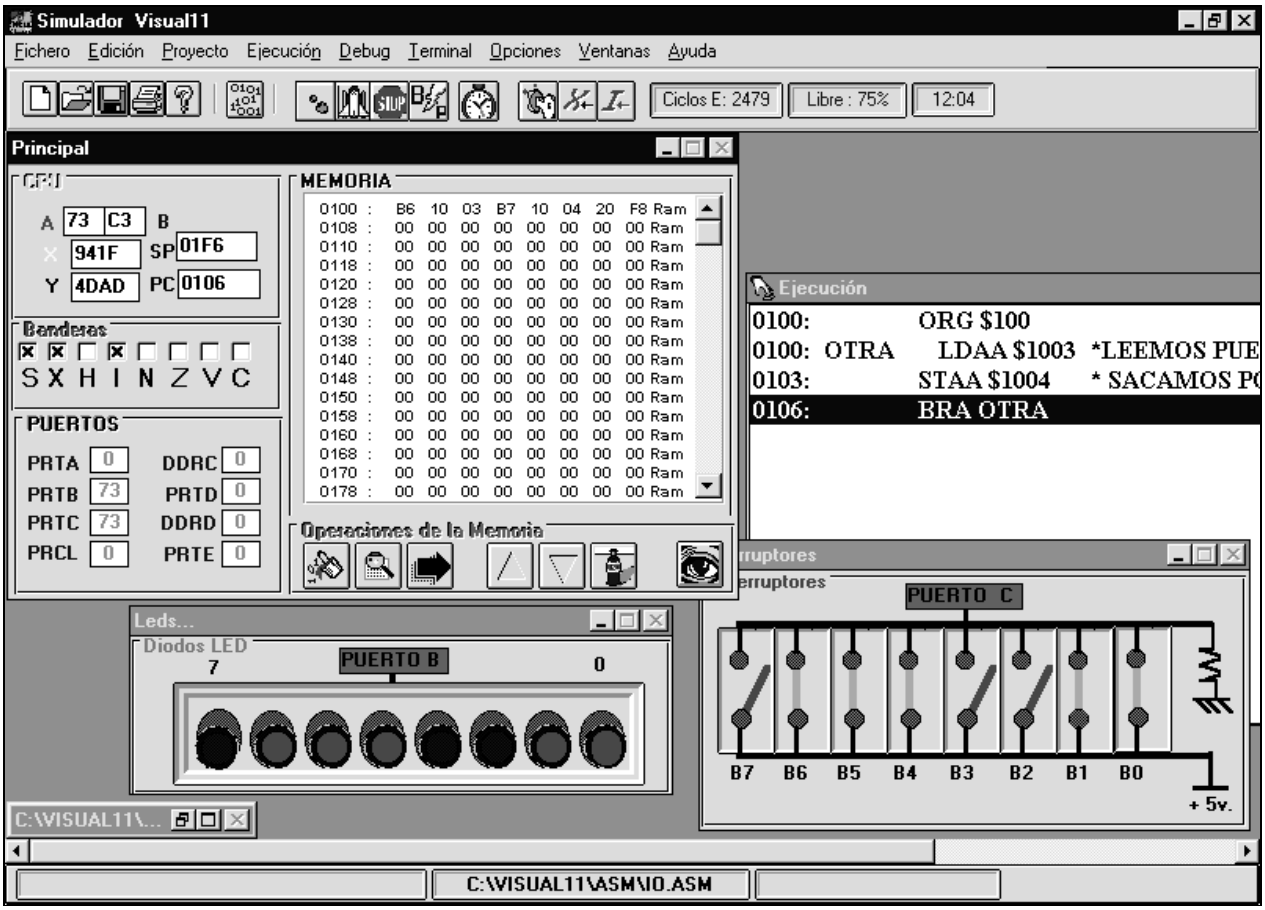


Figura 4. *Entorno de trabajo de Visual11.*

En la misma figura aparecen también desplegadas dos ventanas que presentan el estado de algunos de los periféricos simulados (LEDs e interruptores, en este caso). Finalizando este primer recorrido, puede observarse una barra inferior en la que el sistema muestra al usuario diversos mensajes, la fecha actual, o el nombre del fichero ensamblador actualmente abierto.

3.2 Menú principal.

Desde el menú principal se tiene acceso a distintos submenús, que incorporan diversas opciones, que pasamos a describir. Recordemos que las opciones más habituales tienen asignadas una combinación de teclas o incluso un botón en la barra de herramientas.

Fichero

Clásico menú Windows, para manejo de ficheros ejecutables (S19), impresión y abandonar el programa. Las siguientes son sus opciones:

Cargar fichero S19 a memoria (Ctrl+O)

Carga un fichero ejecutable (de extensión S19) en la memoria simulada del microcontrolador. Esta opción apenas será utilizada en el entorno educativo, puesto que simplemente permite la simulación de código máquina, sin tener presente el código fuente.

Imprimir

Imprime el programa fuente, el estado de los registros, los visualizadores o una zona de la memoria.

Especificar impresora

Permite seleccionar la impresora actualmente conectada al ordenador.

Salir (CTRL+S)

Abandona la sesión actual de trabajo con Visual11. Cierra el programa y retorna al sistema operativo. Antes de ejecutar esta acción, nos pregunta si realmente estamos seguros de que queremos abandonar la sesión.

Edición

Típico menú Windows con herramientas de edición apoyadas en el portapapeles, pensadas para ayudar en la edición de programas ensamblador. Estas acciones se aplican a la selección realizada con el ratón sobre la ventana EDICION, la cual se despliega al abrir o editar un PROYECTO.

Copiar (Ctrl+C)

Copia al portapapeles el texto seleccionado con el ratón en la ventana EDICION.

Cortar (Ctrl+X)

Permite cortar el texto seleccionado en la ventana EDICION, llevándolo al portapapeles.

Pegar (Ctrl+V)

Pega el texto guardado en el portapapeles en el punto de la ventana EDICION donde se encuentre el cursor.

Borrar

Borra el texto seleccionado en la ventana EDICION.

Insertar objeto

Inserta un fichero ensamblador (NOMBRE.ASM) en el lugar de la ventana EDICION donde se encuentre actualmente el cursor.

Proyecto

Es este uno de los menús fundamentales de Visual11. En esta aplicación denominamos PROYECTO a un fichero ensamblador (fuente), de extensión ASM. **En una sesión de trabajo el primer paso suele ser siempre abrir un fichero ensamblador ya existente, o bien crear uno nuevo.**

Crea uno nuevo

Crea un fichero fuente de extensión ASM, en el que escribiremos el programa ensamblador que se trata de simular.

Abre uno existente (F3)

Abre un fichero ASM ya disponible, para su modificación o compilación.

Guarda los cambios (F2)

Guarda los cambios realizados sobre el fichero ASM actualmente abierto.

Guardar como

Guarda el fichero ASM actual con un nuevo nombre.

Cierra proyecto

Cierra el fichero ASM actual.

Compila (F4)

Compila el fichero ASM actualmente abierto, generando el fichero de código objeto (código máquina) de extensión S19, resultado de la compilación (NOMBRE.ASM-> NOMBRE.S19). Además, y de forma automática, carga en la memoria del microcontrolador simulado el código ejecutable resultado de la compilación. No hemos desarrollado ningún compilador específico para la aplicación, simplemente hacemos uso del famoso compilador AS11 de Motorola (dominio público), cuyo código S19 generado (en los denominados 'S-records' de Motorola) nos limitamos a cargar en Visual11.

Ejecución

Menú que implementa diversas posibilidades de simulación de un programa código máquina (ensamblador previamente compilado).

Paso a paso (F8)

Ejecuta un programa paso a paso, es decir, instrucción a instrucción. Equivale al clásico *Step* o *Trace* de los simuladores comerciales.

Lanza simulación (F5)

Lanza la ejecución de todo el programa. Es el clásico *run* de los simuladores comerciales.

Para simulación (F6)

Detiene la simulación.

Reinicialización del programa

Reinicia el entorno de trabajo Visual11, borrando toda la memoria y los registros. Sirve para iniciar una nueva sesión de simulación sin salir del entorno Visual11.

Reinicialización del microcontrolador [reset]

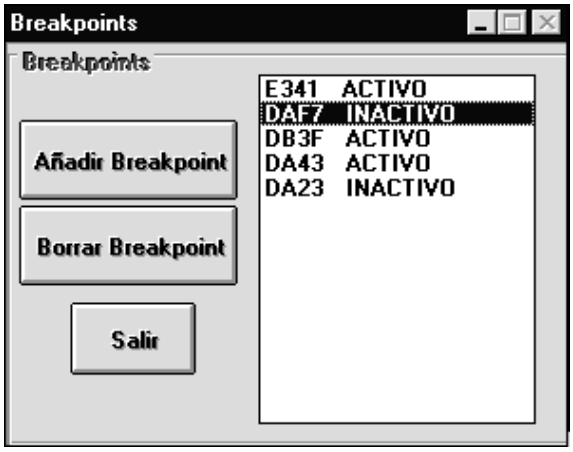
Simula que ha llegado una señal activa por la patilla RESET del 68HC11, de modo que el simulador inicializa los registros del HC11, y carga a continuación en el registro contador de programa (PC) la dirección alojada en $FFFE y $FFFF [2] (direcciones de la tabla de vectores de interrupción asignadas a RESET). Obsérvese que tras un RESET la memoria del 68HC11 mantiene sus contenidos actuales.

Debug

En este menú se incluyen algunas utilidades para el depurado de programas.

Puntos de ruptura

Este submenú gestiona una lista con distintas direcciones donde se desea mantener puntos de ruptura. Estos pueden dejarse activos o desactivarse en un momento dado presionando los botones 'Añadir breakpoint' y 'Borrar breakpoint', o bien actuando directamente sobre los componentes de la lista con el ratón (botón izquierdo).



Reset del contador de ciclos

Pone a cero el contador de ciclos de reloj E del HC11. El contador permite medir el tiempo de ejecución de un programa (tiempo máximo: 9.999 ciclos).

[2] En la notación de Motorola los números hexadecimales (base 16) se indican mediante el símbolo $ (por ejemplo, $3F6A) y los binarios con % (p.e., %10101100). Si no se incluye símbolo alguno el número se supondrá base 10.

Terminal

Este menú convierte Visual11 en una terminal de la placa de evaluación 68HC11EVBU de Motorola, o de cualquier otra que posea en ROM el programa monitor BUFFALO. Debido a su carácter más avanzado, será explicado con detalle en el **Apartado 6**. Incluye tres opciones:

Parámetros de la comunicación

Entrar en la terminal (Ctrl+T)

Programar HC11 con proyecto (Ctrl+F9)

Opciones

El menú OPCIONES incluye distintas posibilidades de la apariencia y configuración del entorno Visual11.

Barra de herramientas flotante

Convierte la barra de herramientas convencional en una barra flotante que queda colocada siempre por encima de todas las ventanas abiertas.

Nivel de advertencia: Nivel 1, nivel 2, nivel 3.

Se han definido diversos niveles de advertencia y error, que van desde el **Nivel 3**, aconsejable para el principiante (en el que se advierte de muchas maniobras sospechosas de error, como el intento de escritura sobre EPROM, o el acceso a casillas de memoria no implementadas), hasta el **Nivel 1**, ideado para el usuario más experto, en el que las advertencias se reducen a las más graves.

Bloques hardware: interruptores, LEDs, teclado, 7 segmentos.

Visual11 simula la operación de los puertos B (dirección $1004, de salida) y C ($1003, programable de entrada/salida), de propósito general. Se simulan también cuatro periféricos sencillos (Fig. 12), que pueden conectarse a estos puertos actuando sobre las opciones de este submenú. Así al puerto B pueden conectarse un conjunto de 8 LEDs o dos visualizadores de 7 segmentos multiplexados; al puerto C pueden conectarse un conjunto de 8 interruptores o un teclado matricial hexadecimal (16 teclas).

En el apartado 3.6 se explicarán más detalles de esta opción, sumamente interesante desde un punto de vista pedagógico.

Barra inferior

Habilita o hace desaparecer la barra inferior de la aplicación, dejando más espacio libre en la pantalla. En el recuadro izquierdo de la barra inferior aparecen mensajes relacionados con las operaciones que están siendo ejecutadas por la aplicación (por ejemplo, 'Reseteando microcontrolador'). En el recuadro central puede aparecer la fecha o el *path* del fichero actualmente abierto.

Ventanas

Menú habitual en las aplicaciones Windows que permite establecer la configuración de las ventanas actualmente abiertas.

Cascada

Coloca en cascada las ventanas actualmente abiertas.

Mosaico

Coloca en mosaico las ventanas actualmente abiertas.

Finalmente, en la parte inferior de este menú aparece la lista de las ventanas que permanecen abiertas, que pueden ser seleccionadas directamente con el ratón (lo que resulta útil, por ejemplo, cuando la ventana que nos interesa en un momento dado se encuentra oculta por las demás).

Ayuda

Menú habitual también en las aplicaciones Windows, con algunas opciones de ayuda para el usuario.

Ayuda del programa

Submenú que ejecuta la ayuda del programa Visual11.

Acerca de ...

Submenú que visualiza la ventana 'Acerca de ...', en la que figura información sobre la versión actual de Visual11 (ver Fig. 1).

Expuestas brevemente las opciones disponibles en cada uno de los menús de Visual11, pasaremos a continuación a explicar más profundamente las más importantes, y su manejo en relación a las ventanas fundamentales del entorno.

3.3 Barra de herramientas.

Para facilitar el acceso a las opciones de los menús de mayor uso se ha incorporado la barra de herramientas de la Figura 5. Exponemos la utilidad de cada uno de sus botones, empezando por el botón situado más a la izquierda:

- Crear nuevo proyecto.
- Abrir proyecto ya existente.
- Guardar proyecto.
- Imprimir.
- Ayuda del programa.
- Compilar proyecto.
- Ejecutar paso a paso.
- Lanzar simulación.
- Detener simulación.
- Puntos de ruptura (*break point*).
- Reset del reloj.
- Reset del HC11.
- Interrupción XIRQ.
- Interrupción IRQ.

Muchos de estos iconos representan opciones ya descritas en el apartado anterior. Señalar solamente la utilidad de las tres herramientas situadas a la derecha de la barra, que permiten solicitar **interrupciones XIRQ (no enmascarable), IRQ (enmascarable) y RESET**, simulando cada uno de los respectivos procesos.

Asimismo, a la derecha de la barra de herramientas se representa el estado actual del contador de ciclos, el porcentaje de recursos del sistema libres y la hora actual.

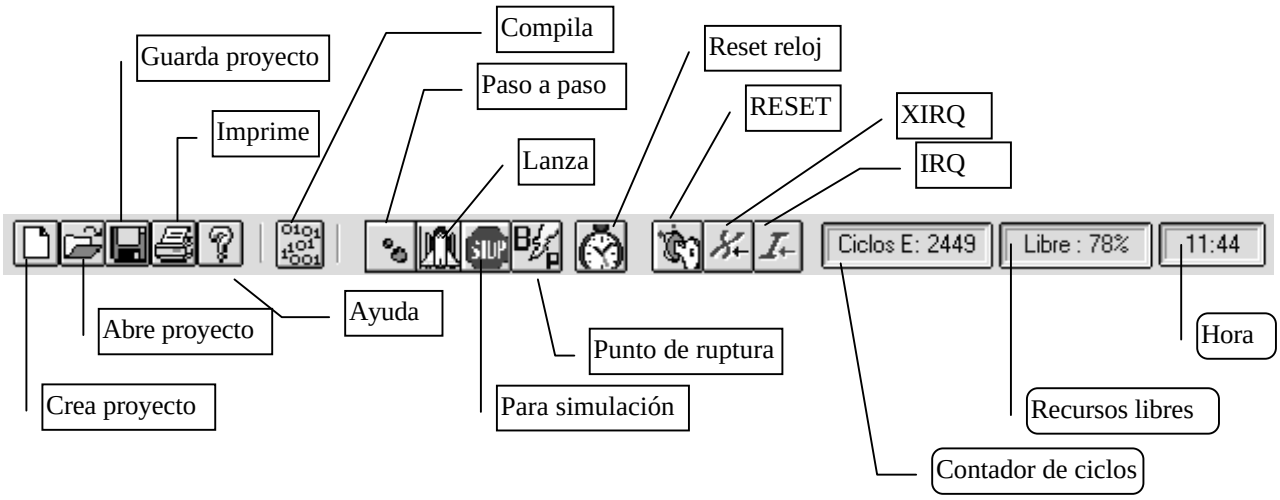


Figura 5. *Barra de herramientas de Visual11.*

Aunque no se incluya su icono en la barra de herramientas, señalar que se dispone en el entorno de una **calculadora**, que puede activarse pinchando con el ratón sobre su icono, el cual se encuentra situado en la parte inferior izquierda del entorno de trabajo. Su principal utilidad es realizar las operaciones básicas en decimal o hexadecimal, y realizar el cambio de números entre ambas bases.

3.4 Ventana PRINCIPAL.

En la **ventana PRINCIPAL** (Fig. 6) se representa el estado actual de los registros de la CPU, los señalizadores o *flag*, los registros relacionados con los puertos, y una zona de la memoria del sistema. Puede **modificarse el valor de los registros** accediendo mediante el ratón y escribiendo el nuevo valor (que sobre-escribe el valor anterior), **y el de los *flag*** simplemente pinchando con el ratón.

Buena parte de la ventana PRINCIPAL se dedica a visualizar un zona de memoria del sistema (parte derecha de la Fig. 6). Dado que simulamos un MC68HC711E9, se contempla la existencia de una zona de memoria RAM (de $0000 a $01FF), zona EEPROM ($B600 a $B7FF), EPROM ($D000 a $FFFF), más los registros de configuración ($1000 a $103F); para que el alumno en todo momento sepa con qué tipo de memoria está trabajando tras cada línea de 16 bytes de memoria añadimos una etiqueta que lo indica (‘RAM’, ‘REG’, ‘EE’ y ‘EP’).

Para modificar el contenido de una casilla de memoria se debe situar el ratón sobre ella en la ventana PRINCIPAL y pinchar dos veces, surgiendo entonces una nueva ventana (Fig. 7) desde la que pueden modificarse el estado de 16 casillas consecutivas de memoria, bien sobre-

escribiendo el viejo valor, bien borrando y escribiendo el nuevo. Pueden introducirse datos tanto en base 10, como 16 o incluso caracteres ASCII.

Si se pulsa el botón que tiene por icono un ojo se despliega la parte derecha de la ventana PRINCIPAL (replegada, por defecto, Fig. 6), que incorpora los denominados **visualizadores**. Los visualizadores de la parte izquierda representan el **estado actual de la pila** del sistema (que establece el registro SP), los de la derecha permiten **visualizar el estado de direcciones concretas de memoria**, de interés especial para el programa que se está simulando. Esta parte de la ventana puede replegarse accionando el botón de la flecha, situado bajo la pila.

