Réalisez votre ordinateur individuel

LE COMPILATEUR BASIC

C'est un article très chargé que nous vous proposons aujourd'hui, d'autant qu'aux « grands » thèmes annoncés en sous-titres vont s'ajouter un certain nombre d'informations que vous êtes nombreux à réclamer ; aussi allons-nous sans plus tarder passer aux choses sérieuses.

Programmateur de PROM bipolaires manuel

Nous avons parlé de ce montage le mois dernier à propos de la carte UVP 09 puisque celle-ci nécessite, pour son adressage, de pouvoir programmer deux PROM bipolaires du type 7611 que vous commencez à bien connaître. Le but du montage proposé ci-après est donc de permettre à ceux d'entre vous qui ne possèdent pas la carte de programmation que nous avons déjà décrite de programmer « à la main », mais avec un confort suffisant tout de même, des PROM 7611. Son application n'est pas limitée à la seule carte UVP 09 et vous pourrez ainsi, si vous le désirez, changer certains adressages du système à votre convenance.

Précisons tout de suite que la programmation manuelle de mémoires du type des 7611 qui sont des 256 mots de 4 bits, est tout à fait réalisable compte tenu de la méthode adoptée pour ce faire sur notre montage, mais aussi compte tenu du fait qu'il y a très peu d'informations à programmer dans ces 7611 utilisées en décodage d'adresse, comme vous pouvez le constater sur le tableau de leur contenu publié n° 1 702 page 176, figure 15. Rappelons que les mémoires 7611 vierges contiennent des F (c'est-à-dire leurs quatre bits à 1) à toutes les adresses ; la seule reprogrammation pénible est donc celle de la PROM DECVIS 09. Enfin, et nous pourrons ensuite passer au schéma, notez bien que ce montage revient aux environs de 60 à 80 F, ce qui est de l'ordre du prix de deux 7611 vierges...

Le schéma

Il vous est proposé figure 1 et nous allons le commenter ci-après, non sans avoir insisté au préalable sur le fait qu'il respecte les indications données par Harris pour la programmation de ses mémoires 7611. Rappelons aussi que si de très nombreuses PROM 256 mots de 4 bits sont compatibles en lecture, il n'en est pas de même en programmation. Ce montage fonctionne pour les PROM Harris, un point c'est tout !

Pour programmer une PROM Harris, il faut respecter la procédure suivante :

— Adresser la PROM à l'endroit à programmer.

— Mettre les pattes CE barre au niveau haut pour désélectionner la mémoire.

— Appliquer une tension de 5 V sur les sorties de façon à ne pas activer la circuiterie de programmation interne.

— Faire monter la tension d'alimentation de 5 V à 12 V.

— Après un délai supérieur à 10 µs minimum et 100 µs maximum, faire monter à 10,5 V la sortie à programmer à 0 pendant un temps compris entre 90 µs et 110 µs puis la ramener à 5 V.

— Programmer de la même façon les autres bits se trouvant à la même adresse si nécessaire.

— Pour vérifier la PROM, et après un temps minimum de 10 µs, ramener la tension d'alimentation à 5 V, mettre CE barre au niveau bas et lire l'état de la sortie.

— Si un bit ne s'est pas programmé, la procédure ci-avant peut être répétée jusqu'à concurrence de dix fois. Si au bout de ces dix répétitions la programmation n'a toujours pas eu lieu, la mémoire doit être considérée comme défectueuse.

Le respect de ces contraintes est assuré par notre montage, dont le seul point « faible » se situe au niveau du respect du temps de programmation. En effet, pour ne pas utiliser de composants de précision qui sont chers et surtout très rares, nous avons fait un montage avec de simples monostables TTL. Le temps de programmation n'est donc pas forcément dans la fourchette très étroite 90 à 110 µs mais cela n'a pas de conséquence fâcheuse, la tolérance des mémoires sur ce point étant suffisante. Si vous êtes perfectionniste et que vous disposiez d'un oscilloscope, vous pourrez toujours ajuster exactement ce temps.

Les interrupteurs S_0 à S_7 permettent d'adresser la mémoire en mettant à 0 les pattes nécessaires. Les lignes d'adresses sont ramenées au + 5 V par des résistances, car il est interdit de les laisser flottantes ou en l'air lors de la phase de programmation. Les quatre bits de la mémoire, 00, 01, 02 et 03, peuvent être reliés, via S_8 qui est un commutateur rotatif, à deux circuits : le circuit de programmation que nous allons voir et le circuit de lecture de l'état du bit utilisé en lecture et en vérification de la mémoire. Ce circuit est constitué de deux transistors montés en darlington qui commandent une LED ; la LED est allumée si le bit relié au circuit par S_8 est à 1, elle est éteinte dans le cas contraire.

Le support de la mémoire à programmer est normalement alimenté en + 5 V via la diode D_2, et il peut recevoir la haute tension de programmation via le transistor T_4 commandé lui-même par T_3. La diode D_2 évite la ré-injection de la tension de programmation sur le + 5 V, ce qui serait néfaste à bon nombre de composants.

Cette haute tension de programmation est obtenue par

régulation à 12,6 V d'une tension continue de 15 V environ, au moyen d'un classique régulateur 12 V dans la patte de masse duquel on a mis une diode. Cette diode D_1 compense les chutes de tension dans T_4 et permet d'appliquer au support de la PROM une haute tension aussi proche de 12 V que possible.