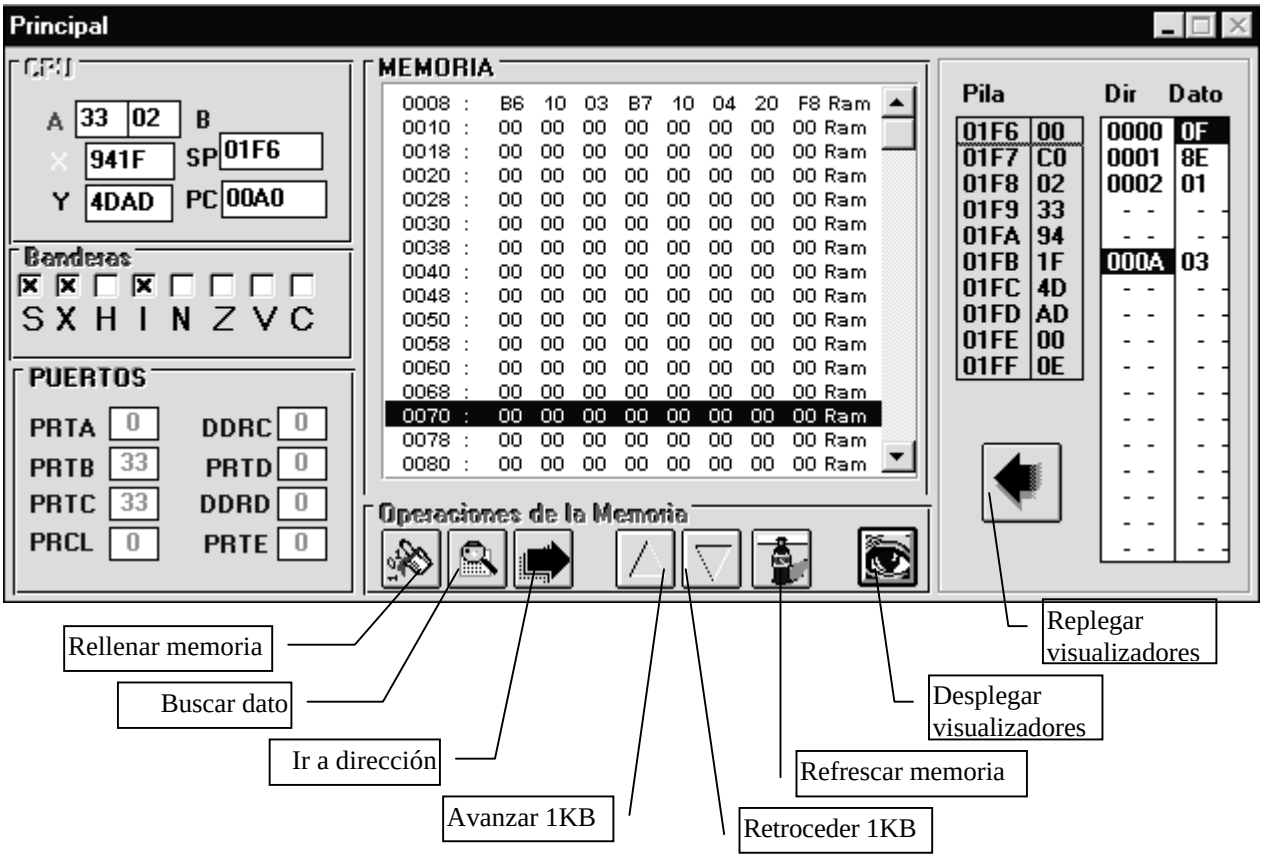


Figura 6. *Ventana PRINCIPAL. De izquierda a derecha: registros, memoria y visualizadores.*

Figura 7. *Ventana que permite la modificación de las casillas de memoria.*

En la ventana PRINCIPAL, bajo el área de memoria visualizada, hemos situado una serie de botones que permiten (Fig. 6, de izquierda a derecha):

- Rellenar un bloque de direcciones de memoria con un mismo dato.
- Buscar un dato en el mapa de memoria.
- Saltar a determinada casilla de memoria.
- Retroceder 1024 casillas de memoria ($400).
- Avanzar 1024 casillas de memoria ($400).
- Refrescar la ventana de memoria.
- Desplegar los visualizadores.

Hay que destacar el papel del **botón de refresco** (refresco del estado de la ventana PRINCIPAL); se recomienda su uso cuando se aprecien problemas de visualización de los datos contenidos en memoria.

3.5 Edición, compilación y ejecución de programas ensamblador.

En este apartado expondremos el proceso que debe seguirse para editar, compilar y cargar en memoria un programa escrito en lenguaje ensamblador dentro del entorno Visual11.

Características del ensamblador AS11.

Visual11 no incorpora un compilador de ensamblador propio, sino que hace uso del conocido programa **AS11.EXE, ensamblador de dominio público de Motorola**. En el **Apéndice A** se adjunta la transcripción completa del manual de AS11 que Motorola incluye en la forma de fichero de texto junto con el software.

No obstante, dado que se trata de un aspecto fundamental en el trabajo con Visual11, describiremos a continuación las características más destacables de AS11, en particular **la sintaxis de los programas ensamblador** que requiere.

AS11 es un ensamblador en dos pasos, que admite el empleo de **etiquetas** para representar direcciones o datos; una etiqueta puede constar de hasta seis caracteres alfanuméricos, y debe comenzar por una letra.

Las **constantes numéricas** por defecto son de tipo decimal, aunque pueden definirse de otro tipo utilizando los siguientes prefijos:

- **$** para hexadecimal.
- **%** para números binarios.

- **@** para octal.
- ' para carácter ASCII.

Así, puede tenerse, 109, $6D, %01101101, @155 ó ‘m.

Un símbolo importante a tener en cuenta es el asterisco, *****, que colocado al principio de una línea (pegado al margen izquierdo) indica que lo que continua debe ser tratado como un **comentario de programa**. AS11 también considera comentarios de programa todo aquello que se escriba a la derecha de una instrucción ensamblador.

AS11 contempla, entre otras, las siguientes **directivas o pseudoinstrucciones**:

- **ORG** (*Origin*) Establece el lugar de memoria donde se alojará el código o datos.
  (Ejemplo: ORG $100 indica que se almacena a partir de la dirección $100).
- **EQU** (*Equal*) Asigna un valor a una etiqueta.
- **FCB** (*Form Constant Byte*). Guarda un byte en una posición de memoria.
  (Ejemplo: FCB 3, 7, $3A, guarda estos bytes en casillas consecutivas).
- **FDB** (*Form Double Byte Constant*) Guarda dos bytes en memoria.
- **FCC** (*Form Constant Character String*) Guarda en memoria los valores ASCII de los caracteres que componen una cadena (*string*).
  (Ejemplo: FCC “HOLA”).
- **RMB *num*** (*Reserve Memory Byte*) Reserva la cantidad *num* de bytes de memoria.
  (Ejemplo: RMB 10 reserva 10 bytes de memoria).
- **END** Fin de fichero.

En la Figura 8 se muestra la estructura típica de un fichero ensamblador para AS11. Dicho fichero incluye instrucciones del juego del ensamblador del 68HC11, como LDAA o INC, pseudointrucciones o directivas, como ORG, FCB y END, y etiquetas como DATO1 y OTRA. Obsérvese que una línea de programa se organiza siempre en tres campos: etiqueta (opcional), instrucción y operando:

	[*Etiqueta*]	*Instrucción*	*Operando*
Por ejemplo:	OTRA	INC	DATO1

```
* Programa de prueba
        ORG $0020
DATO1   FCB $A0

* Aqui empieza el programa
        ORG $0100
        LDAA DATO1
OTRA    INC  DATO1
        BRA  OTRA

        END
```

Figura 8. *Ejemplo de programa ensamblador compilable con AS11 de Motorola.*

Como resultado de la compilación de un programa fuente NOMBRE.ASM se genera un fichero NOMBRE.S19 con el código objeto, escrito en el formato S-Record propio de Motorola.

$$\text{NOMBRE.ASM (fuente)} \xrightarrow{\text{AS11.EXE}} \text{NOMBRE.S19 (objeto)}$$

Si AS11 se emplea directamente desde el sistema operativo MS-DOS (fuera del entorno Visual11), puede ordenarse la compilación de un programa con la opción '-l' (DIR\> AS11 NOMBRE.ASM -l > NOMBRE.LST), en cuyo caso se genera, además del fichero S19 que contiene el código objeto en formato 'S-record' de Motorola, un listado de programa, donde puede verse el código fuente junto al código máquina generado. Visual11 precisamente trabaja a partir de este listado, que lee e interpreta, colocando el código máquina en el lugar apropiado de la memoria simulada del HC11, y las direcciones junto con las instrucciones fuente en la ventana EJECUCION.

Ventanas EDICION y EJECUCION.

El proceso que debe seguirse en Visual11 para **editar, compilar y cargar en memoria** un programa ensamblador es el siguiente. Al solicitar la edición de un programa, es decir, lo que denominamos en Visual11 **proyecto** (PROYECTO-> CREA_UNO_NUEVO), se despliega una **ventana EDICION** (Fig. 9) en la que podemos teclear un programa ensamblador, siguiendo la sintaxis del ensamblador AS11 de Motorola expuesta en el apartado anterior.

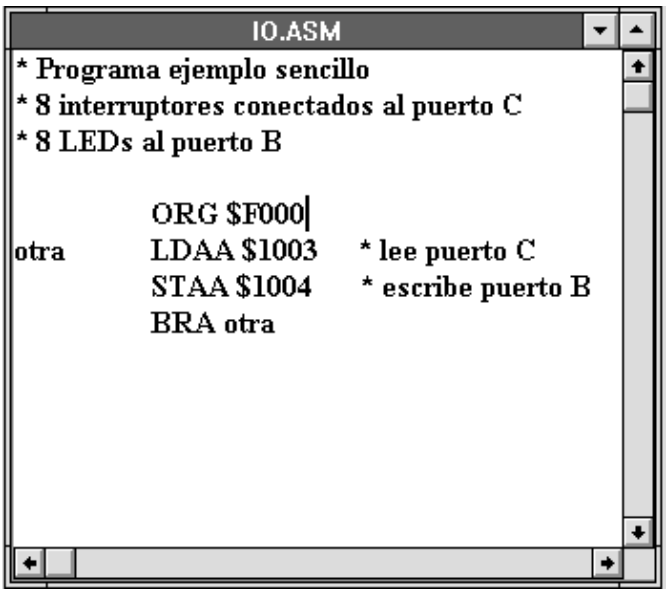


Figura 9. *Ventana EDICION.*

Si al ordenar la compilación de un proyecto (PROYECTO-> COMPILA) **AS11 encuentra algún error** en el fichero ensamblador, aparecerá una nueva ventana con el correspondiente **mensaje de error** (Fig. 10); situando el ratón sobre el mensaje de error y **pulsando su botón izquierdo** automáticamente aparecerá de nuevo la ventana EDICION con el cursor situado en la línea donde el compilador ha encontrado el error.

Si no hay errores de compilación, AS11 generará un fichero de extensión S19 conteniendo el código máquina resultante, Visual11 **automáticamente carga dicho código máquina** en las direcciones de memoria asignadas y despliega la **ventana EJECUCION** (Fig. 11), donde aparece el programa fuente que acaba de ser compilado (incluyendo la dirección de cada instrucción); ésta será la ventana desde la que se debe seguir la ejecución (simulación) de un programa.

Para comenzar la simulación solo resta **asignar al PC (contador de programa) la dirección de la primera instrucción del programa**, lo que se puede hacer tecleando dicho valor en la ventana PRINCIPAL, o bien colocando el ratón sobre la primera línea ejecutable del programa en la ventana EJECUCION y **pulsando dos veces el botón izquierdo**.

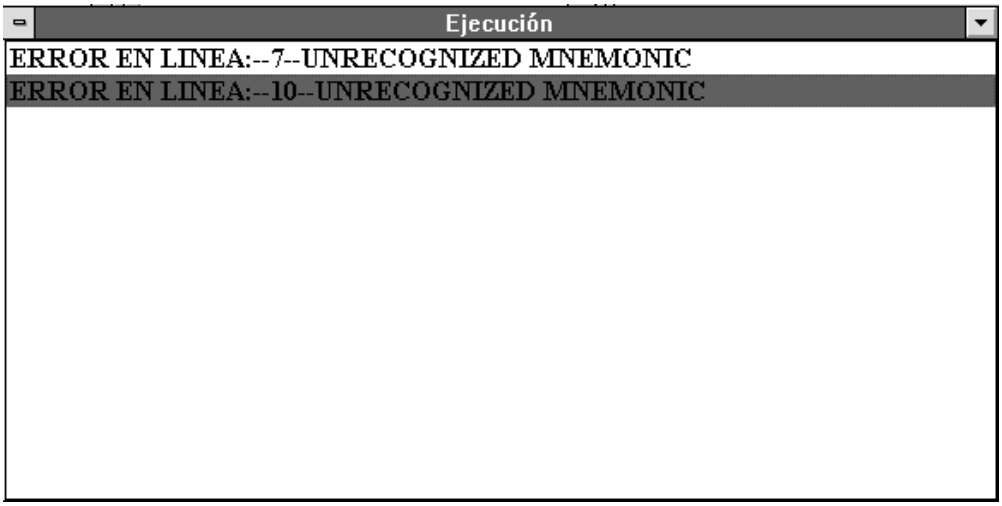


Figura 10. *Ventana EJECUCION con mensajes de error de compilación. Pulsando el botón izquierdo del ratón sobre uno de ellos apareceremos en la línea causante del error de la ventana EDICION.*

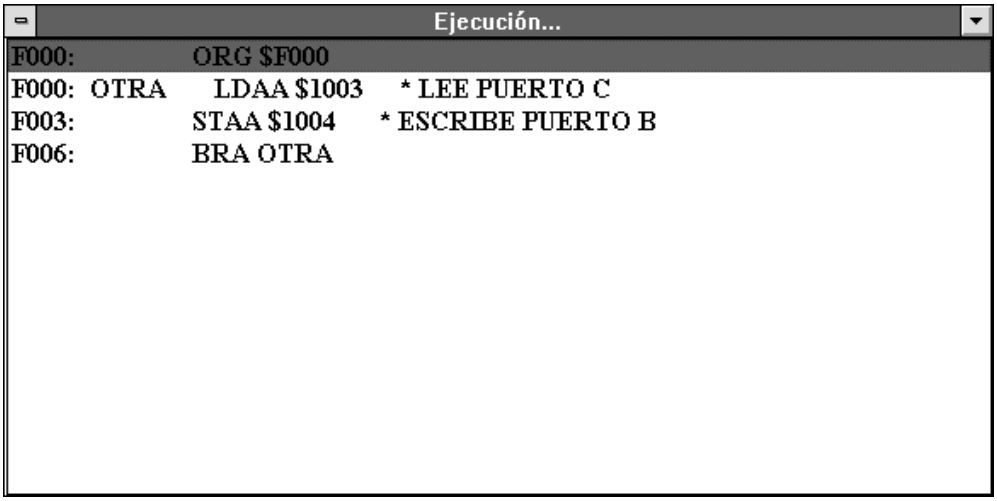


Figura 11. *Ventana EJECUCION con el código fuente del programa que ha sido compilado y cargado en memoria (más direcciones de memoria). Visual11 está ya dispuesto para simular.*

Si en algún momento se desea introducir algún cambio en el programa que se está simulando hay que volver a desplegar la ventana EDICION; ello puede ordenarse actuando sobre su icono, o bien simplemente situando el ratón sobre la línea de EJECUCION que se desee modificar: **presionando el botón derecho del ratón se despliega la ventana EDICION**, quedando el cursor situado en el lugar adecuado, donde podremos modificar y compilar.

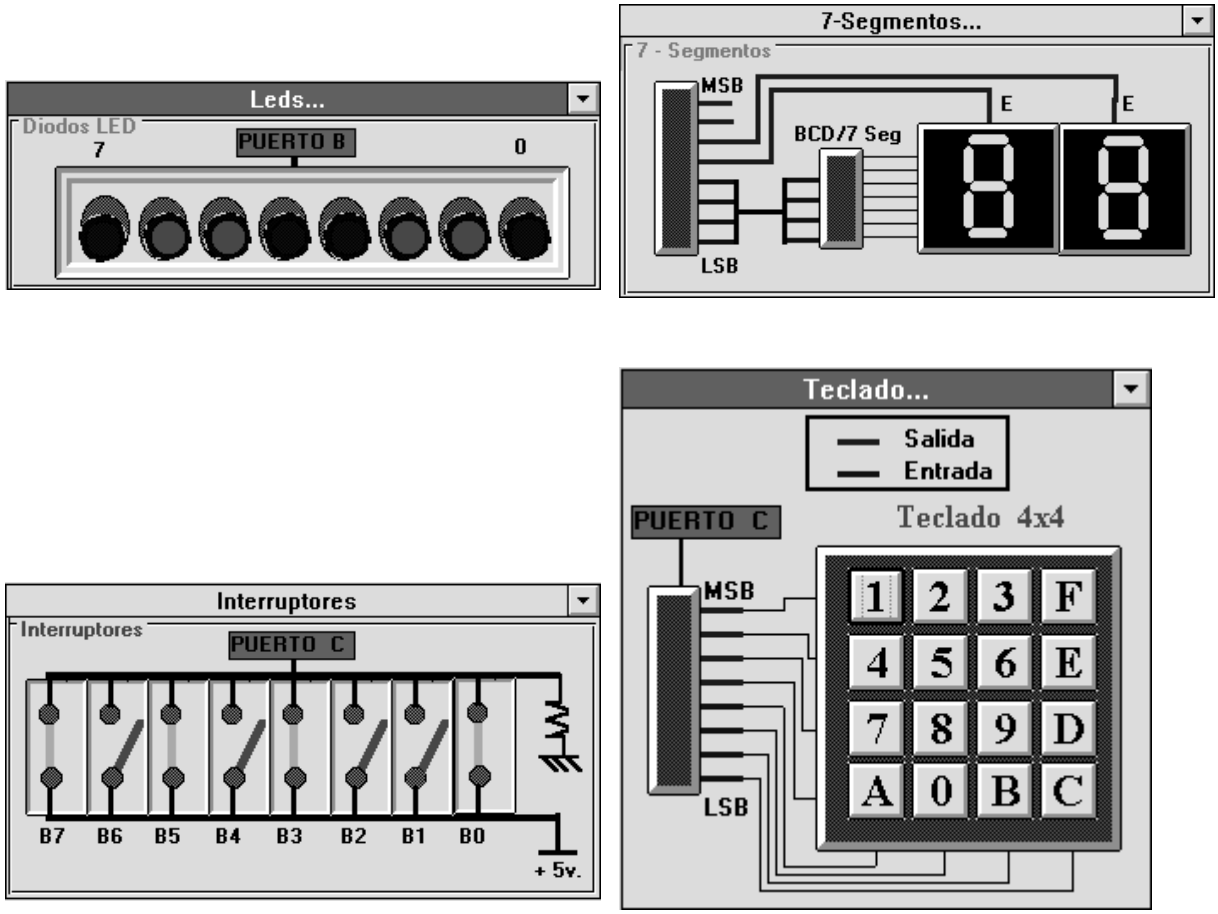


Figura 12. *Periféricos simulados en Visual11.*

3.6 Periféricos simulados.

Como ya se ha expuesto, desde el menú OPCIONES-> BLOQUES_HARDWARE puede seleccionarse entre cuatro periféricos diferentes simulados (Fig. 12), que se conectan a los puertos B y C.

Por medio del **puerto B de salida ($1004)** puede actuarse sobre un grupo de **8 diodos LED**; cada línea del puerto B tiene conectado un diodo de manera tal que se debe enviar un '1' para que el diodo correspondiente se ilumine. El alumno puede así practicar con la realización de juegos de luces por programa; si opera en un ordenador suficientemente rápido (basta un Pentium a 100 MHz) incluso deberá introducir rutinas de retraso para poder visualizar bien los diversos estados por los que pasan los diodos.

Alternativamente, pueden seleccionarse dos **visualizadores de siete segmentos multiplexados**, es decir, ambos están excitados por un único conversor BCD a siete segmentos, de modo que se debe seleccionar aquel de los dos que queremos que actúe; para ello debe ponerse un '1' en la línea del puerto B adecuada (línea 4 para el visualizador de la cifra de las decenas, línea 5 para las unidades), y enviar el valor BCD adecuado por las líneas 0 a 3. Este bloque de

visualizadores de siete segmentos se ha programado de manera que represente tanto los dígitos BCD '0' a '9', como también los hexadecimales 'A' a 'F'.

Por medio del **puerto C ($1003) programable de entrada/salida** (de entrada, por defecto) puede leerse el estado de un grupo de **8 interruptores LED**; cada línea del puerto C tiene conectado un interruptor, que envía un '0' cuando se encuentra abierto, y '1' cuando se cierra. Se actúa sobre cada interruptor individual pinchando dos veces sobre él con el ratón.

Alternativamente puede conectarse al puerto C un **teclado matricial de 16 teclas** (hexadecimal) no decodificado, que debe gestionarse mediante la técnica de exploración de teclado (*scan*). En este caso se supone que las cuatro líneas más significativas del puerto C (4 a 7) están programadas como salidas (en un 68HC11 real habría que grabar '1's en los bits 4 a 7 del registro de la dirección de los datos **DDRC**, $1007), y las cuatro menos significativas como entradas. Pulsando con el ratón una vez sobre una de las teclas del teclado, se simula que ésta se mantiene presionada (esta situación se indica pintando la tecla en color amarillo), pulsar de nuevo con el ratón equivale a liberar dicha tecla.

Los cuatro periféricos programados permiten al alumno practicar en casa con la simple ayuda de un PC los aspectos más básicos de la gestión de entradas/salidas por programa (se proporcionarán algunos ejemplos a lo largo del presente manual). Para aspectos más avanzados el alumno deberá realizar prácticas de montaje específicas en el laboratorio utilizando tarjetas como la MC68HC11EVBU de Motorola, que el alumno podrá controlar desde un PC mediante la opción de Visual11 TERMINAL, que será expuesta más adelante.

3.7 Mensajes de error.

En Visual11 se contemplan **mensajes de error y de advertencia o aviso**. La cantidad de mensajes de advertencia que son enviados al usuario en una sesión se controla mediante el menú OPCIONES-> NIVEL_DE_ADVERTENCIA, pudiéndose seleccionarse entre los **niveles de advertencia 1, 2 y 3**. El nivel 3 está orientado hacia el usuario "experto", y el nivel 3 para el novel: en el primer caso solamente se muestran los mensajes de error (los más graves), mientras que en nivel 3 se envía todo tipo de mensaje de advertencia que pueda resultar útil para un alumno que está comenzando. En las primeras sesiones con Visual11 se recomienda trabajar en nivel 3, pero lo habitual es que en cuanto el usuario se familiarice con el entorno trabaje ya siempre en el nivel 1.

A modo de ejemplo, en la Fig. 13 se muestra una ventana conteniendo mensajes típicos de advertencia que se generaría en nivel 3. En este ejemplo el usuario ha compilado un programa que contiene una parte de su código alojado en memoria EEPROM, que escribe sobre uno de los registros de configuración del 68HC11 (por ejemplo, sobre el correspondiente al puerto B de salida, $1004), y que parte de su código se ha situado sobre una zona de memoria inexistente en la versión 68HC711E9 que Visual11 simula. Los dos primeros hechos (escritura en EEPROM y en un registro de configuración) no son un proceso erróneo en sí, Visual11 en nivel 3 simplemente avisa al usuario de la circunstancia. El tercer mensaje (error de escritura en memoria inexistente) sí es grave, por lo que este mensaje sí que aparecería en nivel 1 (Visual11 solamente carga código en el mapa de memoria que simula, por lo que esta parte del código que se pretende cargar en una zona de memoria no implementada sería obviado).

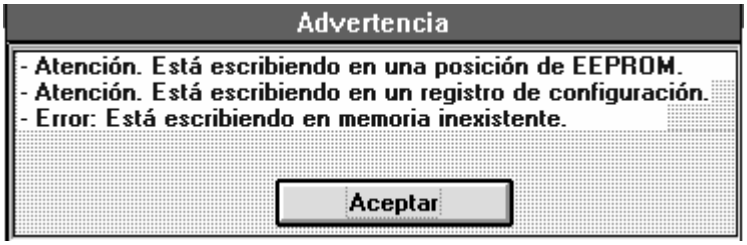


Figura 13. *Ventana que advierte de distintas circunstancias de escritura en memoria.*

Comentaremos el significado de algunos otros mensajes de error importantes. **Al compilar un programa**, si aparecen errores se visualiza en primer lugar la ventana superior de la Fig. 14, tras pulsar ACEPTAR desaparece ésta y surge la ventana inferior de la misma Fig., donde se presentan los errores encontrados y la línea del fichero de edición donde éstos se encuentran. Situando el ratón sobre el mensaje de error que se desea corregir y pulsando el botón izquierdo se despliega la ventana EDICION, apareciendo el cursor colocado justo en la línea donde se encuentra dicho error.

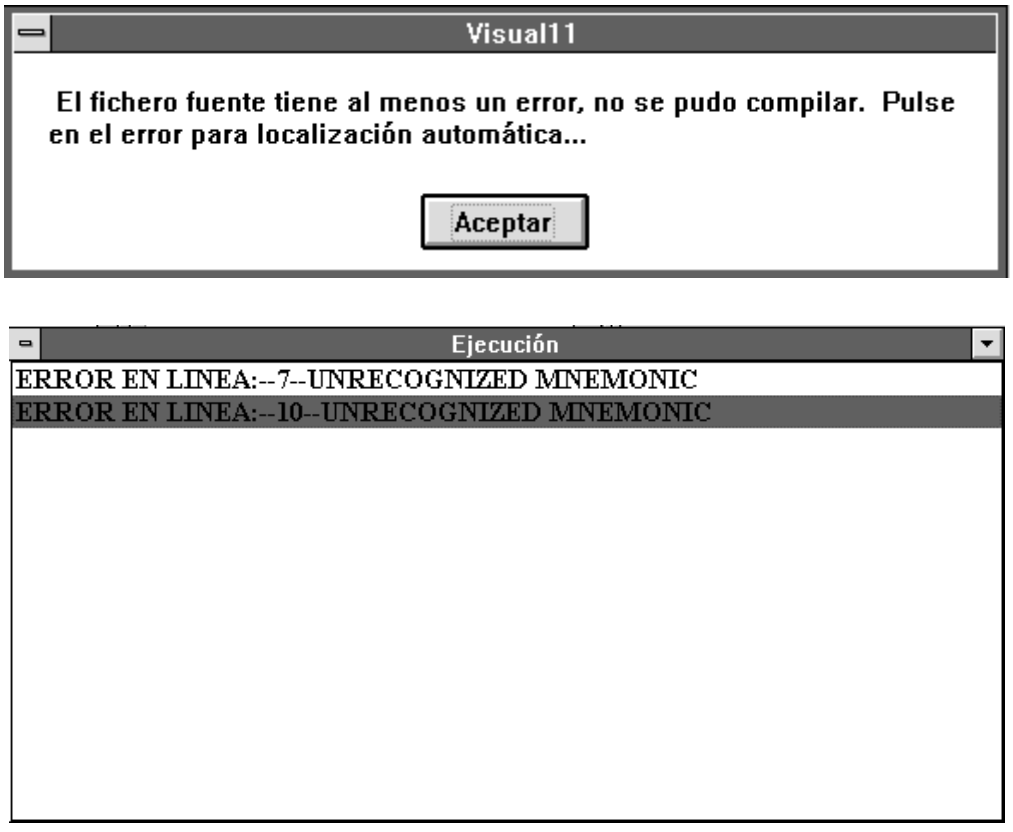


Figura 14. *Ventanas que indican errores de compilación.*

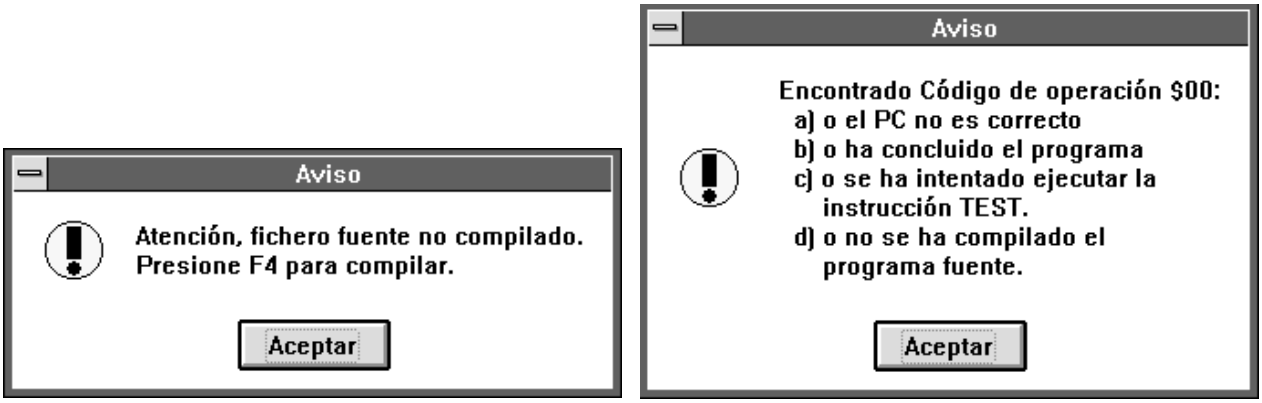


Figura 15. *Ventanas que indican distintas circunstancias al iniciar la ejecución de un programa ensamblador.*

En la Fig. 15 se muestran dos ventanas en las que se indican diversas circunstancias por las que no se puede iniciar la **ejecución (simulación) de un programa ensamblador**. En el caso del mensaje de la izquierda se ha editado un programa ensamblador, pero éste no ha sido compilado, por lo que no se puede proceder a su ejecución. En el caso del mensaje de advertencia de la izquierda se ha solicitado la ejecución de un programa pero la dirección apuntada por el PC se encuentra vacía, por lo que se advierte de ello y se indican cuatro posibles causas: a) el alumno no ha actualizado el valor del PC para que apunte a la primera instrucción del programa; b) tras ejecutar varias instrucciones el programa ha concluido, con lo que nos encontramos con casillas de memoria vacías; c) la instrucción TEST tiene por código de operación $00, por lo que quizás estemos tratando de ejecutar esta instrucción especial, que Motorola emplea en el test de los circuitos integrados y que no es válida en los modos normales de funcionamiento; d) la última posibilidad es que no se haya compilado el programa fuente.

4. Una Sesión de Trabajo.

Expuestas las características básicas de Visual11, y las distintas ventanas y opciones, ilustraremos a continuación el trabajo con Visual11 a través del desarrollo paso a paso de un ejemplo completo.

4.1 Edición, carga y compilación.

Desde el entorno MS-Windows ejecutaremos Visual11 pinchando dos veces con el ratón sobre su icono. Podemos observar que se despliegan las ventana ACERCA_DE..., la PRINCIPAL, y la que indica que se está inicializando la memoria del entorno de simulación.

Cuando acabe el proceso de inicialización, acudiremos al menú PROYECTO-> ABRE_UNO_EXISTENTE, y seleccionaremos el fichero C:\VISUAL11\ASM\INTER.ASM. Ante nuestros ojos surge entonces la ventana EDICION, con el código fuente del programa ensamblador. En el caso de que el lector desee editar su propio programa ensamblador deberá acudir a PROYECTO->CREA_UNO_NUEVO, y teclearlo siguiendo la sintaxis expuesta en el apartado 3.5 (se recomienda trabajar siempre en el directorio que se ofrece por defecto, C:\VISUAL11\ASM).

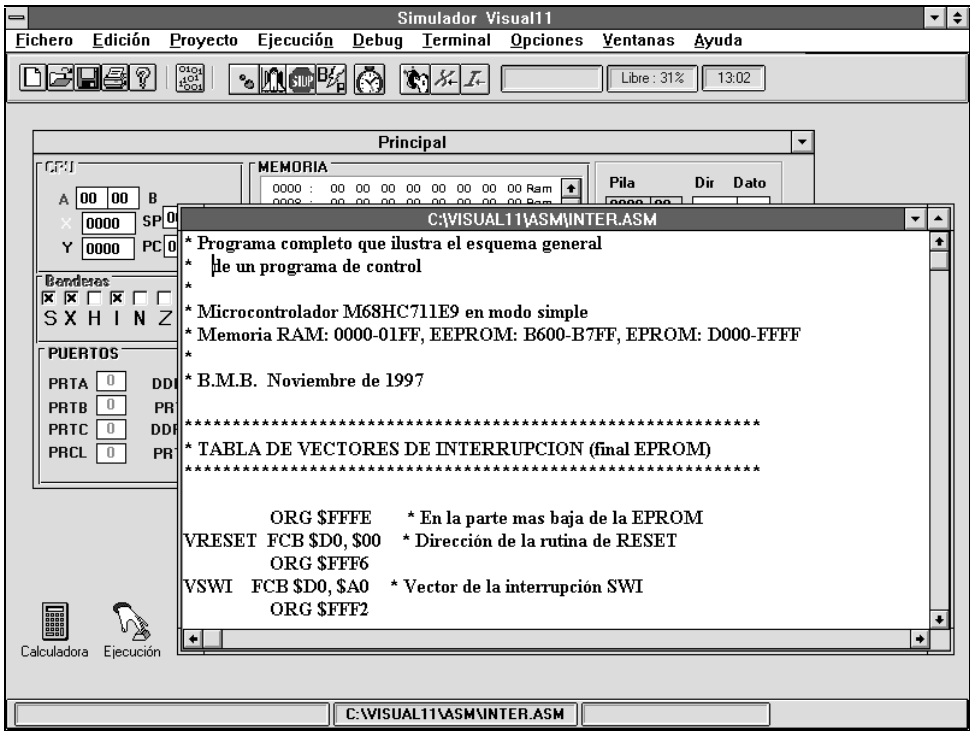


Figura 16. *Programa fuente INTER.ASM (ventana EDICION), en el entorno Visual11.*

A continuación se procederá a **compilar el programa**, bien seleccionando PROYECTO->COMPILA, bien pulsando la tecla F4, bien actuando con el ratón sobre el icono de compilación colocado en la barra de herramientas. Si el programa se compila con éxito[3], la ventana edición se plegará, Visual11 cargará en la memoria del 68HC11 simulado el código máquina resultante, y presentará la ventana EJECUCION (apartado 3.5), desde la que podremos seguir la ejecución paso a paso del programa (EJECUCION->PASO_ A_PASO, tecla F8, o botón de la barra de herramientas).

Si el usuario compila su propio programa y obtiene errores de compilación, recordemos que se explicó el tratamiento de los mismos en los apartados 3.5 y 3.7. Recordemos también los tres niveles de advertencia en los que puede operar Visual11 (OPCIONES->NIVEL_DE_ADVERTENCIA); el nivel 3 es el recomendado para un alumno que comienza.

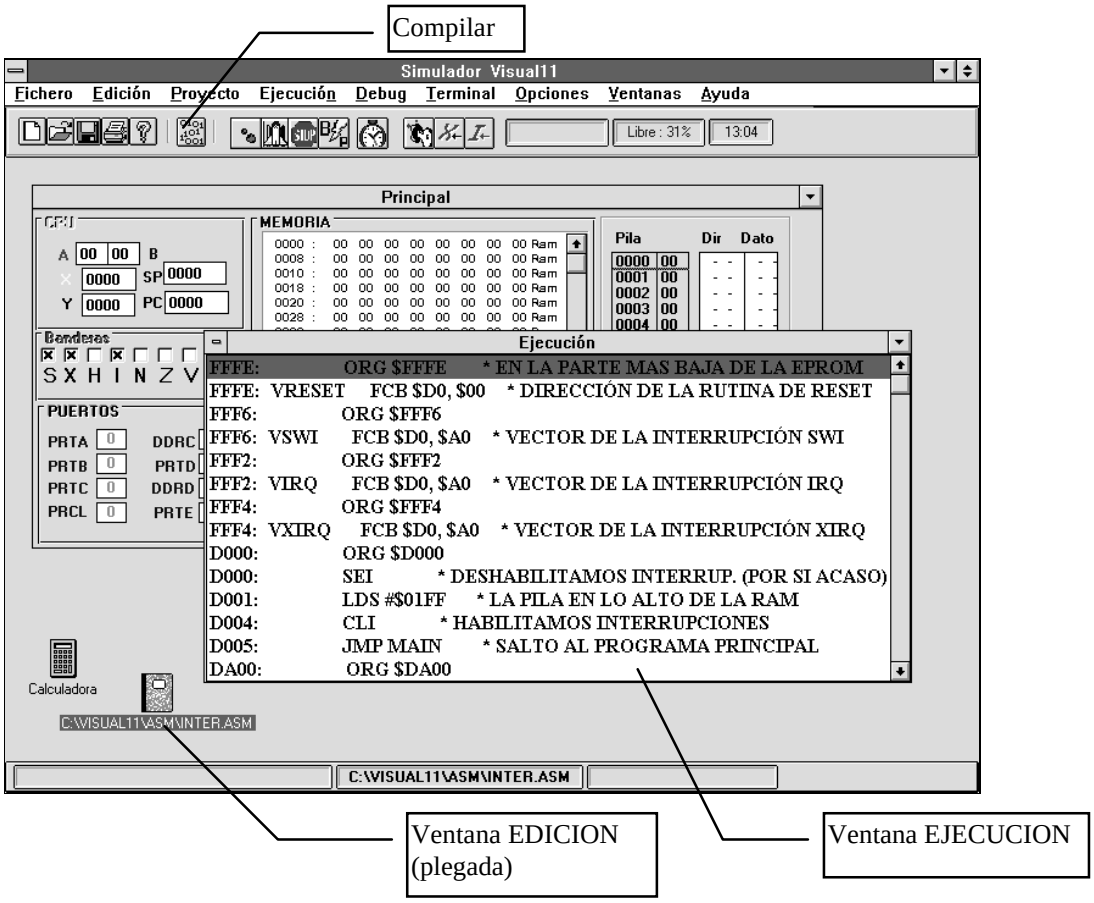


Figura 17. *Programa compilado INTER.ASM, en la ventana EDICION.*

[3] La compilación del fichero INTER.ASM que se emplea como ejemplo no debe plantear ningún problema. Sin embargo, si el usuario trabaja en Windows95 existe una pequeña posibilidad de encontrarse con la circunstancia de que la compilación parezca no concluir (como si el ordenador quedara 'colgado'), en cuyo caso debe consultar el comentario que se realiza en el apartado 1 del capítulo 8.