Voyons maintenant la partie génération des séquences de programmation. Un poussoir P déclenche un monostable 74121, via son entrée trigger de Schmitt. Cela permet de supprimer très simplement les rebonds du poussoir au moyen d'une simple cellule RC. Ce monostable génère une impulsion positive sur Q et une négative sur Q barre de durée tcyc (voir chronogramme de la fig. 2). L'impulsion positive dévalide la PROM et déclenche simultanément le monostable M_2 qui génère une impulsion positive de durée td qui n'est autre que le temps d'attente évoqué ci-avant (celui qui est compris entre 10 et 100 µs). La descente de cette impulsion déclenche alors M_3 qui génère l'impulsion de programmation proprement dite de durée tp (celle qui doit faire de 90 à 110 µs). Cette impulsion commande une porte NAND à collecteur ouvert qui, grâce aux résistances de 220 Ω et de 390 Ω, applique le niveau requis sur la sortie sélectionnée.

Le temps généré par M_1 est calculé de telle sorte qu'après l'impulsion de programmation, il s'écoule encore un temps supérieur ou égal à td permettant de ne vérifier le contenu de la PROM qu'après écoulement de celui-ci. La sortie Q barre de M_1, dont nous n'avons pas parlé, commande T_3 et fait appliquer la haute tension à la PROM pendant toute la phase de programmation.

Le résumé chronologique de ces opérations est indiqué figure 2 et, si vous voulez contrôler votre montage à l'oscilloscope, sachez que : td doit faire de 10 à 100 µs, tp de 90 à 110 µs et le td final de nouveau de 10 à 100 µs. Sachez aussi que M_2 régit td, M_3 régit tp et M_1 régit tcyc c'est-à-dire le td final puisque td final = tcyc − tp − td.

Le condensateur de 100 pF en sortie de la porte est indispensable et permet de compenser les temps de transferts dans les divers circuits, évitant ainsi que la mémoire ne soit sélectionnée en lecture (lorsque M_1 termine son temps) alors que la tension de programmation est toujours appliquée sur la patte de sortie.

La réalisation

Pour conserver son caractère économique à cette réalisation, nous ne lui avons pas prévu d'alimentation. Le + 5 V sera prélevé sur le système lui-même et la tension non régulée + V appliquée au régulateur est tout simplement celle que l'on trouve sur l'alimentation de l'ordinateur avant le régulateur du + 12 V.

Les composants utilisés sont très classiques ; il faut seulement veiller à ne pas acheter de 74L121 ou 74LS121 ni de 74L123 ou 74LS123 ; en effet, les constantes de temps ont été calculées pour des 74121 et 74123 normaux, d'ailleurs bien plus répandus que les versions L ou LS de ces mêmes circuits.

Les huit interrupteurs S_0 à S_7 sont constitués par huit interrupteurs en boîtier DIL ; choisissez des modèles maniables et de bonne qualité, car ils seront souvent actionnés.

Le commutateur rotatif S_8 est un 3 circuits 4 positions classique, dont les pattes sont coupées en pointes pour entrer dans les trous du CI (à moins que vous ne trouviez la version pour CI qui existe aussi).

Le dernier point de détail

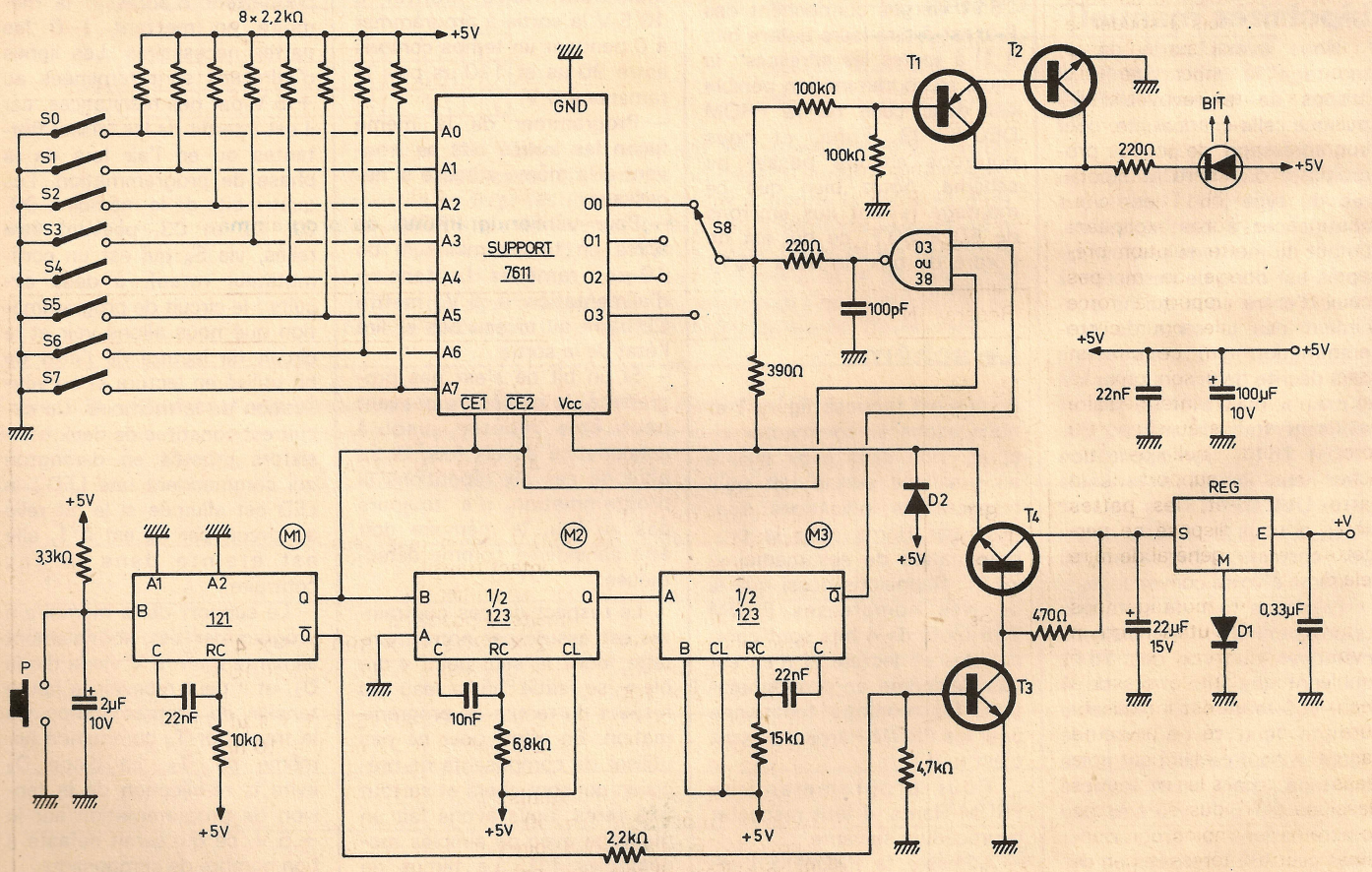

Fig. 1

important concerne les supports de circuits intégrés ; pour ce qui est du 74121, du 74123, du 7403 ou 38, vous pouvez en mettre ou non selon la confiance que vous avez en vos talents de soudeur. Pour ce qui est de la PROM, il est évident qu'il faut un support et trois solutions vous sont proposées :

– La solution « pauvre », à n'employer que si vous comptez utiliser ce montage très rarement, consiste à monter comme support de PROM un modèle normal mais de bonne qualité tout de même (contacts tulipes si possible).

– La solution fixe consiste à acheter un support à force d'insertion nulle (marque Textool si possible ou équivalent), qui sera soudé à l'emplacement prévu pour le support de PROM. Attention, ce support vaut aussi cher que tout le reste du montage ! Pour cela nous vous proposons la dernière solution...

– La solution mobile consiste à souder sur la carte un support normal ; puis à souder le support Textool sur un autre support et à enficher le tout l'un dans l'autre (revoyez si nécessaire notre article sur le programmateur de PROM publié dans le numéro d'octobre 1983, dans lequel des schémas et des photos expliquent tout cela). Cette solution présente l'avantage de ne pas mobiliser le support à force d'insertion nulle pour cette seule application, ce qui est assez logique vu son prix. Le but du support intermédiaire est de permettre au support à force d'insertion nulle de s'enficher dans le support de la carte ; en effet, les pattes plates dont il dispose ne permettent pas en général de faire cela directement.

La simplicité du montage nous a permis de n'utiliser qu'un « vulgaire » circuit imprimé simple face, dont le dessin à l'échelle 1 vous est indiqué figure 4. Son tracé ne présente pas de difficulté et il peut être réalisé par toutes les méthodes classiques. Si vous êtes assez nombreux (au moins une centaine) à être intéressés par ce circuit prêt à l'emploi, la société Facim le commercialisera. Pour cela, faites part de vos demandes éventuelles directement à cette société.

L'implantation des composants est présentée figure 5 et ne doit poser aucun problème d'ordre pratique, si ce n'est la mise en place du commutateur S_8, surtout si votre modèle comporte des cosses de connexion. Il faut alors tailler celles-ci en pointe (voire couper les cosses inutiles) pour pouvoir monter le commutateur sur le CI. Tous les composants se montent côté composants. L'emplacement du poussoir P est prévu pour un modèle digitast maintenant très répandu ; retouchez le dessin si nécessaire à ce niveau compte tenu de vos disponibilités.

Hormis le respect du sens des circuits intégrés et condensateurs chimiques, le montage ne requiert pas de précaution particulière. Le régulateur n'a pas besoin de radiateur.

Essais et utilisation

Après une ultime vérification du câblage, les essais pourront être faits, sans toutefois mettre de mémoire sur le support. Reliez le point +5 V de la carte via un interrupteur au +5 V du mini-ordinateur, et le point +V de la carte à l'entrée du régulateur +12 V de l'alimentation, via un interrupteur également. Ouvrez cet interrupteur et mettez votre système sous tension. Contrôlez au voltmètre que vous avez bien les tensions voulues sur le support de PROM au niveau des lignes d'adresses, compte tenu des positions de S_0 à S_7. Constatez que la LED est allumée quelle que soit la position de S_8. Fermez alors l'interrupteur du +V et vérifiez que la patte 16 du support est toujours à +5 V, sinon vous avez un problème au niveau de T_3, T_4 ou M1. Laissez votre voltmètre sur 16 et appuyez sur P, vous devez voir l'aiguille de celui-ci, faire un saut vers le haut. Cela correspond à l'impulsion de durée tcyc (fig. 2), qui est trop courte pour que le voltmètre ait le temps de se stabiliser. Si vous avez un oscilloscope, vous pouvez aussi vérifier les chronogrammes de la figure 2 et corriger les valeurs des résistances associées à M1, M2 et M3 si nécessaire ; une augmentation de résistance correspond à une augmentation de durée de l'impulsion.

Si ces vérifications (sauf celle à l'oscillo qui n'est à faire que si vous disposez d'un tel appareil) se sont avérées positives, vous pouvez passer à l'utilisation du montage selon le mode d'emploi suivant :

– Avant toute programma-

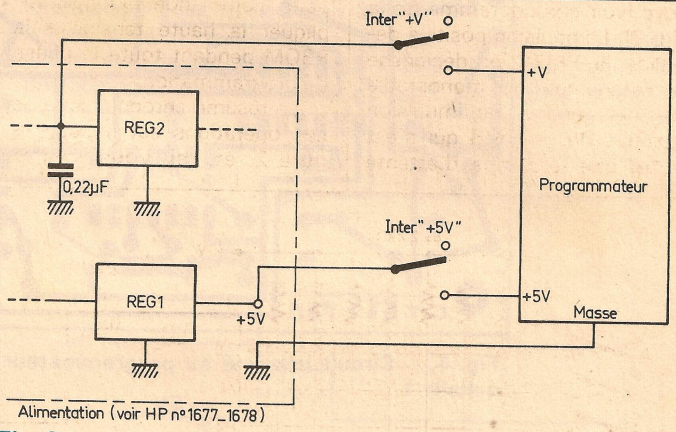

Fig. 2. – Chronogrammes du programmateur de PROM manuel.