4.2 Simulación del programa.

El programa INTER.ASM que utilizamos como ejemplo es un programa de control sencillo pero completo, que se aloja en memoria EPROM (a partir de $D000), y que incluye todos los módulos típicos que requiere un microcontrolador para que pueda ser ejecutado nada más arrancar el sistema: programa de reset, programa principal y rutina de servicio de interrupciones enmascarables IRQ; además, en INTER.ASM se inicia la tabla de vectores de interrupción colocada al final del mapa de memoria (acaba en $FFFF).

El programa principal lee el conjunto de 8 interruptores conectados al puerto C y visualiza su estado en los 8 LED del puerto B, todo ello de forma continua (una y otra vez). Para probar el programa abriremos las ventanas de interruptores y LEDs, mediante OPCIONES->BLOQUES_HARDWARE->INTERRUPTORES y OPCIONES-> BLOQUES_HARDWARE->INTERRUPTORES, con lo que el entorno de trabajo adquiere el aspecto de la Figura 18.

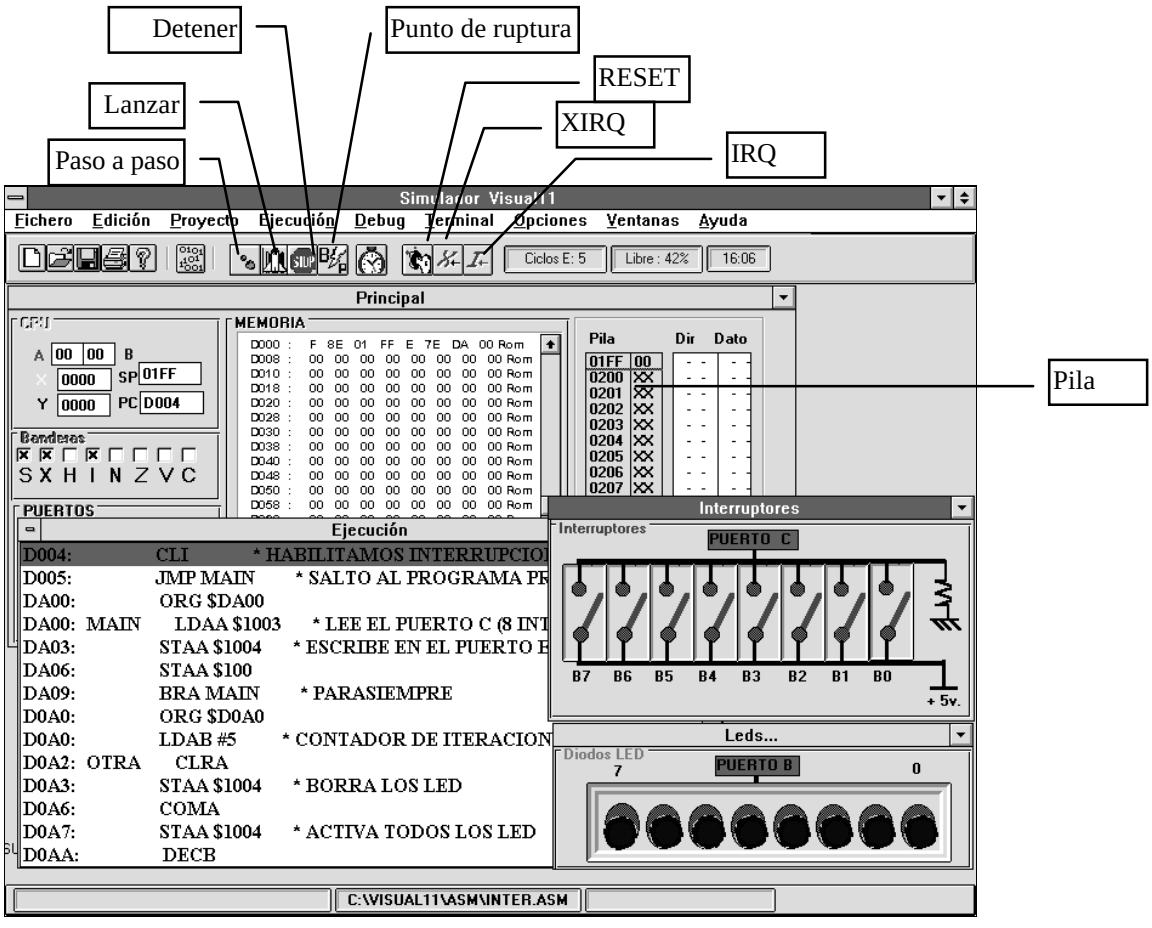


Figura 18. *Entorno Visual11 simulando INTER.ASM.*

Comenzaremos la **simulación del programa**. En primer lugar se debe **establecer el valor del registro PC** (contador de programa), lo que se puede realizar de tres formas:

- Cargar directamente en el PC (accediendo con el ratón) la dirección de comienzo del programa; en nuestro caso, la primera instrucción se aloja en $D000.

- Situar el ratón sobre la primera línea ejecutable del programa ($D000) dentro de la ventana EJECUCION y pulsar dos veces el botón izquierdo.
- Pulsar con el ratón el botón de reset de la barra de herramientas, el cual inicia el estado del microcontrolador y carga en el PC el valor guardado en las direcciones asignadas a la rutina de reset de la tabla de vectores de interrupciones, $FFFE y $FFFF, en las cuales tenemos almacenado el lugar de comienzo del programa, $D0 y $00, respectivamente. Esta última es la manera más próxima a la realidad (al alimentar el microcontrolador real se ejecutaría un reset que inicializaría el sistema y cargaría en el PC el valor guardado en la tabla de vectores).

Se comenzaría ya la **simulación paso a paso** del programa, para ello, o bien se acude al menú EJECUCION->PASO_A_PASO, o bien se pulsa la tecla F8, o bien se pulsa el correspondiente botón de la barra de herramientas.

Si pulsamos con el ratón sobre alguno de los interruptores simulados (ventana INTERRUPTORES, Fig. 18), observaremos que su estado cambia; el programa ensamblador lee dicho estado y lo visualizará sobre los LED, con lo que veremos que el diodo(s) correspondiente(s) se ilumina(n). Podemos **lanzar la simulación (*run*)** mediante el menú EJECUCION->LANZA_SIMULACION, o bien mediante la tecla F6 o pulsando el botón de la barra de herramientas correspondiente. Si realizamos la ejecución continua podremos ver cómo al modificar con el ratón el estado de un interruptor cambia también el del LED.

Desde un punto de vista pedagógico, uno de los aspectos más interesantes de Visual11 es la posibilidad de **simular interrupciones**, tanto enmascarables (IRQ), como no enmascarables (XIRQ), lo que permite al alumno observar paso a paso todos y cada uno de los detalles que acontecen en un proceso de interrupción, así como practicar con la programación de rutinas de servicio.

Para ilustrar este importante aspecto, en el programa ensamblador que estamos empleando como ejemplo hemos incorporado una rutina de servicio IRQ, de manera que cuando solicitemos interrupción (pulsando con el ratón sobre el botón de la barra de herramientas asignado a IRQ), el HC11 acudirá a las direcciones de la tabla de vectores asignados a IRQ, $FFF2 y $FFF3, tomará los datos allí guardados, $DA y $00 en nuestro caso, y compondrá la dirección donde comenzará la rutina de servicio, $DA00, la cual será cargada en el PC, prosiguiendo el HC11 con la ejecución de instrucciones desde dicha dirección. En este ejemplo la rutina de servicio de IRQ programada realiza cinco encendidos y apagados de los 8 LED, devolviendo luego el control al programa original, con lo que el alumno observará que el microcontrolador prosigue con su tarea anterior de leer interruptores y visualizar en los LED. El alumno puede ver también cómo en el servicio de IRQ el HC11 inhibe la posibilidad de otras interrupciones IRQ activando automáticamente el bit I (máscara de IRQ), y cómo preserva automáticamente el estado de todos sus registros en **la pila del sistema** (que ocupa una zona especial de la ventana PRINCIPAL, Fig. 18), para luego recuperarlos cuando concluya la rutina de servicio.

Estas posibilidades incorporadas en Visual11 permiten al alumno, además de aprender los fundamentos de la programación en ensamblador, practicar también el control de periféricos sencillos, la gestión de interrupciones o, incluso, el diseño de programas de control completos, incluyendo todos los elementos necesarios en una aplicación real, de manera tal que si fuesen cargados en un 68HC11 real éste los ejecutaría nada más conectar su alimentación (establecimiento de vectores de interrupción, inicio de la pila y puertos, programa principal y rutinas de servicio).

En este sentido, **Visual11 acerca al alumno a la realidad mucho más que otros simuladores disponibles**.

Además, hemos incorporado en Visual11 una opción que lo convierte en una terminal de comunicaciones (TERMINAL), de modo que un alumno en una fase posterior de su aprendizaje podrá **cargar un programa simulado en una tarjeta real basada en un 68HC11** conectada al puerto serie del PC. Dado el amplio uso en el entorno educativo de la tarjeta de evaluación de Motorola MC68HC11EVBU, que es gestionada mediante al programa monitor BUFFALO grabado en su ROM interna, hemos desarrollado esta parte del programa pensando en dicha tarjeta concreta (incluso el alumno puede construirse fácilmente una tarjeta de características similares que podría también manejar desde Visual11 [Martín del Brío 97b]). Esta TERMINAL se describirá en el Capítulo 6.

5. Algunos Ejemplos.

Presentamos a continuación unos cuantos ejemplos de programas ensamblador para el 68HC11, adaptados al mapa de memoria del 711E9 simulado por Visual11; el usuario puede encontrar dichos programas en el directorio que emplea Visual11 por defecto C:\VISUAL11\ASM (se recomienda trabajar en él siempre). Resaltaremos las características pedagógicas incorporadas en Visual11 que en cada uno de los ejemplos expuestos resultan de mayor interés. Una común a todos ellos es que el alumno, como en el caso de un microcontrolador real, se verá forzado a trabajar adecuadamente con el mapa de memoria disponible (RAM, EPROM, EEPROM). Por otro lado, si trabaja en el nivel de advertencia 3 (OPCIONES->NIVEL_DE_ ADVERTENCIA->NIVEL_3) el programa en todo momento le avisará de posibles errores (en el nivel 1 se limita a los de mayor gravedad).

5.1 Copia de una cadena de caracteres.

Este primer programa ensamblador, COPIA.ASM, copia una cadena de caracteres almacenada a partir de la dirección $00 de RAM a la dirección $50; el fin de cadena viene dado por el código EOT (*End of Transmision*), de valor hexadecimal $04. Hemos alojado el código en memoria RAM también. Con ejercicios de este tipo, el alumno puede aprender los fundamentos de la programación en ensamblador; Visual11 le resulta de gran ayuda, puesto que puede editar, compilar y simular dentro de un mismo entorno Windows convencional.

```
* PROGRAMA: Copia una cadena de 10 caracteres

****************** Datos ***************
        ORG $00
INI     FCB 'H, 'o, 'l, 'a, $04          *cadena inicial

        ORG $50
FIN     FCB 0                            * cadena final

****************** Programa ***************
        ORG $100

        LDX  #INI    * X e Y como índices de las cadenas
        LDY #FIN
otra    LDAA 0,X
        STAA 0,Y
        INX
        INY
        CMPA #$04
        BNE otra

        END
```

5.2 Visualización en 2x7seg MUX

Visual11 simula también algunos periféricos sencillos conectados a sus puertos B y C. El programa DISPLAYS.ASM ilustra el manejo en forma multiplexada de los dos visualizadores de siete segmentos: ambos están excitados por un único conversor BCD a siete segmentos (Fig. 12 y Fig. 19), de modo que se debe seleccionar aquel de los dos que queremos que actúe; para ello debe ponerse un '1' en la línea del puerto B adecuada (línea 4 para el visualizador de la cifra de las decenas, línea 5 para las unidades), y enviar el valor BCD adecuado por las líneas 0 a 3. Este bloque de visualizadores de siete segmentos se ha programado de manera que represente tanto los dígitos BCD '0' a '9', como también los hexadecimales 'A' a 'F'.

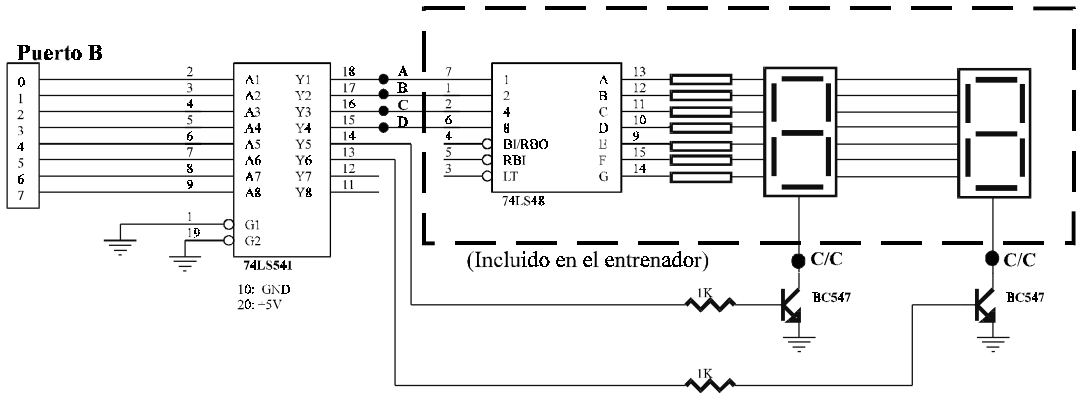


Figura 19. *Esquema eléctrico de los visualizadores de siete segmentos simulados por Visual11.*

El programa guarda en dos casillas de memoria el valor de la cifra de las decenas y de las unidades, las completa con el bit habilitador correspondiente, y las envía una y otra vez al puerto. Si mientras se ejecuta el programa accedemos a las casillas $00 y $01 de memoria y cambiamos el valor de las unidades y decenas observaremos cómo las nuevas cifras se visualizan en los siete segmentos. Se recomienda realizar una ejecución paso a paso primero, y continua después; en una aplicación real, basta activar cada visualizador unas 20 veces por segundo para que el ojo crea que ambos permanecen constantemente encendidos.

```
* PROGRAMA:  Manejo multiplexado de dos
*       displays de 7 segmentos desde el puerto B

****************** Etiquetas ********************
PORTB   EQU  $1004

******************  Datos ******************
        ORG $0
DECE                    FCB 3   * cifra decenas
UNID    FCB 7   * cifra unidades

****************** Programa ******************
        ORG $100
otra    LDAA DECE       *activar decenas
        ORA #%00010000
        STAA $1004

        LDAB UNID       * activar unidades
        ORB #%00100000
        STAB $1004

        BRA otra        * una y otra vez

        END
```

5.3 Controlador de teclado.

Otro de los periféricos simulados en Visual11 es un **teclado matricial de 16 teclas** (hexadecimal) no decodificado (Fig. 12), que puede conectarse al puerto C (de entrada/salida) del 68HC11; dicho teclado debe gestionarse mediante la clásica exploración de teclado (*scan*).

En este caso el simulador considera que las cuatro líneas más significativas del puerto C (4 a 7) están programadas como salidas, las cuales se conectarán a las filas del teclado, y las cuatro menos significativas (0 a 3) como entradas, conectándose a las columnas. En un 68HC11 real habría que actuar sobre el registro DDRC (dirección de los datos del puerto C, $1007) en él habría que grabar la palabra $11110000, para programar las cuatro líneas más significativas como entradas y las demás como salidas, tal y como el simulador considera.

El programa ensamblador TECLADO.ASM en primer lugar programa el DDRC (simplemente para recordar al alumno que en un sistema real debería hacerlo). A continuación el programa ejecuta la exploración de teclado, enviando un '1' alternativamente por cada una de las filas del teclado, y leyendo las cuatro columnas; cuando el programa encuentra una lectura diferente a %0000 en las columnas interpreta que una tecla ha sido pulsada, por lo que sale del bucle de rastreo y procede a consultar una tabla guardada en EEPROM, en la que se almacenan las distintas combinaciones fila-columna, junto a su valor ASCII correspondiente, que es la salida que el programa proporciona en el acumulador B. El programa no proseguirá con un nuevo proceso de búsqueda de tecla hasta que la tecla pulsada haya sido liberada.

En este sentido, recordemos que pulsando con el ratón una vez sobre la imagen de una tecla del teclado simulado, ésta se mantiene presionada (situación indicada representando la tecla en color amarillo); pulsar de nuevo con el ratón equivale a liberar dicha tecla.

```
*****************************************************
*** PROGRAMA:  Controlador de teclado ****
*****************************************************
          ORG $100

* En un HC11 real hay que programar el DDRC
          LDAA #$F0    * 0-3 in, 4-7 outs
          STAA $1007  * DDRC

*********************************
*** Rastreo de filas (scan)
*********************************

rastre    LDAA #%00001000 * A es la fila
          CLC

scan      LSLA           * siguiente fila (izda)
          BCC conti
          LDAA #%00010000
conti     STAA $1003      * sacar por Port C
          LDAB $1003     * leemos Port C
          ANDB #%00001111 * solo columnas
          BEQ  scan       * ¿tecla pulsada?

          LDAB $1003     * leemos Port C
* en B queda código tecla, B=fila:columna
```

```
*********************************
*** Consulta tabla para ASCII tecla pulsada
*********************************

tabla     LDX #$B600      *  tabla en $B600
NoEs      CMPB 0,X
          BEQ  SiEs
          INX
          INX
          CPX #$B61f       * ultima posicion tabla
          BLS NoEs
          LDAB #00         * si fin tabla, b<-00...
          BRA rastre       *  ... y mas scan
SiEs      LDAB 1,X         * en B ASCII tecla

*********************************
*** Visualizar (A)
*********************************

espera    LDAA $1003      * Espera soltar tecla
          ANDA #%00001111 * mask MSB
          BNE espera

          JMP rastre      * volver al comienzo

*** fin programa principal
```

```
*********************************
*** Tabla equivalencias fila:columna->ASCII
*********************************

      ORG $b600

      FCB $88, '1'
      FCB $84, '2'
      FCB $82, '3'
      FCB $81, 'F'
      FCB $48, '4'
      FCB $44, '5'
      FCB $42, '6'
      FCB $41, 'E'
      FCB $28, '7'
      FCB $24, '8'
      FCB $22, '9'
      FCB $21, 'D'
      FCB $18, 'A'
      FCB $14, '0'
      FCB $12, 'B'
      FCB $11, 'C'

      END
```

5.4 Interrupciones.

El programa INTER.ASM se empleó ya en el Capítulo 4 para ilustrar el manejo y posibilidades del entorno Visual11. Se trata de un programa de control completo, que incluye todos los módulos necesarios en un microcontrolador real para que nada más conectar la alimentación comience su operación. Incluye: tabla de vectores de interrupción, rutina de arranque (reset), programa principal y rutina de servicio de IRQ, todo ello alojado en EPROM.

La tabla de vectores establece el lugar donde el microcontrolador debe comenzar a ejecutar instrucciones (dirección alojada en $FFFE y $FFFF), y la dirección de la rutina de servicio (alojada en $FFF2 y $FFF3). La rutina de arranque inicia la pila (colocada en memoria RAM), y habilita las IRQ. El programa principal lee los interruptores conectados al puerto C y visualiza su estado en los LED conectados al B. Finalmente, la rutina de servicio ejecuta cinco encendidos y apagados de todos los LED cuando sea solicitada una interrupción IRQ (actuando con el ratón sobre el botón correspondiente de la barra de herramientas).

```
* Programa completo que ilustra el esquema
*    general de un programa de control
*
* M68HC711E9 en modo simple
* RAM: 0000-01FF, EEPROM: B600-B7FF
* EPROM: D000-FFFF
*
* B.M.B.  Noviembre de 1997

*******************************************************
* TABLA DE VECTORES DE INTERRUPCION
(final EPROM)
*******************************************************

          ORG $FFFE      * EPROM mas baja
VReset    FCB $D0, $00   * Dir rutina de RESET
          ORG $FFF6
VSWI      FCB $D0, $A0   * Vector de la SWI
          ORG $FFF2
VIRQ      FCB $D0, $A0   * Vector de la IRQ
          ORG $FFF4
VXIRQ     FCB $D0, $A0   * Vector de la XIRQ

*******************************************************
***** Comienzo del programa*****************
*******************************************************

*******************************************************
* RUTINA DE RESET (comienzo EPROM)
*******************************************************

          ORG $D000
          LDS #$01FF     * La pila RAM alta
          CLI            * habilitamos IRQ
          JMP MAIN       * salto al principal

* Fin de la rutina de reset

*******************************************************
* PROGRAMA PRINCIPAL (en EPROM)
*******************************************************

          ORG $DA00
MAIN      LDAA $1003     * Lee puerto C
          STAA $1004     * Escribe en puerto B
```

```
        BRA MAIN        * parasiempre


*********************************************************
* RUTINA DE SERVICIO DE LAS
INTERRUPCIONES  (EPROM)
*********************************************************

* 5 encendidos y apagados de los 8 LED
        ORG $D0A0
        LDAB #5         * contador de iterac.
otra    CLRA
        STAA $1004      * borra los LED
* en un programa de verdad aqui bucle retraso
        COMA
        STAA $1004      * activa todos los LED
* en un programa de verdad aqui bucle retraso
        DECB
        BNE otra        * ¿otra vez?
        RTI

        END     * FIN DE FICHERO
```

6. Terminal de Comunicaciones

Este capítulo está orientado a un estudiante más avanzado, que ya ha trabajado a nivel de simulación con el 68HC11, y se adentra en una segunda fase de aprendizaje donde trabajará con hardware real. El lector que se acerca por primera vez al microcontrolador 68HC11 y al entorno Visual11 puede saltarse este capítulo sin ningún perjuicio para comprender lo que viene a continuación.

6.1 El trabajo con hardware real.

Recomendamos al estudiante que se acerca por primera vez al campo de los microcontroladores trabajar a nivel de simulación. Superada esta primera fase de aprendizaje, en la que con ayuda de Visual11 habrá asimilado la arquitectura de la CPU-11, su programación en ensamblador, y habrá trabajado los aspectos más básicos de las entradas-salidas, el estudiante debería acceder a una segunda fase, consistente en el trabajo con hardware real. En el campo de la electrónica obviamente la simulación nunca debe sustituir al trabajo con circuitería real, tan solo debe emplearse como una ayuda a la enseñanza; el estudiante debe en una segunda fase afrontar los problemas asociados al desarrollo de sistemas microprocesadores reales, donde intervienen características eléctricas y temporales difícilmente plasmables en toda su extensión en un simulador.

Para este cometido recomendamos la **tarjeta de evaluación de Motorola MC68HC11EVBU** [Motorola 92], sistema mínimo que incluye un MC68HC11E9 trabajando en modo sencillo. Esta tarjeta de relativamente bajo coste puede manejarse desde el puerto serie de cualquier PC compatible, o bien desde una simple terminal serie clásica. El microcontrolador MC68HC11E9 que incluye dicha tarjeta posee una ROM interna de 12KB, en la que se encuentra el programa monitor BUFFALO, a modo de pequeño sistema operativo que permite editar, compilar, ejecutar paso a paso y de forma continua programas ensamblador. Ya que buena parte de los puertos y recursos internos del microcontrolador que incluye la tarjeta EVBU son accesibles, un alumno con la ayuda de un PC puede realizar en un laboratorio de electrónica de una manera relativamente sencilla prácticas hardware interesantes, como realizar juegos de luces, controlar pequeños motores, visualizadores y teclados, construir un termostato digital, etc.

Una interesante posibilidad para el alumno consiste en construir su propia tarjeta, similar a la EVBU; la propia Motorola propone esquemas [Motorola 88] y software para su manejo [Motorola 94]. En la Universidad de Zaragoza hemos recopilado información de este tipo, que suministramos a los alumnos [Martín del Brío 97b].

Para controlar este tipo de tarjetas puede hacerse uso de software como PCBUG11 de Motorola [Motorola 94], o si el microcontrolador incluye BUFFALO, desde un programa tipo terminal convencional (como el incluido en MS-Windows 3.11).

En Visual11 decidimos incluir como una opción más la posibilidad de disponer de una **ventana que haga el papel de terminal de la tarjeta EVBU**, o de cualquier otra basada en el HC11 que posea BUFFALO en su ROM, para que de este modo el alumno en su segunda fase de aprendizaje siga manejándose en el mismo entorno, desde el que puede **editar, compilar, simular, cargar en una tarjeta real y ejecutar el programa** residente ya en la memoria interna del microcontrolador.

6.2 Características de la terminal de comunicaciones.

Describiremos en este apartado la TERMINAL de Visual11, que permite trabajar con una tarjeta basada en un 68HC11 con monitor BUFFALO. Para acceder a ella hay que situarse en el menú TERMINAL, que contiene tres opciones. La primera es TERMINAL-> PARAMETROS_DE_LA_COMUNICACION, que tiene por defecto asignados los valores adecuados para la comunicación con BUFFALO. TERMINAL-> ENTRAR_EN_LA_TERMINAL genera una pantalla que hace de terminal de BUFFALO (Fig. 20), desde la cual el alumno puede acceder a la tarjeta HC11EVBU, cargar un programa, ejecutar paso a paso, programar la EEPROM del HC11, etc. TERMINAL-> PROGRAMAR_HC11_CON_PROYECTO permite cargar un fichero compilado (S19) en el 68HC11 de la tarjeta.

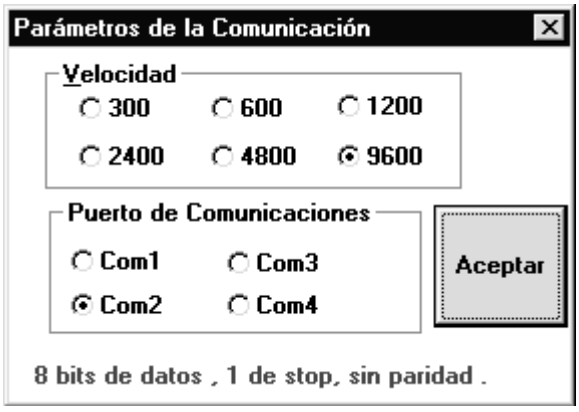
Exponemos a continuación las tres opciones con mayor detalle.

Terminal

Este menú convierte Visual11 en una terminal de la placa de evaluación 68HC11EVBU de Motorola, o de cualquier placa que posea en ROM el programa monitor BUFFALO de Motorola.

Parámetros de la comunicación

Permite establecer los valores de comunicación de la tarjeta. Se incluye por defecto el estándar de BUFFALO (9600 baudios, 8 bits, 1 bit stop y sin paridad, por COM2), pudiéndose cambiar la velocidad de transmisión y el puerto de comunicaciones serie del PC a emplear.



Entrar en la terminal (Ctrl+T)

Al activar esta opción surge una nueva ventana que hace de TERMINAL de una tarjeta con BUFFALO; desde ella pueden ejecutarse los comandos de BUFFALO (‘R’, ‘S’, ‘G’, ‘A’, ‘D’, ‘M’, etc.). Se han incorporado algunos botones que ejecutan directamente algunos de los comandos más habituales, como los que permiten ejecutar el programa paso a paso, introducir puntos de ruptura, lanzar la ejecución, cambiar la velocidad de transmisión a 9600 o 300 baudios (esta última es la requerida para cargar un programa en EEPROM), y cargar un fichero S19 del disco duro.

Seleccionando esta opción se accede a un **nuevo menú** específico para la comunicación con una tarjeta real basada en un 68HC11. Dicho menú se expondrá detalladamente en el siguiente apartado.

Programar HC11 con proyecto (Ctrl+F9)

Carga el fichero S19 con el que actualmente está trabajando el simulador de Visual11 en la memoria del microcontrolador de la tarjeta conectada al puerto serie del PC.

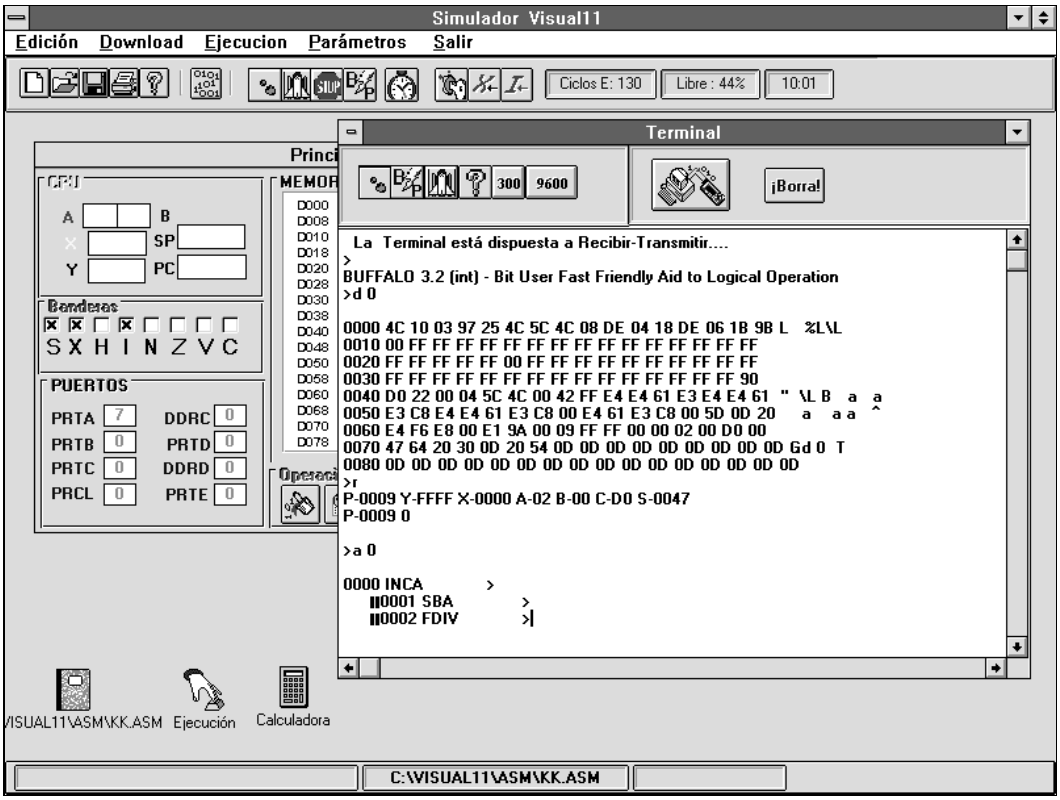


Figura 20. *Terminal de comunicaciones con una tarjeta basada en BUFFALO.*

6.3 Nuevo menú de TERMINAL.

Al acceder a TERMINAL-> ENTRAR_EN_LA_TERMINAL el menú principal de Visual11 cambia, adaptándose a los requisitos típicos del trabajo con una tarjeta como la EVBU. Su nuevo aspecto es el que aparece en la siguiente figura:

Expondremos a continuación el contenido de cada uno de estos nuevos submenús asociados a la TERMINAL.

Edición

Típico menú Windows con herramientas de edición que se apoyan en el portapapeles. Estas acciones se aplican a la selección realizada con el ratón sobre la ventana TERMINAL. Su mayor utilidad es evitar que el usuario tenga que introducir carácter a carácter, una y otra vez, comandos de BUFFALO.

Copiar (Ctrl+C)

Copia al portapapeles el texto seleccionado con el ratón en la ventana TERMINAL.

Cortar (Ctrl+X)

Permite cortar el texto seleccionado en la ventana TERMINAL, llevándolo al portapapeles.

Pegar (Ctrl+V)

Pega el texto guardado en el portapapeles en el punto de la ventana TERMINAL donde se encuentre el cursor. Sirve sobre todo para volver a ejecutar comandos ya empleados con anterioridad, ahorrando tiempo de escritura al usuario.

Borrar

Borra el texto seleccionado en la ventana TERMINAL.

Limpiar el buffer

Borra la pantalla de la TERMINAL.

Download

Envía un fichero código máquina (de extensión S19) a la tarjeta para su carga en memoria, tanto RAM como EEPROM (estableciendo en este último caso la velocidad de transmisión a 300 baudios por segundo, para su correcta programación).

Ejecución

Menú de comandos de ejecución del programa cargado en el 68HC11 real.

Paso a paso (F8)

Envía el comando 'T' (*trace*) al BUFFALO residente en la tarjeta del 68HC11; este comando ejecuta una sola instrucción.

Ejecución continua

Envía el comando 'R' (*run*) al BUFFALO residente en la tarjeta del 68HC11, ejecutando un programa.

Breakpoints

Envía el comando 'B' (*breakpoint*) al BUFFALO residente en la tarjeta del 68HC11; insertando un punto de ruptura.

Parámetros

Accede de nuevo a la ventana de parámetros de la comunicación serie, ya descrita en el apartado anterior.

Salir

Abandona la TERMINAL.

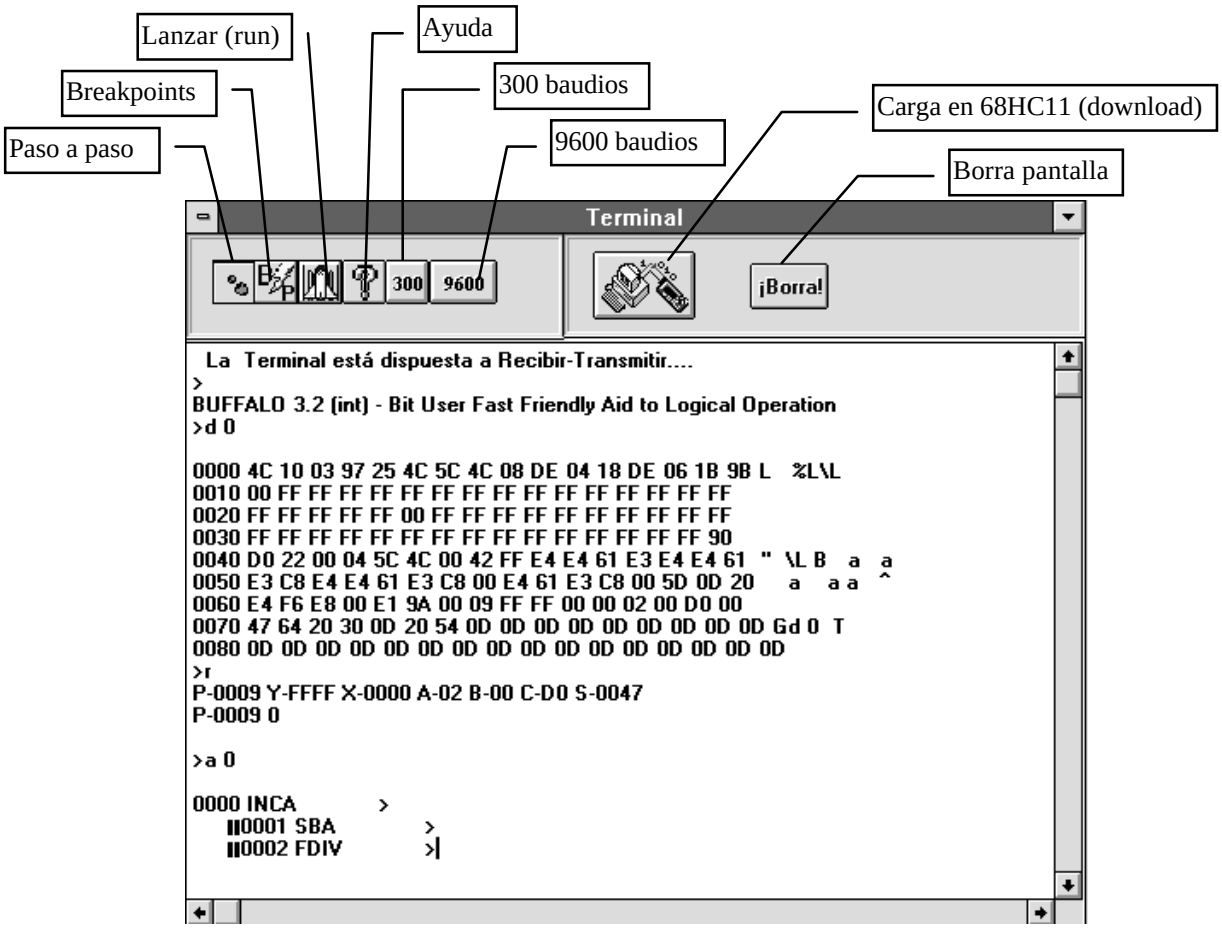


Figura 21. *Ventana de comunicaciones de TERMINAL.*

Nota final. *Esta terminal de Visual11 puede presentar problemas al tratar de comunicar con una tarjeta externa basada en el 68HC11 si el ordenador desde el que se ejecuta Visual11 no es lo suficientemente rápido, debido a las dificultades que presenta el sistema Windows para sincronizar eventos, y debido a los elevados recursos del procesador del PC que Windows acapara. Así, la terminal ha funcionado correctamente en un Pentium 166 MHz con Windows 95, pero presentaba algún problema ocasional (pérdida de datos) en un Pentium 100 MHz con Windows 3.11 (en este último caso había que cerrar el resto de aplicaciones y ventanas para que funcionase correctamente).*

7. Otros Recursos para el Aprendizaje del M68HC11.

El 68HC11 es uno de los microcontroladores más empleados en el entorno educativo a nivel mundial, por lo que existe a disposición del usuario una enorme cantidad de material de apoyo a la enseñanza, si bien es verdad que apenas hay material disponible en lengua española (Visual11 es una de las pocas excepciones). Expondremos en este apartado algunos recursos de interés para el aprendizaje del microcontrolador 68HC11

7.1 Libros

Quizás el 68HC11 sea el microcontrolador que más libros educativos tratan a nivel mundial (aunque apenas existe literatura en castellano sobre él); a continuación incluimos unas cuantas referencias que hemos manejado y que nos parecen recomendables. Los textos a nuestro juicio más interesantes para su empleo en una asignatura de microcontroladores son el de Spasov, extenso y completo, y sobre todo el de Huang, no tan extenso, pero muy claro y con muchos ejemplos. El de Driscoll y otros es un libro muy interesante, en el que se desarrollan detalladamente (tanto a nivel de hardware como de software) diversas aplicaciones del 68HC11 a tareas de adquisición de datos y control. Finalmente, el propio manual de referencia de Motorola puede ser empleado casi como un libro de texto; es excelente.