Repère	Nombre	Type
M_1	1	74121, pas de L121 ou LS121
M_2 - M_3	1	74123, pas de L123 ou LS123
03 - 38	1	7403, 74LS03, 7438, 74LS38
REG	1	µA 7805, MC 7805, régulateur 5 V 1 A TO 220
T_1, T_2, T_3	3	BC 107, 108, 109, 547, 548, 549, 2N2222A
T_4	1	2N2222A, 2N2219A
BIT	1	LED n'importe quel type
D_1, D_2	2	1N4001, 1N4002 diode 50 V 1 ampère ou plus
P	1	Poussoir type digitest, 1 contact travail
S_0 à S_7	1	Bloc de 8 mini-interrupteurs en boîtier DIL
S_8	1	Commutateur rotatif 3 circuits, 4 positions
	20	Résistances 1/2 W 5 % : 2 × 220 Ω, 1 × 390 Ω, 1 × 470 Ω 9 × 2,2 kΩ, 1 × 3,3 kΩ, 1 × 4,7 kΩ, 1 × 6,8 kΩ, 1 × 10 kΩ, 1 × 15 kΩ, 2 × 100 kΩ
	3	Chimiques : 1 × 100 µF 10 V, 1 × 22 µF 15 V, 1 × 1 µF 10 V
	6	Céramique ou mylar : 1 × 0,33 µF, 3 × 22 nF, 1 × 10 nF 1 × 100 pF
	2	Supports 14 pattes (facultatifs)
	1	Support 16 pattes (facultatif)
	2	Supports 16 pattes obligatoires et 1 support Textool
ou	1	Support Textool seul (voir texte)

Fig. 3. – Nomenclature des composants.

tion, ouvrez l'interrupteur du +V et celui du +5 V.
– Insérez la mémoire sur son support.
– Fermez l'interrupteur du + 5 V ET ENSUITE celui du +V. L'ordre inverse peut conduire à la destruction de la mémoire.
– Sélectionnez l'adresse à programmer avec S_0 à S_7 ; S_0 correspond à A_0 et S_7 à A_7 et ils sont disposés dans l'ordre sur la carte.
– Positionnez S_8 sur le bit à programmer à cette adresse. Si la mémoire est vierge, la LED doit être allumée indiquant un bit à 1.
– Appuyez sur P une fois. La LED doit s'éteindre. Si elle ne s'éteint pas, renouvelez les appuis sur P jusqu'à concurrence de dix. Si, au dixième, la LED n'est toujours pas éteinte, la mémoire est défectueuse.
– Lorsque le bit voulu est programmé, passez au bit suivant (à la même adresse ou à une autre adresse) en manipulant S_8 et (ou) S_0 à S_7 et procédez comme ci-avant pour manipuler P.
– Lorsque la programmation est terminée, ouvrez l'interrupteur de +V AVANT celui du +5 V ; l'ordre inverse risquant de détruire la mémoire.

Le montage peut aussi être utilisé pour lire une PROM ; il suffit de ne pas toucher P et, pour plus de sûreté, de laisser ouvert l'interrupteur du +V. La LED indique alors l'état de la sortie sur laquelle elle est connectée. Dernière remarque : un taux de 5 % de rejet est normal parmi les PROM bipolaires : si vous êtes nettement au-dessus de ces valeurs ou si plusieurs mémoires vierges et neuves ne se programment pas, votre montage comporte certainement un défaut qui, s'il n'est pas dû à une erreur de câblage ou un composant défectueux, peut provenir des temps générés par les monostables qui doivent trop s'éloigner des valeurs préconisées ; un contrôle à l'oscillo s'impose alors.

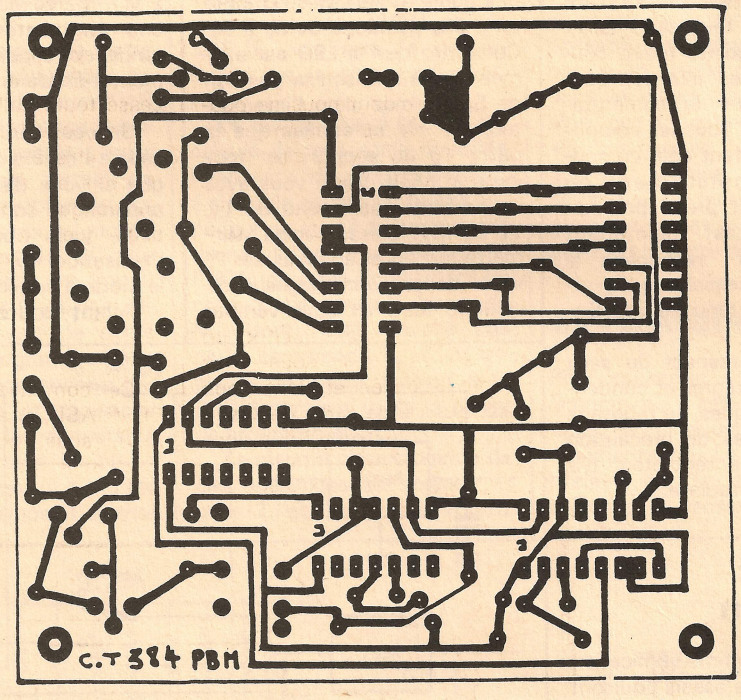

Fig. 4. – Circuit imprimé du programmateur, vu côté cuivre, échelle 1.