[Driscoll 94] Driscoll, F.F., Coughlin, R.F., Villanucci, R.S. *Data Adquisition and Process Control with the M68HC11 Microcontroler*. Prentice-Hall, 1994.

[Huang 96] Huang, H.W. *MC68HC11. An Introduction (Software and Hardware Interfacing)*. West Publishing Company, 1996.

[Lipovski 88] Lipovski, G.J. *Single and Multiple-chip Microcomputer Interfacing*. Prentice-Hall, 1988.

[Miller 93] Miller, G.H. *Microcomputer Engineering*. Prentice-Hall, 1993.

[Motorola 91] *M68HC11 Reference Manual*. Motorola, 1991. (disponible en formato pdf en http://www.motorola.com).

[Spasov 96] Spasov, P. *Microcontroler Technology: the 68HC11*. 2ed, Prentice-Hall, 1996.

Como hemos dicho, desafortunadamente apenas hay textos en castellano que traten este microcontrolador, tan solo podemos indicar un par de referencias construidas como apuntes de clase. Por un lado, [Martín del Brío 97a] es un texto genérico sobre microprocesadores y microcontroladores, que toma el 6800 y 68HC11 como ejemplos; la referencia [Sosa 95] trata exclusivamente el 68HC11, explicando todos sus bloques internos casi a modo de manual de usuario.

[Martín del Brío 97a] B. Martín del Brío. *Curso de Microprocesadores y Microcontroladores (6800 y 68HC11).* Universidad de Zaragoza, 1997.

[Sosa 95] J. M. Sosa Navarro. *Teoría y Práctica del Microcontrolador MC68HC11E9.* E.U.I.T.T. Universidad de Las Palmas de Gran Canaria, 1995.

7.2 Software e internet.

Existe una buena cantidad de software relacionado con el 68HC11, tanto gratuito como de pago: ensambladores, compiladores de C, de lógica *fuzzy*, entornos de trabajo y simuladores. Recomendamos al lector consultar con las casas que habitualmente desarrollan sistemas para microprocesadores, pues casi todas incluyen software para el 68HC11 en sus catálogos. En el libro [Spasov 96] se incluyen unas cuantas referencias sobre software gratuito y direcciones internet.

Para conseguir tanto programas, como todo tipo de información y documentación, puede recurrirse a cualquiera de las direcciones internet de Motorola, por ejemplo http://www.motorola.com. De dichos nodos puede obtenerse incluso el famoso *M68HC11 Reference Manual* ("el libro rosa"), en la forma de fichero pdf, que puede leerse en cualquier ordenador personal con el Acrobat Reader de la casa Adobe, también disponible gratuitamente en internet.

Presentamos a continuación unas cuantas direcciones internet desde las que se puede empezar a navegar en busca de información útil; en realidad la cantidad de información disponible en internet sobre el 68HC11 es enorme, y los sitios a visitar son innumerables. En ellas puede encontrarse información de productos, hojas de características, notas de aplicación, compiladores, simuladores, esquemas de montajes, etc.

Webs de Motorola: http://www.motorola.com http://www.mot.com
http://freeware.aus.sps.mot.com

Web de literatura de Motorola: http://design-net.com

Webs varios: http://www.amc.com/chipdir/6811.html (un buen sitio para empezar a curiosear, con múltiples conexiones)
http://www.cam.org/~bisson (entorno de trabajo)
ftp://nyquist.ee.ualberta.ca/pub/motorola/mcu11 (software HC11)
http://zape.unizar.es

Directorio de circuitos integrados: http://www.xs4all.nl/~ganswijk/chipdir/chipdir.html

Para los lectores que gustan de estar a la última (o quizás penúltima, quien sabe), señalar que existen diversos entornos que permiten el desarrollo de sistemas de control borrosos (*fuzzy*) con el 68HC11; en el propio web de Motorola se proporciona información, tutoriales, demos e, incluso, un programa de dominio público.

7.3 Hardware para el 68HC11.

Simplemente recordar lo ya indicado en el punto anterior: a) todas las casas importantes que desarrollan sistemas de desarrollo para microprocesadores incluyen herramientas para el 68HC11; b) existe numerosa información disponible en internet.

Recordar también que la tarjeta de Motorola MC68HC11EVBU es una interesante herramienta de relativamente bajo coste, y que un alumno puede construirse fácilmente su propia tarjeta: el 68HC11 en este sentido es muy "agradecido", ya que construir una placa basada en él resulta fácil, barato, la programación de su EEPROM interna no requiere circuitería adicional, y Motorola proporciona todo tipo de información y software de dominio público para ello (información de este tipo la recopilamos en el "Kit del 68HC11").

[Martín del Brío 97b] B. Martín del Brío. *Kit de Iniciación al microcontrolador 68HC11*. Dept. Ingeniería Electrónica y Comunicaciones, Universidad de Zaragoza. Revisión nº 2, noviembre 1997.

[Motorola 88] MC68HC11 *EEPROM programming from a personal computer*. Application Note AN1010, Motorola, 1988.

[Motorola 92] *M68HC11EVBU Universal Evaluation Board User´s Manual*. Motorola, 1992.

[Motorola 94] *PCBUG11 User´s Manual.* Motorola, 1994.

8. Notas Finales sobre Visual11.

La actual versión de Visual11 es la 0.95, la cual se está empleando ya en prácticas de microprocesadores en la E.U. de Ingeniería Técnica Industrial de Zaragoza. A la luz de la experiencia obtenida en su empleo por los estudiantes, estamos incorporando modificaciones que mejorarán su empleo en prácticas, y corrigiendo los pequeños detalles de funcionamiento que siempre aparecen en una nueva aplicación.

8.1 Problemas conocidos ("bugs").

Presentamos una lista de pequeños problemas que el usuario de la versión 0.95 de Visual11 podría encontrar en su uso intensivo, junto a sugerencias para su solución. En general, dichos problemas ni son de importancia, ni ocasionan inconvenientes en una sesión de trabajo convencional. Muchos de estos pequeños detalles de operación pueden estar corregidos ya cuando el usuario lea este manual.

Si el usuario detecta algún problema de funcionamiento diferente a los que se exponen, se agradecería el envío de una nota a la dirección de correo electrónico `nenet@posta.unizar.es`, describiendo el problema encontrado, y añadiendo el tipo de sistema operativo bajo el que se ha ejecutado el programa (MS-Windows 3.1, 3.11, 95, ...), y las características del ordenador utilizado: tipo de procesador (486DX33, 66, Pentium 75, 100, 166, Pentium II, etc.), memoria RAM, tarjeta gráfica y resolución empleada.

Problemas en la compilación.

- Visual11 hace uso del compilador de dominio público de Motorola AS11, por lo que el usuario en ocasiones puede encontrarse con problemas relacionados con la compilación, que no son de Visual11 sino de AS11. Así, el ejecutable AS11.EXE se encuentra en el directorio C:\VISUAL11\ASM, el cual por defecto se considera el directorio de trabajo. Ocasionalmente se ha observado algún problema al tratar de compilar ficheros ensamblador que se guardan en un directorio distinto al directorio donde está colocado el compilador. Por este motivo, **se recomienda trabajar siempre en el directorio C:\VISUAL11\ASM, establecido por defecto**.
- En línea con el punto anterior, se han observado problemas a la hora de compilar las dos primeras instrucciones del ensamblador del 68HC11, **ABA y ABX** (afortunadamente no son instrucciones de uso excesivamente frecuente, y pueden suplirse fácilmente por otras). No se han observado problemas con ninguna otra instrucción.

- Visual11 ordena la ejecución de **AS11** a través del fichero de **ENS.PIF**. **En Windows95** ocasionalmente se ha observado que en función de las opciones establecidas en ENS.PIF al ordenar una compilación parece que ésta no se lleva a cabo (se observa el cursor con el típico reloj de arena que indica proceso de ejecución, el cual no parece acabar nunca); la solución es desplazar este cursor "reloj de arena" sobre el icono MS-DOS que representa la sesión AS11 abierta en Windows95 y pinchar sobre él: inmediatamente se observa cómo el proceso de compilación se completa sin problemas. Para resolver este pequeño inconveniente basta editar ENS.PIF y modificar alguna de las opciones relacionadas con la operación multitarea de Windows95, como las de ejecución exclusiva, prioridad, o MISCELANEA-> FONDO-> SIEMPRE _SUSPENDIDO (no poner 'X' en el recuadro).

Problemas diversos.

- Se ha observado que en ocasiones si se ejecuta y abandona Visual11 repetidas veces desde una misma sesión Windows, eventualmente al ejecutarlo por n-ésima vez presenta el error **"Disk or network error"**, y nos devuelve al sistema operativo. La solución a este problema es sencilla: **salir de Windows y volver a entrar** (obsérvese que esta situación es muy común en muchas aplicaciones comerciales que operan en el entorno MS-Windows).
- Al ordenar desde Visual11 una impresión, **FICHERO-> IMPRIMIR**, dependiendo del *driver* concreto de impresora instalado en MS-Windows, podría ocurrir que aunque Visual11 presente el mensaje "Datos enviados a la impresora", aparentemente la impresión no se ejecuta. Se ha observado que en dichos casos de aparente conflicto con el *driver*, la impresión solicitada se lleva a efecto sin problemas nada más abandonar la sesión actual de Visual11 (es decir, al salir de Visual11 automáticamente la impresora se pone en marcha e imprime todo lo solicitado durante la sesión de trabajo).
- La **TERMINAL** de comunicaciones, que no hemos empleado en prácticas todavía, no se encuentra tan depurada como el resto del entorno de Visual11. Si en una sesión se observa la pérdida de comunicación con la tarjeta exterior se sugiere pulsar su botón de reset (al presionar el botón de reset de una tarjeta basada en BUFFALO no se pierde información de ningún tipo), o, si es necesario, se debe salir de TERMINAL y volver a entrar.

8.2 Trabajo futuro.

El objetivo fundamental de desarrollar Visual11 era proporcionar al alumno una herramienta de iniciación a los microcontroladores sencilla de uso, que pudiese emplear libremente en una sala de usuarios o en casa. Aunque Visual11 en su versión 0.95 cumple ya estos requisitos, estamos trabajando ya en la incorporación de diversas mejoras, como pueda ser el desarrollo de un compilador de ensamblador específico para Visual11 (para no depender del que Motorola gentilmente proporciona como dominio público), o completar la terminal de comunicaciones con una tarjeta basada en BUFFALO.

Otro de los objetivos a corto-medio plazo sería distribuir el simulador entre la comunidad iberoamericana, cosa que no hemos podido hacer hasta ahora debido a la "juventud" de Visual11. Lo ideal en este sentido sería la realización de un texto de aprendizaje de microcontroladores basado en el 68HC11 que se apoye en Visual11, aspecto en el que ya estamos trabajando. Recordemos que el 68HC11 es uno de los microcontroladores más empleado en el mundo en entornos educativos, y que en español apenas cuenta con referencias.

El lector interesado puede permanecer atento a las direcciones internet http://zape.unizar.es, y http://www.unizar.es, donde expondremos futuras novedades y noticias en relación a este simulador.

9. Referencias.

[Bursky 95] Bursky, D. *Here an MCU, there an MCU.* Electronic Design, Oct. 13, 1995.

[Driscoll 94] Driscoll, F.F., Coughlin, R.F., Villanucci, R.S. *Data Adquisition and Process Control with the M68HC11 Microcontroler.* Prentice-Hall, 1994.

[Huang 96] Huang, H.W. *MC68HC11. An Introduction (Software and Hardware Interfacing).* West Publishing Company, 1996.

[Lipovski 88] Lipovski, G.J. *Single and Multiple-chip Microcomputer Interfacing.* Prentice-Hall, 1988.

[Malone 96] Malone, M. *The Microprocessor. A Biography*. Springer-Verlag, 1996.

[Martín del Brío 96] B. Martín del Brío, A. Bono, A. Ciriano, J.M. López. *Simulador didáctico para los primeros pasos en microprocesadores*. II Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica TAEE'96, Sevilla 1996.

[Martín del Brío 97a] B. Martín del Brío. *Curso de Microprocesadores y Microcontroladores (6800 y 68HC11).* Universidad de Zaragoza, 1997.

[Martín del Brío 97b] B. Martín del Brío. *Kit de Iniciación al microcontrolador 68HC11.* Dept. Ingeniería Electrónica y Comunicaciones, Universidad de Zaragoza. Revisión nº 2, noviembre 1997.

[Martín del Brío 97c] B. Martín del Brío, A. Bono, A. Ciriano, J.M. López. *Consideraciones sobre la integración de simulación y montaje en prácticas de microprocesadores.* Revista de Enseñanza y Tecnología, 7, 46-55, 1997.

[Miller 93] Miller, G.H. *Microcomputer Engineering*. Prentice-Hall, 1993.

[Motorola 88] MC68HC11 *EEPROM programming from a personal computer.* Application Note AN1010, Motorola, 1988.

[Motorola 91] *M68HC11 Reference Manual.* Motorola, 1991. (disponible en formato pdf en http://www.motorola.com).

[Motorola 91b] *MC68HC11E9 Techanical Data.* Motorola, 1991.

[Motorola 92] *M68HC11EVBU Universal Evaluation Board User´s Manual*. Motorola, 1992.

[Motorola 94] *PCBUG11 User´s Manual.* Motorola, 1994.

[Motorola 95] *M68HC05 Applications Guide.* Motorola, 1995.

[Sibigtroth 95] Sibigtroth, J.M. *Understanding Small Microcontrollers (M68HC705J1A).* Motorola/Prentice-Hall, 1995.

[Sosa 95] J. M. Sosa Navarro. *Teoría y Práctica del Microcontrolador MC68HC11E9.* E.U.I.T.T. Universidad de Las Palmas de Gran Canaria, 1995.

[Spasov 96] Spasov, P. *Microcontroler Technology: the 68HC11.* 2ed, Prentice-Hall, 1996.

[TAEE 94] *Actas del Primer Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica TAEE94.* Universidad Politécnica de Madrid, Junio 1994.

[TAEE 96] *Actas del 2º Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica TAEE94.* Universidad de Sevilla, Septiembre 1996.

[Varela 94] Varela, F.J., Gómez, J.M., Castro, M. *Programa Micro. Aprendizaje de Microprocesadores con PC.* Editorial Marcombo, 1994.

Apéndices.

A. Introducción al M68HC11 de Motorola.

El M68HC11 de Motorola no es tan solo un potente microcontrolador (y uno de los más conocidos), sino que también resulta una excelente herramienta educativa que permite enseñar una enorme variedad de conceptos gracias a los numerosos periféricos que integra. Por otro lado, resulta extremadamente sencillo y barato construir un sistema mínimo basado en un 68HC11, pudiendo grabarse el software de aplicación en su EEPROM interna desde un PC convencional sin necesidad de hardware adicional. Como su precio es relativamente asequible (teniendo en cuenta la gran cantidad de periféricos que integra), prácticamente cualquier alumno puede montarse su placa (un pequeño y barato sistema de desarrollo), y experimentar por su cuenta.

A.1 La familia Motorola 68xx.

El 68HC11 pertenece a la gran familia Motorola 68xx, que domina con claridad el mercado de microcontroladores (abarca cerca del 20% del mercado mundial de microcontroladores, alcanzando el 30% en el caso del de 8 bits). Dentro de la gran familia 68xx, las "subfamilias" más conocidas son la 6805 y 68HC11, la primera de gama baja-media, la segunda de gama media-alta. Otras familias son la 6808 (recientemente introducida como mejora de la 6805), la 68HC12 (de 16 bits, pensada para mejorar las prestaciones de la 68HC11), y la 68HC16 (de 16 bits, y con características DSP).

En particular, la **familia 6805**, en sus diferentes versiones, es el microcontrolador más vendido en el mundo. El 6805 cuenta con limitadas prestaciones y precio reducido (con ejemplares que cuestan menos de 300 pesetas), incluyendo numerosas versiones (más de cien) y posibilidad de diseño a medida para grandes series (en realidad, muchas de las versiones del 6805 son diseños que grandes compañías solicitan a Motorola para alguna aplicación particular, y que Motorola más adelante incluye en su catálogo).

A modo de ilustración, podemos señalar que el modelo MC68HC05K posee 16 pines en formato DIP, con de 2 a 4 KB de ROM, 176 bytes de RAM, 8 líneas de E/S y un temporizador/contador. Si se adquiere en grandes cantidades, su precio es de tan solo 0.9 dólares (unas 100 pesetas).

A.2 EL MC68HC11.

La familia **M68HC11** está compuesta por microcontroladores de 8 bits de gama media y alta. El 68HC11 se introdujo en el año 1985 como actualización de un microcontrolador más antiguo, el 6801, con una clara orientación inicial hacia las aplicaciones de automoción (encendido electrónico,

control electrónico del motor, frenado ABS, etc.); grandes compañías como General Motors, Chrysler, Volkswagen o Toyota lo vienen empleando desde hace años. No obstante, se viene utilizando de manera masiva en diversos campos, como aplicaciones de consumo (p.e., las cámaras fotográficas Canon EOS), telecomunicaciones (p.e., teléfonos móviles de Motorola), e industriales (detectores de incendios, instrumentación biomédica, etc.). En definitiva, el 68HC11 es uno de los microcontroladorees más populares en todo el mundo (en 1996 se llevaban vendidos ya 400 millones de ejemplares), y uno de los más ampliamente utilizados en la enseñanza gracias a sus características pedagógicas.

El 68HC11 se construyó entorno a una CPU basada en la arquitectura del clásico microprocesador 6800, pero reforzada con registros adicionales y nuevas instrucciones. El resultado es un núcleo procesador capaz de ejecutar las mismas instrucciones que el 6800 (algunas en menos ciclos de reloj), además de otras nuevas, como las de producto y división, manejo de bits individuales o instrucciones para el trabajo en doble precisión. Por otra parte, implementa los mismos modos de direccionamiento que el M6800.

Las siglas HC en la denominación de los miembros de esta familia hace referencia al proceso HCMOS (*High* CMOS) empleado en su confección, lo que conlleva dispositivos de reducido consumo, alta inmunidad al ruido y elevada velocidad de operación. Una versión típica de este microcontrolador, como el MC68HC11A8, incluye en un dado de silicio de 6.5x7.4 mm más de 100.000 transistores.