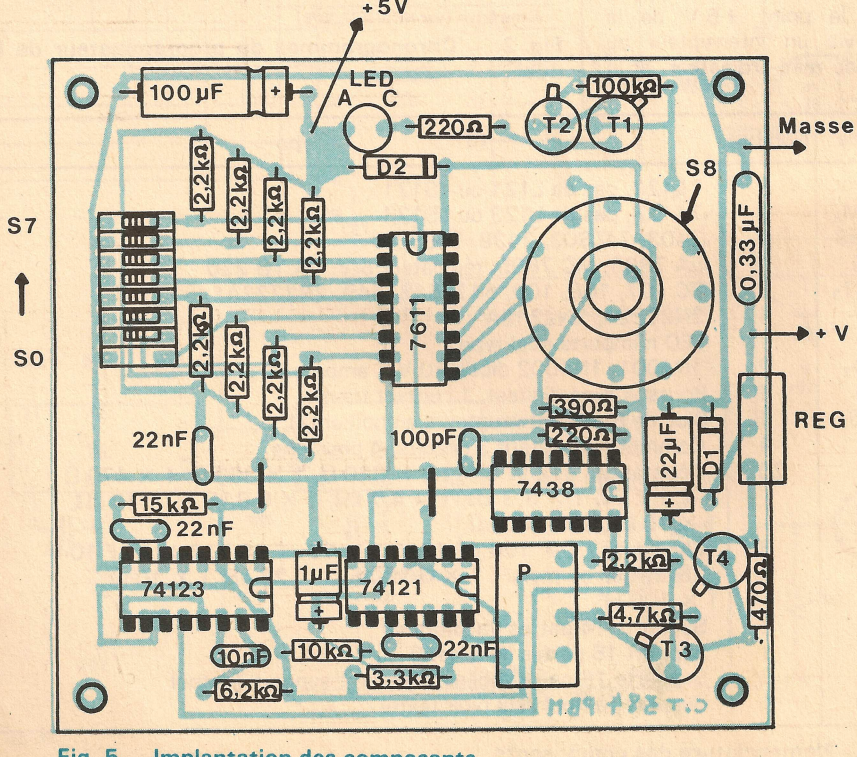

Fig. 5. – Implantation des composants.

A propos d'IVG 09

Cette carte commence à faire couler beaucoup d'encre et à donner des soucis à nombre d'entre vous. En effet, hormis quelques rares fonds de tiroirs, les mémoires qui l'équipent (les TMS 4044 ou leurs équivalentes exactes) commencent à devenir introuvables sur le marché et vous êtes cependant très nombreux à vouloir réaliser cette carte. Pour ce faire, et pour ne pas perdre la compatibilité avec tout ce qui existe déjà tant sur les plans logiciels que matériels, nous sommes en train de concevoir une carte de remplacement dont les propriétés seront les suivantes :

Utilisable sur le système actuel à la place d'IVG 09.
– Ne nécessite aucune modification de nos logiciels.
– Utilisation de mémoires très répandues et dont l'approvi-

sionnement est assuré pour longtemps encore.
– Prix de revient moindre qu'IVG 09 pour la version de base de la carte.
– Diverses options proposées avec le même circuit imprimé de base ; options qui peuvent être ajoutées à tout instant sur la carte.
– Utilisation du 6845 et du 6821, de façon à ne pas grever le budget de ceux d'entre vous qui ont déjà approvisionné ces circuits pour IVG 09 et qui sont restés « coincés » à cause des TMS 4044.

Parmi les options que comportera cette carte, on peut citer, entre autres choses :
– Possibilité de travailler en couleur avec huit couleurs différentes par caractère et huit couleurs différentes pour le fond au niveau de chaque caractère.
– Générateur de caractères en RAM permettant à l'utilisateur de charger en temps réel son propre jeu de caractères et de réaliser ainsi des graphismes de grande qualité avec beaucoup de facilité.
– Sortie couleur péritélévision.
– Enfin, et sous réserve des dernières mises au point qui sont en cours actuellement (ces lignes sont écrites en mars), augmentation de la résolution graphique puisque nous espérons monter à 256 points sur 256 points.

Le circuit imprimé de cette carte devrait être disponible dès le mois prochain chez Facim et nous pensons être en mesure de vous présenter cette réalisation dans notre numéro de juin.

A propos de CGC 09

Vous êtes nombreux à souhaiter utiliser cette carte comme terminal dans une configuration que nous avions prévue à l'origine du système et que nous avons allègrement oubliée depuis. Deux problèmes se posent alors :
– Quel doit être le moniteur puisque TAVBUG 09 et TAVBUG 09 V 1.0 ne sont pas prévus pour ce cas ?
– Où devez-vous brancher le clavier puisque la carte CGC 09 ne comporte pas de PIA ?

Nous étudions actuellement une solution à ce problème et espérons vous donner la réponse dans notre prochain numéro. Quoi qu'il en soit, et si vous vous étiez déjà procuré un moniteur TAVBUG 09 ou un DOS avec TAVBUG 09 V1.0 auprès de l'auteur, soyez assuré de l'échange de cette PROM pour une compatible de votre carte sans aucun frais. Toutes informations à ce sujet vous seront données avec la solution évoquée ci-avant.

L'horloge temps réel

Nous attendons toujours le logiciel au moment où nous écrivons ces lignes (mars), mais nous pensons que le problème que rencontre le concepteur de ce module devrait aboutir très vite à une solution. Ceux d'entre vous qui ont déjà commandé cette carte auront peut-être ce logiciel entre les mains au moment où ils liront ces lignes et, si tel est le cas, sa description paraîtra dans le numéro suivant en raison des délais d'impression.

La fourniture des logiciels

Vu le courrier que nous recevons depuis quelques temps, une mise au point s'impose à ce sujet. Les logiciels décrits dans ces articles sont fournis exclusivement par l'auteur de ces lignes selon les conditions décrites dans le document « Informations 6809 » que vous pouvez vous procurer par une procédure maintes fois décrite. Certains autres logiciels ont été réellement développés par des sociétés ou des particuliers ; c'est le cas de Graphix de la Centrale d'Achat Informatique, c'est le cas aussi de logiciels réalisés par Micropross mais dont nous ne pouvons vous parler, faute d'en avoir eu un exemplaire entre les mains.

Quoi qu'il en soit, aucune société n'est à ce jour autorisée à distribuer les logiciels figurant sur le document « Informations 6809 » édité par l'auteur. La société Saint-Ignan Informatique que nous vous avions signalée et dont nous avons cessé de parler en raison des nombreux problèmes dont vous nous avez fait part, distribue ces logiciels sans notre accord après avoir remplacé dans ceux-ci les trois lettres TAV (de TAVBUG ou TAVDOS par exemple) par SIE (ce qui donne SIEBUG, SIEDOS, etc.). Ces logiciels sont des copies non autorisées des logiciels de l'auteur et leur fonctionnement n'est absolument pas garanti sur le système décrit dans ces pages. Par ailleurs, l'auteur a cessé toute collaboration avec cette société depuis août 1983 et elle n'a en aucun cas le droit de se prévaloir d'un quelconque accord avec nous sous quelque forme que ce soit.

Le compilateur Basic

Ce compilateur référencé COMBASIC peut vous être fourni sous forme d'une dis-

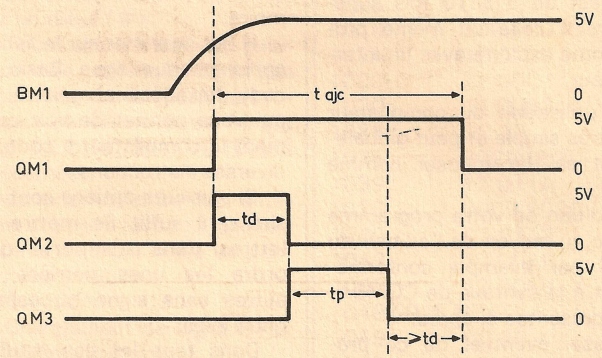

Fig. 7. – Câblage du programmateur sur le système.

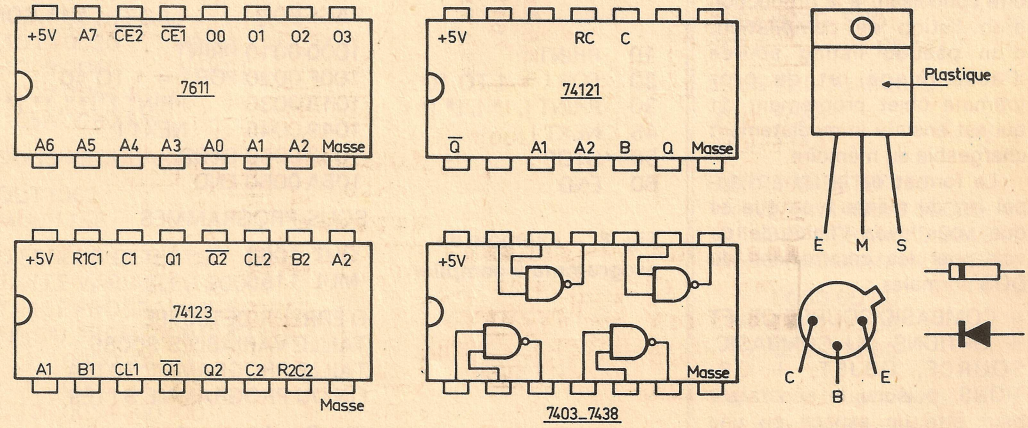

Fig. 6. – Brochage des semi-conducteurs.

quette 40 ou 80 pistes, 5 pouces ou 3 pouces. La disquette supporte un seul fichier baptisé COMBASIC.CMD qui est le compilateur lui-même.

Rappelons qu'un compilateur, par opposition à un interpréteur, traduit lors de la phase dite de compilation, le langage évolué qui est ici le Basic en code machine directement exécutable par le microprocesseur. En d'autres termes, un programme Basic passé au compilateur est transformé en du code hexadécimal, que vous pouvez ensuite charger en mémoire et faire fonctionner alors que le compilateur n'est plus présent sur la machine. Les avantages majeurs du compilateur sur l'interpréteur sont une occupation mémoire beaucoup moins importante, puisque le compilateur n'a pas à être présent en mémoire lors de l'exécution du programme, et une rapidité d'exécution des programmes qui est de 3 à 10 fois supérieure à celle du même programme exploité avec un interpréteur.

L'utilisation du compilateur est très simple et peut globalement se décomposer comme suit :

– Edition de votre programme Basic au moyen de l'éditeur du DOS par exemple conformément à la syntaxe de COMBASIC présentée ci-après.

– Essai éventuel de ce programme avec le Basic disquette moyennant certaines restrictions dues aux différences entre ces deux programmes.

– Compilation proprement dite conduisant à la production d'un listing de compilation, d'un pseudo listing source d'assemblage, et du programme objet proprement dit qui est ensuite immédiatement chargeable en mémoire.

Le format de la ligne d'appel est du même type que ce que vous avez l'habitude de voir pour les commandes du DOS normales :

– COMBASIC SOURCE OBJET + OPTIONS OU COMBASIC, SOURCE, OBJET, + OPTIONS, puisque le séparateur peut être un espace ou une virgule.

SOURCE est le nom du fichier contenant la source de votre programme, c'est-à-dire le texte Basic de celui-ci.

OBJET est facultatif. Si ce nom est spécifié le compilateur placera dans ce fichier le code obtenu après compilation ; ce fichier aura donc généralement un suffixe BIN ou CMD. Aucune valeur par défaut n'est choisie pour ce suffixe et c'est à vous de le spécifier. Si OBJET n'est pas spécifié, le compilateur ne produira qu'un listing (c'est très utile pour voir les éventuelles erreurs de compilation avant de faire réaliser le fichier OBJET).