La frecuencia típica de trabajo de estos microcontroladores es de 2 MHz, requiriendo para ello un cristal de cuarzo de 8 MHz (dividen la frecuencia por cuatro); el resto del circuito de reloj está integrado. Algunas versiones operan también a 3 y 4 MHz. Estas frecuencias de reloj no deben llevar a engaño: aunque microcontroladores de otras casas operan a frecuencias aparentemente más altas, los de Motorola ejecutan un ciclo de bus en un único ciclo de reloj, mientras que otros (como los de Intel) precisan tres, cuatro o más, de modo que deben operar a frecuencias unas cuatro veces superiores para obtener un rendimiento equiparable.

Por otra parte, el diseño del 68HC11 es completamente estático, de modo que no necesita refrescar los valores de sus registros internos, no presentando por ello límite inferior en su frecuencia de reloj. De esta manera, en los diseños cuyo requisito fundamental sea el bajo consumo puede reducirse su frecuencia de operación todo lo que sea preciso (recordemos que en los dispositivos CMOS el consumo se incrementa con la frecuencia de operación).

Su bus de direcciones es de 16 líneas, de manera que puede direccionar hasta 64 KBytes de memoria, desde la dirección $0000 a la $FFFF [4] (no obstante, algunos de los miembros de la familia 68HC11 pueden direccionar más de 1 MB). Por otro lado, aunque su bus de datos es de 8 bits, está pensado para trabajar fácilmente con datos de 16.

El núcleo procesador del 68HC11 incluye el siguiente conjunto de registros (modelo para el programador):

a) **Acumulador D**, de 16 bits, compuesto por dos **acumuladores A y B** de 8 bits concatenados. El programador en todo momento puede trabajar con un único registro de 16 bits (D), o con dos de 8 (A y B), a su elección.

b) Registros **índices X e Y**, de 16 bits, empleados en el direccionamiento indexado (como índices de tablas de datos, vectores, etc.).

[4] En la notación de Motorola los números hexadecimales (base 16) se indican mediante el símbolo $ (por ejemplo, $3F6A) y los binarios con % (p.e., %10101100). Si no se incluye símbolo alguno el número se supondrá base 10.

c) Contador de **programa PC** (*Program Counter*) de 16 bits, apunta a la dirección donde se encuentra la siguiente instrucción que se va a ejecutar.

d) **Puntero de pila SP** (*Stack Pointer*), de 16 bits, apunta a la pila del sistema.

e) Registro de **códigos de condición o CCR** (*Condition Code Register*). Es el registro de estado, que incluye 8 señalizadores o *flags*: acarreo (C), desbordamiento (V), resultado cero (Z), resultado negativo (N), máscara de las interrupciones IRQ (I), acarreo a la mitad (H), máscara de las interrupciones XIRQ (X) y habilitación del modo STOP de bajo consumo (S).

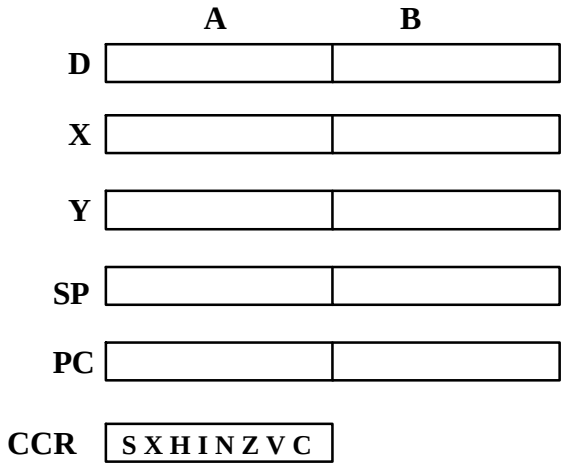


Figura 22. *Modelo del programador del MC68HC11.*

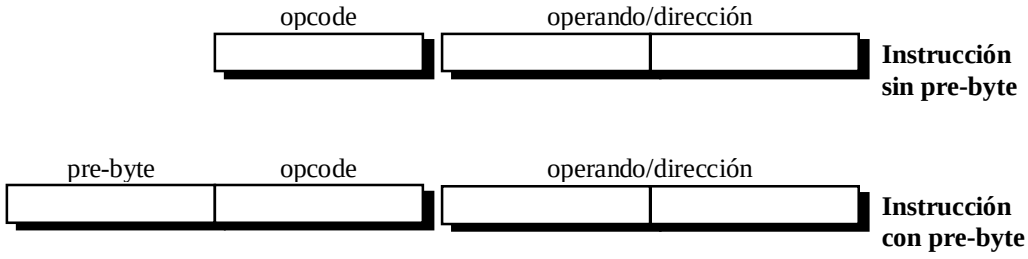


Figura 23 *Tamaño y composición de las instrucciones en el 68HC11).*

A.3 Modos de direccionamiento e instrucciones.

Una de las características más importantes de un microprocesador es su **juego de instrucciones**, es decir, el conjunto de instrucciones que puede ejecutar. Cada instrucción de un microprocesador se compone de: a) un **código de operación u *opcode*** (*operation code*), número binario que define la instrucción a ejecutar (de 8 o 16 bits en el caso del 68HC11); b) los **operandos** con los que la instrucción debe trabajar, o bien la dirección donde éstos se encuentran. Las distintas maneras que pueden contemplarse a la hora de obtener los operandos se conocen

como **modos de direccionamiento** (si vienen justo a continuación del opcode, si están en cierta casilla de memoria, en determinado registro de la CPU, etc.).

El juego de instrucciones del M68HC11 incluye **307 opcodes**. Debido a que mediante 8 bits solamente pueden codificarse hasta 256 diferentes, en el HC11 algunos códigos de operación van precedidos de un grupo adicional de 8 bits que Motorola denomina **pre-byte**. Así, en el 68HC11 algunos códigos de operación están compuesto por un único byte y otros por dos. Como cada instrucción puede venir completada por un dato de 8 o 16 bits, o una dirección de 8 o 16 bits, las *instrucciones del HC11 estarán formadas en total por un número de bytes comprendido entre 1 y 4*.

En el Apéndice C se incluye el juego completo de instrucciones del 68HC11; puede observarse que la mayor parte de ellas operan entre el dato guardado en uno de los registros de la CPU (A, B, D,...) y uno almacenado en memoria. Las instrucciones pueden clasificarse en los siguientes grupos:

- **Instrucciones de transferencia de información.** Carga de un dato de memoria en un registro LD (*Load*): LDAA (carga en el acumulador A), LDAB (carga en B), LDD (carga en D),...; almacenamiento en memoria del dato guardado en un registro ST (*Store*): STAA (guarda A en una casilla de memoria), STAB, STD,...; intercambio de datos entre registros: TAB, XGDX, etc.
- **Instrucciones aritméticas.** Incremento (INC): INCA, INCB,...; decremento (DEC): DECA, DECB, ...; suma (ADD=*addition*): ADDA, ADDB, ADDD, ADCA, ...; resta (SUB=*subtraction*): SUBA, SUBB, ...; producto y division: MUL, IDIV, FDIV.
- **Instrucciones lógicas.** Operaciones lógicas 'and', 'or', etc.: ANDA, ANDB, ...; operaciones de desplazamientos y rotación: ROLA, ROLB, ASLA, etc.
- **Instrucciones de comparación y test.** Compara un número con otro (CMP): CMPA, CMPB, CMP, ...; compara con cero (TST): TSTA, TSTB, TST, ...
- **Instrucciones de salto.** Salto incondicional absoluto JMP (*jump*) y relativo BRA (*branch*); saltos condicionales: BCC (*branch if carry clear*), BPL (*branch if plus*), BGT (*branch if greater or equal*), etc.
- **Instrucciones de manejo de pila y subprogramas.** Poner dato en la pila (PSH=*push*): PSHA, PSHB, ...; tomar dato de la pila (PUL=*pull*): PULA, PULB, ...; llamada a subrutina: JSR, BSR; retorno de subrutina RTS, y de rutina de interrupción RTI.
- **Otras.** Acceso a bits específicos: CLC (*clear carry*), BSET (activación de bit), etc.; control del hardware: interrupción software SWI, espera interrupción WAI; no opera: NOP, BRN.

Finalmente, el M68HC11 contempla los siguientes siete **modos de direccionamiento**:

a) Modo extendido. En este caso se proporciona explícitamente tras el opcode una dirección de 16 bits. Así, en memoria aparecen justo a continuación del opcode dos bytes adicionales que indican la dirección donde se encuentra el dato sobre el que hay que operar, o bien la dirección a la que hay que saltar. Por ejemplo, la instrucción de carga en el acumulador A en modo extendido, LDAA $1032, toma el dato contenido en la dirección de memoria $1032, y lo lleva a A, A←($1032). Se emplea para trabajar con variables de programa.

b) Modo directo. En los microprocesadores de 8 bits de Motorola este modo es un caso particular del extendido, con la restricción de que solamente se indica una dirección de 8 bits, la cual señalará una de las primeras 256 casillas de memoria (de la $00 a la $FF). Por ejemplo, la instrucción de carga en el acumulador A en modo directo, LDAA $18, toma el dato contenido en la dirección de memoria $0018, y lo lleva a A, A←($0018).

c) Modo indexado. Es el modo empleado en el tratamiento de tablas y matrices de datos. Para el cálculo de la dirección efectiva la CPU se apoya en un registro índice (el X ó el Y), que almacena una dirección de memoria de 16 bits, cantidad a la que se suma un desplazamiento (*offset*) de 1 byte (considerado sin signo), el cual se aloja a continuación del opcode. Por ejemplo, la instrucción de carga en el acumulador A en modo indexado LDAA $20,X , interpreta el dato contenido en el registro X como una dirección de memoria, a la cual le suma el desplazamiento $20, construyendo así una dirección efectiva (por ejemplo, si X contenía el valor $1012, se tendrá la dirección efectiva $1032); por último, toma el dato colocado en ésta dirección y lo lleva a A: A←((X)+$20).

d) Modo inmediato. Consiste en colocar el propio dato con el que se trabajará justo a continuación del opcode, es decir, en memoria aparece el opcode seguido por el propio dato (un número de 8 ó 16 bits). Este modo se emplea para almacenar constantes del programa. Por ejemplo, la instrucción de carga en el acumulador A en modo inmediato LDAA #$53, simplemente toma el dato $53 y lo lleva a A, A←$53.

e) Modo inherente o implícito. En este caso la dirección de los datos va implícita en el código de operación, por lo que no es preciso proporcionar ninguna dirección o dato adicional. Por ejemplo, la instrucción CLC (*CLear Carry*) pone a cero el bit indicador de acarreo C, C←0.

f) Modo acumulador. Es el empleado por aquellas instrucciones que pueden operar con el acumulador A o con el B, indicando simplemente con cual de los dos se trabaja. Por ejemplo, INCA e INCB (*INCrement A* e *INCrement B*), instrucciones que incrementan el registro A o el B, respectivamente, A←00, B←00.

g) Modo relativo. Empleado en instrucciones de salto, a continuación del opcode aparece un byte adicional que representa un desplazamiento (*offset*) relativo al actual valor del contador de programa PC; el desplazamiento se interpreta en este caso en complemento a dos (de -128 a +127). La dirección efectiva de salto se calcula sumando al actual valor del PC el desplazamiento indicado, de modo que pueden ejecutarse saltos entre (PC)-128 y (PC)+127. Por ejemplo, la instrucción de salto relativo BRA $03 (*BRAnch*), salta tres casillas más allá de la posición actualmente indicada por el PC, PC←(PC)+$03. La utilización de saltos relativos permite la confección de rutinas reubicables.

A.4 Aspectos hardware del 68HC11.

Bloques internos del 68HC11.

Como ya se ha expuesto, un microcontrolador integra en una única pastilla de silicio el núcleo procesador (CPU) y diversa cantidad de memoria y periféricos, lo que permite realizar un sistema completo de una manera sencilla, barata, fiable y ocupando poco espacio. Las distintas versiones que comprende la familia 68HC11 (más de 60) se diferencian en la memoria y periféricos que

integran, todas poseen el mismo núcleo procesador CPU-11, y todas se asientan sobre las mismas bases.

Los bloques internos presentes en el MC68HC11E9, uno de los más comunes (y empleado tanto en la tarjeta de evaluación EVBU como en el simulador Visual11), ilustran perfectamente los que podemos encontrar en las demás de las versiones del HC11 (Fig. 24).

- **CPU** (núcleo microprocesador). Arquitectura basada en el 6800, mejorada y con un repertorio ampliado a 307 opcodes, con fácil manejo de números de 16 bits.
- **ROM/EPROM**. 12 KB para almacenamiento de programa (parte alta de la memoria). Las versiones que integran ROM se denominan 68HC11, y las que integran EPROM añaden un '7' a su nombre, 68HC711; por ejemplo, se tiene la versión MC68HC11E9 con 12K ROM, y la MC68HC711E9, idéntico a ella pero con 12K EPROM.
- **RAM**. 512 bytes (por defecto en la parte baja de la memoria, aunque relocalizable). Se emplea para albergar la pila del sistema y datos (variables).
- **EEPROM**. 512 bytes (desde la dirección $B600). Puede emplearse como memoria de programa (para programas pequeños), para guardar datos de calibración, datos a mantener en caso de pérdida de alimentación, etc.
- **Oscilador interno**. El chip necesita externamente tan solo un cristal (o un resonador cerámico, más barato), y dos condensadores.
- **SCI** (*Serial Communications Interface*). Módulo de comunicación serie asíncrona o UART, para comunicación con dispositivos remotos (como un ordenador PC).
- **SPI** (*Serial Peripheral Interface*). Canal de comunicaciones serie síncrono, para comunicación serie de alta velocidad con dispositivos cercanos.
- **Puertos de entrada/salida.** El 68HC11E9 posee cinco puertos: A (con entradas y salidas), B (de salida), C y D (programables de entrada y de salida), y E (de entrada).
- **Sistema programable de temporización**. Permite la medida de intervalos temporales y frecuencias (por ejemplo, para medir r.p.m.), y el control de salidas temporizadas (por ejemplo, para generar PWM).
- **Acumulador de pulsos**. Empleado en el contaje de eventos externos.
- **RTI** (*Real Time Clock*). Reloj en tiempo real que permite ejecutar interrupciones periódicas, cada cierto tiempo.
- **Sistema de conversión A/D**. Incluye un conversor A/D de 8 bits, conectado a 8 líneas de entrada por medio de un multiplexor analógico.
- **COP** (*Computer Operating Properly*). Es un *watch-dog timmer*, o dispositivo de vigilancia. El sistema de seguridad del 68HC11 incluye además un monitor de funcionamiento correcto del reloj y de ejecución de instrucciones ilegales.
- **Bloque de registros de configuración** (por defecto, de las direcciones $1000 a $103F, aunque relocalizables). Se trata de 64 registros de 8 bits que se emplean para programar o configurar las diversas opciones del hardware, puertos, etc.

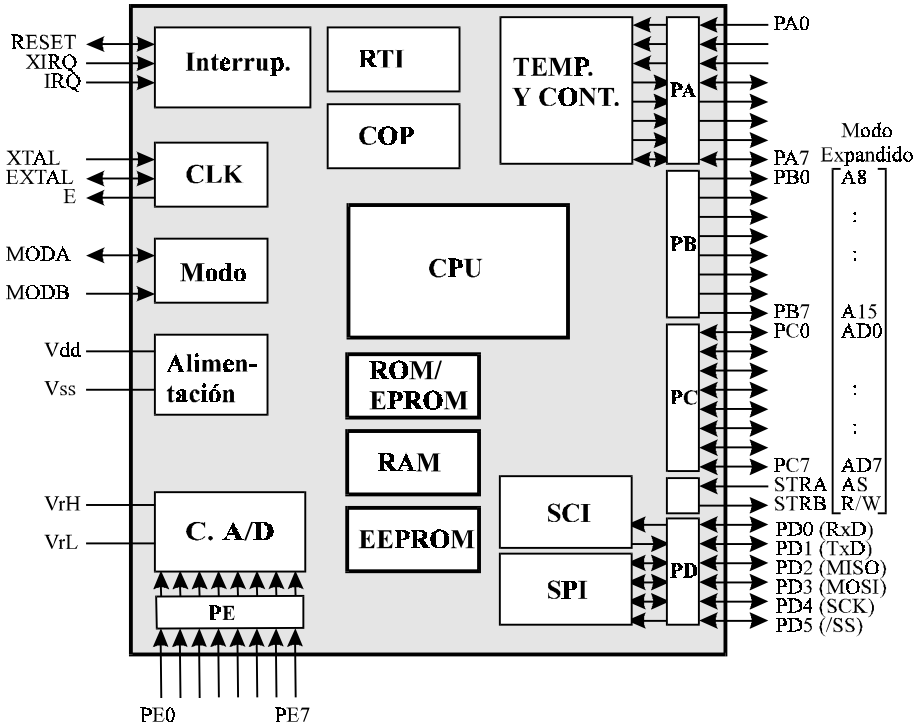


Figura 24. *Bloques internos del 68HC11E9. Se muestra el papel de cada uno de sus patillas, en modo normal y expandido (líneas entre corchetes).*

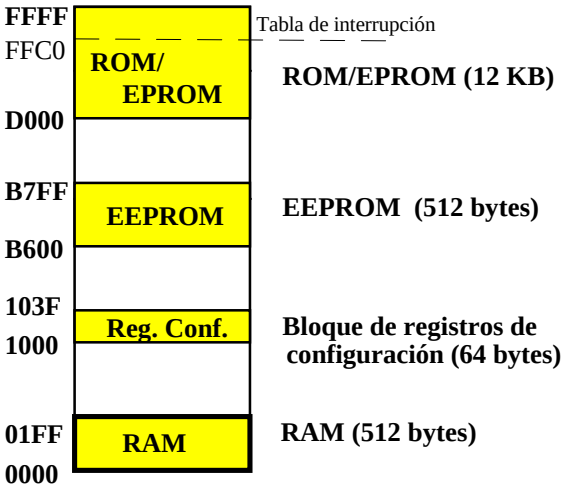


Figura 25. *Mapa de memoria del MC68HC11E9 (en modos normales de operación).*

Es importante tener en cuenta que las líneas de los distintos puertos del 68HC11, además de ser líneas de propósito general de entrada, salida o ambas, permiten también el acceso a los módulos internos descritos. Por ejemplo, el puerto E, de entrada, puede utilizarse como puerto de entrada de señales digitales, o para enviar señales analógicas procedentes de sensores al bloque de conversión A/D interno.

MODB	MODA	Modo seleccionado
1	0	Sencillo (*single-chip*)
1	1	Expandido
0	1	Especial: test
0	0	Especial: arranque externo (*bootstrap*)

Tabla 1 *Modos de operación del MC68HC11.*

Modos de operación.

Los microcontroladores de la familia 68HC11 pueden funcionar en **cuatro modos de operación** diferentes, dos normales y dos especiales: sencillo (*single-chip*), expandido (*expanded*), especial de test (*special test*) y especial de arranque externo (*bootstrap*). Estos modos se establecen mediante dos pines externos, MODA y MODB: tras la señal de reset el 68HC11 comprueba el estado de estos pines, y establece su modo de operación según los valores presentes en ellos (ver Tabla 1); este estado no es modificable posteriormente por software.