OPTIONS sont des lettres qui permettent de spécifier diverses options de compilation :

– L interdit la sortie du listing de compilation.

– S génère la table des symboles, c'est-à-dire la liste de toutes les variables et constantes symboliques utilisées dans le programme Basic avec leurs implantations en mémoire.

– H fait imprimer sur le listing, après chaque ligne Basic, le code hexadécimal généré, ce qui vous permet de voir comment le compilateur a codé les diverses instructions.

Si plusieurs options sont désirées, il suffit de mettre les lettres dans n'importe quel ordre les unes derrière les autres sans signe ou espace entre elles.

Dans tous les cas, sauf si vous ne voulez pas de listing bien sûr, une cartographie mémoire du programme compilé est affichée indiquant son adresse de fin, la taille mémoire des programmes, la taille mémoire des variables et les sous-programmes standards de la bibliothèque incluse dans le compilateur appelés avec leurs adresses d'implantation.

La figure 8 présente, à titre d'exemple, un programme Basic élémentaire tel que vous pouvez le frapper sous contrôle de l'éditeur. Ce même programme compilé sans aucune option est visible figure 9. Le listing comporte trois « champs » : le plus à gauche est l'adresse en hexadécimal de la première instruction en langage machine correspondant à la ligne Basic qui suit ; le deuxième champ est le numéro de ligne Basic et le troisième champ est la ligne Basic que vous aviez frappée formatée de façon lisible.

En fin de listing, les sous-programmes appelés, ici OUT et MUL, sont indiqués avec l'adresse de leur première instruction. Viennent ensuite des indications relatives à la taille du programme et des variables associées.

La figure 10 présente ce même listing avec l'option S activée. Les informations supplémentaires visibles sont les adresses mémoire des variables utilisées. Les variables ont le même nom que celui qu'elles portent dans votre programme Basic. Toutes les variables marquées * ou d'un nom ne figurant pas dans votre programme sont propres aux sous-programmes du compilateur.

La figure 11 enfin vous montre ce que donne l'option H (et l'option S que nous avions laissée activée) ; chaque ligne Basic est suivie par les codes hexadécimaux qu'elle fait générer. Ainsi, par exemple, la ligne 10 PRINT est traduite en un 17 00A3 qui n'est autre qu'un LBSR $A3, c'est-à-dire un saut au sous-programme de sortie d'une ligne blanche se trouvant dans le sous-programme OUT du compilateur. Cette analyse peut aller plus loin ; ainsi, si vous regardez les codes qui suivent la ligne 0045, vous pourrez facilement les traduire en langage machine suivant :

LDD $30 qui charge D avec la variable I (I est en 0030 comme indiqué en fin de listing).

ADDD #01 qui augmente I d'une unité, puisque le pas de notre boucle FOR NEXT est de 1.

STD $30 qui place le contenu de D (donc I) en 30.

CMPD $0032 qui compare I avec sa valeur maximale qui est stockée en 0032.

BGT 03 qui fait sauter sur Stop si I est égal ou supérieur à sa valeur maximale.

LBRA $FFC2 qui fait remonter au début de la boucle FOR NEXT si I est inférieur à sa valeur maximale.

Généralités

Il faut avoir présent à l'esprit que le programme source à compiler se présente presque exactement comme un pro-

```
10   PRINT
20   FOR I = 1 TO 10
30   PRINT I, I* I, I* I* I
45   NEXT I
50   STOP
60   END
```

Fig. 8. – Exemple de programme à compiler.

```
PAGE 001                    *** COMPILATEUR BASIC C.T. 6809 ***

1000 0010 PRINT
100F 0020 FOR I = 1 TO 10
1019 0030       PRINT I, I* I, I* I* I
1049 0045       NEXT I
1057 0050 STOP
105A 0060 END

SOUS-PROGRAMMES

 OUT 1062
 MUL 1156

0 ERREUR DETECTEE
TAILLE VARIABLES $0085
TAILLE PROGRAMME $01EA
FIN DU PROGRAMME $11E9
```

Fig. 9. – Listing de compilation sans aucune option.

gramme que vous passeriez à l'interpréteur ; vous ne devez donc pas être dépaysé.

Le programme à compiler comporte donc une suite de lignes qui commencent impérativement par un numéro de un à quatre chiffres maximum, ces chiffres étant décimaux et entiers.

Les commandes multiples sont autorisées sur une même ligne ; elles doivent être séparées par deux points (:). La longueur maximale d'une ligne ne doit en aucun cas excéder 80 caractères.

Les espaces sont ignorés, hormis dans les chaînes de caractères, comme ils n'utilisent pas de place mémoire après compilation, vous pouvez les employer largement pour améliorer la lisibilité du programme. Les REM peuvent aussi être très largement utilisés pour la même raison.

Même si vous ne mettez pas d'espace, le compilateur formate automatiquement votre listing à la compilation pour en améliorer la lisibilité.

Enfin, la dernière ligne d'un programme est celle contenant le END ; la compilation s'arrêtant là et ignorant tout ce qui peut se trouver après.

Nombres et variables

Les nombres sont représentés après compilation sur 16 bits ; cela signifie que le Basic ne travaille que sur des entiers compris entre + 32 767 et – 32 768, lorsque l'on emploie des entiers signés. La plage de variation va de 0 à + 65 535 lorsque l'on travaille sur des nombres non signés.

Les constantes pouvant apparaître dans un programme Basic, peuvent être exprimées en décimal ou en hexadécimal ; dans ce dernier cas, elles doivent être, comme en assembleur, précédées par un dollar ($). Une valeur négative doit être précédée par le signe – ; en revanche, le signe + devant une constante n'est pas autorisé, l'absence de moins étant implicitement considérée comme un plus. Ce Basic se prêtant bien aux opérations logiques, il est possible de représenter le complément d'un nombre (complément à 1, bit à bit de ce nombre représenté en binaire) en faisant précéder celui-ci du signe dièse (#).