En **modo sencillo**, el 68HC11 trabaja exclusivamente con los recursos integrados en su propia pastilla (memoria interna, puertos, conversor A/D, etc.). Es el modo preferido por cuestiones de tamaño, consumo, precio y fiabilidad. En este caso, solamente deben incluirse externamente una serie de componentes electrónicos en torno al chip que aseguran su correcta operación, como son un cristal de cuarzo o resonador cerámico, un circuito de reset, y un conjunto de resistencias y condensadores (para polarizar o terminar ciertas patillas, filtrar la alimentación, proporcionar los niveles de referencia del conversor A/D, y establecer el modo de operación), como se muestran en la Fig. 26.

El **modo expandido** de operación está pensado para aquellas aplicaciones en las que el microcontrolador requiere más memoria de la que integra. Como en este caso se precisa el acceso desde el exterior a los buses internos, en modo expandido el HC11 sus puertos B y C, se transforman en las líneas de los buses: el puerto B se convierte en las líneas altas del bus de direcciones (A8 a A15), mientras que el C contiene las líneas bajas multiplexadas con las líneas de datos (AD0 a AD8). La línea AS (*Address Strobe*) indica con un '1' que en ellas hay presente una dirección, y con un '0' cuando se trata de un dato; para demultiplexar los buses debe incluirse un registro de retención como el 74HC373, que mantenga el valor bajo del bus de direcciones antes que aparezca por las mismas líneas el dato de 8 bits del bus de datos.

En modo expandido el microcontrolador se comporta esencialmente como un microprocesador, con la particularidad de que incluye parte del sistema en su interior. Los buses obtenidos son totalmente compatibles con los del M6800 de modo que comparten todos los periféricos, por lo que se habla de la familia 68xx de microprocesadores, microcomputadores y periféricos. Se debe tener en cuenta además que el sistema está diseñado de manera tal que el multiplexado de direcciones y datos no reduce la velocidad de su operación respecto del modo simple (pues se emplea un solo ciclo de reloj tanto en el acceso a memoria interna como externa).

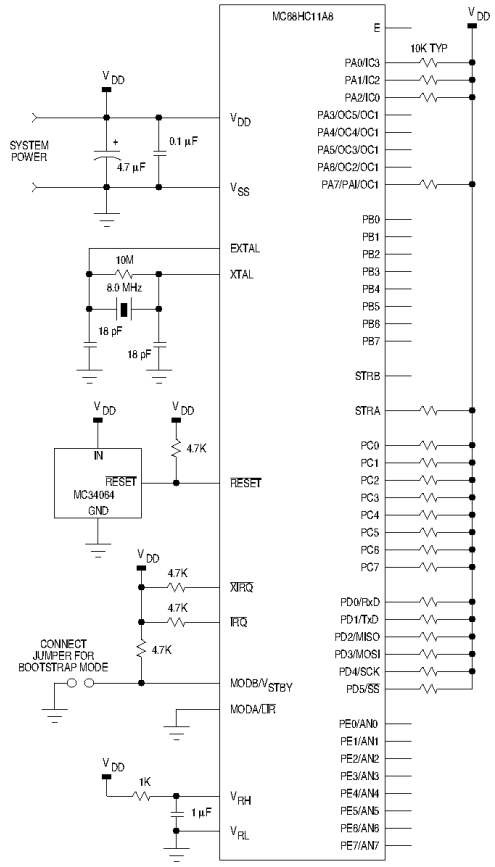


Figura 26 *MC68HC11A8 configurado en modo simple [Motorola 91].*

Finalmente, el **modo especial *bootstrap*** está pensado para permitir la carga desde el exterior (usualmente un PC) de un programa ensamblador, para programar la memoria integrada, etc.; resulta tremendamente útil para el desarrollo de prototipos. El **modo especial de test** lo emplea Motorola para la comprobación del integrado.

Patillaje del 68HC11.

Como se ha comentado, el 68HC11 posee numerosas versiones y tipos de encapsulado, por lo que las patillas presentes varían de un tipo a otro, no obstante, existe un conjunto de pines más o menos comunes o habituales en todos ellos. En concreto describiremos las características del 68HC11E9 (Fig. 27), por las razones ya expuestas. Debe tenerse en cuenta también que el papel de algunas de las patillas varía según opere en modo simple o expandido.

- **VDD** y **VSS**. Terminales de alimentación (+5 v) y tierra (0 v).
- **XTAL** y **EXTAL**. Conectores del cristal de reloj (8 MHz).
- **MODA** y **MODB**. Modo actual de operación del µC (entradas).
- **RESET**. Línea de reset (entrada).
- **IRQ** y **XIRQ**. Entradas de interrupción.
- **VrH** y **VrL**. Tensiones de referencia alta (*High*, VrH) y baja (*Low*, VrL) para el conversor A/D interno.

- **E**. Reloj E (salida); es el reloj del sistema.
- **Puerto A (PA0-PA7)**. Puerto con conexión al sistema de temporizadores y contadores, con líneas de entrada, salida y programables.
- **Puerto D (PD0-PD6)**. Puerto programables de entrada/salida, conectado a las líneas de los canales serie asíncrono y síncrono (SCI y SPI).
- **Puerto E (PE0-PE8)**. Puerto de entradas tanto digitales como analógicas, para el acceso al conversor A/D.
- **Puerto B (PB0-PB7) [A8-A15]**. Puerto de salida. En modo expandido las líneas del puerto B se convierten en las líneas altas del bus de direcciones, A8 a A15.
- **Puerto C (PC0-PC7) [AD0-AD7]**. Puerto programable como entrada o salida. En modo expandido las líneas del puerto C se convierten en las líneas bajas del bus de direcciones y bus de datos multiplexadas, AD0 a AD7.
- **STRA[AS]** y **STRB[R/W]** (*strobe* A y *strobe* B). Líneas de entrada y salida, respectivamente, empleadas en el diálogo con periféricos externos. En modo expandido STRA se convierte en la salida AS (*address strobe*) (AS='1' indica dirección presente en las líneas de datos y direcciones multiplexadas, y AS='0' indica dato); por su parte, STRB se convierte en la línea de lectura/escritura R/W.

Tanto el MC68HC11E9 como el MC68HC711E9 se comercializan en encapsulado PLCC-52, con 12K de ROM y EPROM. La versión con EPROM, MC68HC711E9, se comercializa para la realización de prototipos con ventana, y para pequeñas series sin ventana; en este último caso la EPROM del microcontrolador no puede borrarse, por lo que se suele denominar "programable una sola vez" u **OTP** (*One Time Programmable*). También puede adquirirse un MC68HC11E9 con el programa monitor BUFFALO integrado en su ROM interna (al precio de menos de 10 dólares).

Por otro lado, se comercializan variaciones de este E9. Por ejemplo, la versión E1 es un E9 con la ROM deshabilitada, y la E0 es un E9 con ROM y EEPROM deshabilitadas. La versión MC68HC11E2 es ideal para el desarrollo de prototipos, puesto que integra 2KB de EEPROM, fácilmente programable desde un PC sin necesidad de sistema de desarrollo alguno; además se comercializa en versión DIP-48.

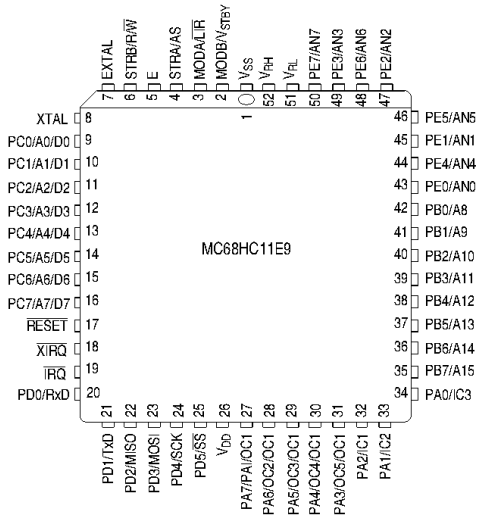


Figura 27. *Patillaje del MC68HC11E9.*

Sistema de interrupciones del 68HC11.

El 68HC11 posee un muy amplio sistema de interrupciones, algunas externas y otras internas. Fuentes externas de interrupción son la interrupción enmascarable IRQ (se puede enmascarar mediante el *flag* I), la no enmascarable XIRQ (similar a la NMI del 6800, aunque de comportamiento algo más sofisticado para evitar algunos problemas concretos, ¡y que en determinadas circunstancias puede enmascararse mediante el *flag* X!), y las asociadas a algunas funciones del sistema de temporización. La situación de RESET se suele incluir también en este grupo. Fuentes internas de interrupción son el COP, monitor de reloj, RTI, temporizadores, acumulador de pulsos, SCI y SPI, además de la interrupción software SWI.

El sistema de interrupciones es autovectorizado: cuando se recibe una solicitud de interrupción, si la CPU la acepta guarda todos los registros en la pila y acude a la denominada tabla de vectores de interrupción, donde cada fuente de interrupción tiene asignada dos casillas en las que se almacena la dirección donde se encuentra la rutina de servicio, que el microcontrolador carga en el PC, procediendo a su ejecución. La tabla de vectores está situada en la parte más alta de memoria (desde $FFFF hacia abajo); su organización se muestra en la Tabla 2.

Vector interrupción	Dispositivo	Máscara en el CCR	Máscara específica
FFC0 a FFD5	Direcciones reservadas	ninguno	ninguno
FFD6, FFD7	SCI: Registro receptor de datos lleno	I	RIE
	SCI: Receptor sobre-escrito	I	RIE
	SCI: Detección de línea inactiva	I	ILIE
	SCI: Registro transmisor de datos vacío	I	TIE
	SCI: Transmisión completa	I	TCIE
FFD8, FFD9	SPI: Transmisión serie completa	I	SPIE
FFDA, FFDB	Flanco de entrada en acumulador de pulsos	I	PAII
FFDC, FFDD	Acumulador de pulsos lleno	I	PAOVI
FFDE, FFDF	Temporizador desbordado	I	TOI
FFE0, FFE1	Temporizador OC5	I	OC5I
FFE2, FFE3	Temporizador OC4	I	OC4I
FFE4, FFE5	Temporizador OC3	I	OC3I
FFE6, FFE7	Temporizador OC2	I	OC2I
FFE8, FFE9	Temporizador OC1	I	OC1I
FFEA, FFEB	Temporizador IC3	I	IC3I
FFEC, FFED	Temporizador IC2	I	IC2I
FFEE, FFEF	Temporizador IC1	I	IC1I
FFF0, FFF1	Interrupción en tiempo real (RTI)	I	RTII
FFF2, FFF3	IRQ (pin externo o E/S paralelo)	I	ninguno
	IRQ: Puerto E/S paralelo	I	STAF
FFF4, FFF5	Pin XIRQ	X	ninguno
FFF6, FFF7	SWI	ninguno	ninguno
FFF8, FFF9	Opcode ilegal	ninguno	ninguno
FFFA, FFFB	COP desbordado	ninguno	NOCOP
FFFC, FFFD	Monitor de reloj	ninguno	CME
FFFE, FFFF	RESET	ninguno	ninguno

Tabla 2. *Tabla de vectores de interrupción del MC68HC11.*

¿Qué simula Visual11?

Recordemos que la presente versión de **Visual11 simula un MC68HC711E9 operando en modo simple**. De los bloques expuestos, Visual11 simula los siguiente:

- CPU-11
- Memoria RAM (512 bytes), EEPROM (512) y EPROM (12K).
- Puertos B (8 bits, salida) y C (8 bits, entrada/salida).
- Interrupción XIRQ (no enmascarable), IRQ (enmascarable) y SWI (software).
- El estado de RESET.

Además, Visual11 simula los siguientes periféricos básicos que pueden ser conectados a los puertos B y C: grupo de 8 diodos LED, 8 interruptores, dos visualizadores de 7 segmentos multiplexados, y un teclado hexadecimal no codificado.

B. Manual del Ensamblador AS11

En el Capítulo 3 se realizó una breve introducción al empleo del compilador AS11. A continuación transcribimos el manual abreviado que Motorola adjunta en forma de fichero ASCII junto con dicho compilador.

==========

The IBM PC 6800/01/04/05/09/11 cross assemblers (Motorola)

GENERAL

The assemblers are named **AS*.EXE**, where '*' is any of 0,1,h1,4,5,9 or 11 depending on which one you're using. Command line arguments specify the filenames to assemble.

The assemblers accept **options** from the command line to be included in the assembly. These options are the following:

l enable output listing.
nol disable output listing (default).
cre generate cross reference table.
s generate a symbol table.
c enable cycle count.
noc disable cycle count.

The command line looks like this :

as* file1 file2 ... [- option1 option2 ...]

If this method of passing commands to the assembler is used rather than the OPT pseudo op code, a space should separate the minus sign from the last file name and the first option. Example:

as5 program -l cre

This command assembles file 'program' with an output listing and a cross reference table.

The `S1' formatted object file is placed in file `filename.S19', the listing and error messages are written to the standard output. If multiple files are assembled, the 'S1' file will be placed under the first file's name.S19.

The listing file contains the address and bytes assembled for each line of input followed by the original input line (unchanged, but moved over to the right some). If an input line causes more than 6 bytes to be output (e.g. a long FCC directive), additional bytes (up to 64) are listed on succeding lines with no address preceding them.

Equates cause the value of the expression to replace the address field in the listing. Equates that have forward references cause Phasing Errors in Pass 2.

Expressions may consist of symbols, constants or the character '*' (denoting the current value of the program counter) joined together by one of the operators: +-*/%&|^. The operators are the same as in C:

+	add
-	subtract
*	multiply
/	divide
%	remainder after division
&	bitwise and
|	bitwise or
^	bitwise exclusive-or

Expressions are evaluated left to right and there is no provision for parenthesized expressions. Arithmetic is carried out in signed twos-complement integer precision (16 bits on the IBM PC)

Constants are constructed with the same syntax as the Motorola MDOS assembler:

'	followed by ASCII character
$	followed by hexadecimal constant
@	followed by octal constant
%	followed by binary constant
digit	decimal constant

ERRORS

Error diagnostics are placed in the listing file just before the line containing the error. Format of the error line is:

Line_number: Description of error

or

Line_number: Warning --- Description of error

Errors of the first type in pass one cause cancellation of pass two. Warnings do not cause cancellation of pass two but should cause you to wonder where they came from.

Error messages are meant to be self-explanatory.

If more than one file is being assembled, the file name precedes the error:

File_name,Line_number: Description of error

Finally, some errors are classed as fatal and cause an immediate termination of the assembly. Generally these errors occur when a temporary file cannot be created or is lost during the assembly. Consult your local guru if this happens.

DIFFERENCES

For indexed addressing, the comma is required before the register; `inc x' and `inc ,x' are not the same.

Macros are not supported. (try M4 or M6)

The force size operators ('>' and '<') are implemented for all assemblers.

The only **pseudo-ops** supported are:

ORG, FCC, FDB, FCB, EQU, RMB, BSZ, ZMB, FILL
PAGE and OPT.

The **OPT** pseudo-op allows the following operands:

nol	Turn off output listing
l	Turn on output listing (default)
noc	Disable cycle counts in listing (default)
c	Enable cycle counts in listing (clear total cycles)
contc	Re-enable cycle counts (don't clear total cycles)
cre	Enable printing of a cross reference table
s	generate a symbol table

Some of the more common pseudo-ops are not present:

SPC	Use blank lines instead
END	The assembly ends when there is no more input
TTL	use `pr' to get headings and page numbers
NAM[E]	Did you ever use this one anyway?

The above 4 pseudo-ops are recognized, but ignored.

ZMB (Zero Memory Bytes) is equivalent to BSZ (Block Store Zeroes).

FILL can be used to initialize memory to something other than zero:
FILL val,nbytes.

TARGET MACHINE SPECIFICS

(as0) 6800: Use for 6802 and 6808 too.

(as1) 6801: You could use this one for the 6800 and avoid
LSRD, ASLD, PULX, ABX, PSHX, MUL, SUBD, ADDD, LDD and STD.

(as4) 6804: The symbols 'a', 'x' and 'y' are predefined as
$FF, $80 and $81 respectively. Also defined as
'A', 'X' and 'Y'. Because of the 6804
architecture, this means that 'clr x' will work
since the x register is just a memory location.
To use short-direct addressing, the symbol
involved must not be a forward reference (i.e.
undefined) and must be in the range $80-$83.

Remember that bytes assembled in the range
$10-$7F will go into the data space; There is no
program space ROM for these locations.

The syntax for Register indirect addressing is as follows:
menmonic [<x>or<y>]
an example is:
lda [x]
the comma ',' is not allowed.

The MVI instruction (move immediate) has its own format :
mvi address,#data where address is
an 8-bit address in page zero, and data is the value to be written
to specified location.

(as5) 6805: There is no 'opt cmos' pseudo, so be careful not
to use STOP or WAIT in a program that is destined
for an NMOS version of the 6805. The MUL
instruction should also be avoided on all
versions of the 6805 except the C4. Cycle times

are for the NMOS versions.

(as9) 6809: The SETDP pseudo-op is not implemented.
Use the '>' and '<` operators to force the size of operands.
For compatibility, CPX is equal to CMPX.

(as11) 68HC11: Bit manipulation operands are separated by blanks instead of commas since the 'HC11 has bit manipulation instructions that operate on indexed addresses.

DETAILS

Symbol: A string of characters with a non-initial digit. The string of characters may be from the set:

[a-z][A-Z]_.[0-9]$

(. and _ count as non-digits). The `$' counts as a digit to avoid confusion with hexadecimal constants. All characters of a symbol are significant, with upper and lower case characters being distinct. The maximum number of characters in a symbol is currently set at 15.

The symbol table has room for at least 2000 symbols of length 8 characters or less.

Label: A symbol starting in the first column is a label and may optionally be ended with a ':'. A label may appear on a line by itself and is then interpreted as:

```
Label   EQU   *
```

Mnemonic: A symbol preceded by at least one whitespace character. Upper case characters in this field are converted to lower case before being checked as a legal mnemonic. Thus `nop', `NOP' and even `NoP' are recognized as the same mnemonic.

Note that register names that sometimes appear at the end of a mnemonic (e.g. nega or stu) must not be separated by any whitespace characters. Thus `clra' means clear accumulator A, but that `clr a' means clear memory location `a'.

Operand: Follows mnemonic, separated by at least one whitespace character. The contents of the operand field is interpreted by each instruction.

Whitespace: A blank or a tab

Comment: Any text after all operands for a given mnemonic have been processed or, a line beginning with '*' up to the end of line or, an empty line.

Continuations: If a line ends with a backslash (\) then the next line is fetched and added to the end of the first line. This continues until a line is seen which doesn't end in \ or until MAXBUF characters have been collected (MAXBUF >= 256).

FILES

filename.S19	S-record output file
STDOUT	listing and errors (use redirection for listing file)
Fwd_refs	Temporary file for forward references.

IMPLEMENTATION NOTES

This is a classic 2-pass assembler. Pass 1 establishes the symbol table and pass 2 generates the code.

12/11/84 E.J.Rupp

This version of the cross assemblers ported to the IBM PC **4/13/87**

C. Juego de Instrucciones del M68HC11

Adjuntamos el juego de instrucciones del M68HC11, tal y como aparece en las hojas de características de Motorola.


HC11