Ainsi, par exemple, les constantes suivantes sont valables :

– 300, – 4 000, $ 80, $ 2 500, # 2, #$ FF00.

En revanche, ne sont pas autorisées, par exemple :

– 8.45, 100 000, + 15.

Les variables sont représentées, dans ce Basic, par une lettre de A à Z ou par une lettre suivie d'un chiffre de 0 à 9.

Comme dans tout Basic, les variables peuvent être indicées et former des tableaux à une ou deux dimensions. Les indices commencent à 1 et peuvent aller jusqu'à 255. Attention cependant à ce que vous faites ; une directive du style DIMA (255, 255) risque fort de ne pas passer à la compilation car cela génère un tableau de 65 025 éléments, soit toute l'étendue mémoire du 6809 !

Les indices peuvent être des constantes, des variables ou des expressions arithmétiques, par exemple, sont valables : N(M), A(15), B5(C, D), U(N*(A/B), 8).

Le Basic considère une variable non indicée comme étant le premier élément d'un tableau ; ainsi la variable A sera considérée comme étant A(1, 1) ; attention donc au choix de vos noms de variables.

Sachez aussi que :

– Il faut 250 µs pour accéder à un élément d'un tableau à deux dimensions contre 20 µs pour un tableau à une seule dimension.

– Chaque variable ou élément d'un tableau se voit assigner deux octets de RAM.

Opérateurs arithmétiques

Le Basic dispose des quatre opérateurs classiques : addition (+), soustraction (–), multiplication (X) et division (/) ; par ailleurs, il dispose également de quatres opérateurs logiques qui sont :

– Le ET logique représenté par le caractère « et commercial » (&).

– Le OU logique représenté par un point d'exclamation (!).

– Le OU EXCLUSIF logique représenté par le caractère « pour cent » (%).

– Le complément représenté par un dièse (#).

Tous ces opérateurs peuvent être mélangés dans des expressions. Les opérateurs logiques travaillent bit par bit sur les 16 bits des variables concernées.

La priorité entre les différents opérateurs s'établit comme suit, par ordre de priorité décroissante :

– Fonctions

– Changement de signe (–) et complément logique

– ET logique, OU logique et OU EXCLUSIF logique

– Multiplication et division

– Addition et soustraction.

Des parenthèses peuvent, bien sûr, être utilisées pour modifier ces priorités.

Fonctions arithmétiques

Ce Basic dispose de quatre fonctions dites arithmétiques, ce sont :

```
PAGE 001                    *** COMPILATEUR BASIC C.T. 6809 ***
1000 0010  PRINT
100F 0020 FOR I = 1 TO 10
1019 0030       PRINT I, I* I, I* I* I
1049 0045       NEXT I
1057 0050 STOP
105A 0060 END

SOUS-PROGRAMMES

OUT1062
MUL 1156

O ERREUR DETECTEE
TAILLE VARIABLES $0085
TAILLE PROGRAMME $01EA
FIN DU PROGRAMME $11E9
   I    0030 00 00
   *    0032 00 00
  10    0034 00 00
```

Fig. 10. – L'option S fait afficher la table des symboles.

```
PAGE 001
*** COMPILATEUR BASIC
                    C.T. 6809 ***

1000 0010  PRINT
100C       17 00A3
100F 0020  FOR I = 1 TO 10
100F       CC 0001
1012       DD 30
1014       CC 000A
1017       DD 32
1019 0030  PRINT I, I* I, I* I* I
1019       17 0061
101C       DC 30
101E       17 00D0
1021       17 0062
1024       DC 30
1026       30C9 0030
102A       17 0129
102D       17 00C1
1030       17 0053
1033       DC 30
1035       30C9 0030
1039       17 011A
103C       30C9 0030
1040       17 0113
1043       17 00AB
1046       17 005B
1049 0045  NEXT I
1049       DC 30
104B       C3 0001
104E       DD 30
1050       93 32
1052       2E 03
1054       16 FFC2
1057 0050  STOP
1057       7E CD03
105A 0060  END
105A       7E CD03

SOUS-PROGRAMMES

 OUT 1062
 MUL 1156

O ERREUR DETECTEE
TAILLE VARIABLES $0085
TAILLE PROGRAMME $01EA
FIN DU PROGRAMME $11E9

   I    0030 00 00
   *    0032 00 00
  10    0034 00 00
```

Fig. 11. – L'option H fait afficher les codes hexadécimaux générés.

– ABS (EXPRESSION) : qui donne la valeur absolue de l'expression concernée.
– RND ou RND (EXPRESSION) : produit un nombre pseudo-aléatoire compris entre 0 et + 32 767 si RND est utilisé seul. Si une expression suit RND, le nombre produit est le même chaque fois que l'expression se reproduit identique à elle-même.
– PEEK (EXPRESSION) : donne la valeur se trouvant à l'adresse spécifiée par l'expression. Contrairement à beaucoup de Basic, l'expression peut être exprimée en hexadécimal si elle est précédée par un dollar, ce qui est très pratique.
– POS : indique la position actuelle du curseur du terminal (ou de la tête d'impression de l'imprimante).
– SWAP (EXPRESSION) : échange les deux octets de l'expression considérée.
– ERR : indique par un numéro de code le dernier type d'erreur rencontré (voir tableau de la figure 12 pour la signification de ce numéro de code).

Les chaînes de caractères

Une chaîne de caractères est constituée de tout ce qui se trouve compris entre deux guillemets à l'exception classique du retour chariot, du saut ligne et du caractère de contrôle dont la valeur en ASCII est $ 1A, car celui-ci joue un rôle particulier. Si un guillemet doit être inclus dans une chaîne de caractères, il doit être frappé en double pour ne pas le confondre avec ceux de début et de fin de chaîne.

Une chaîne de caractères peut avoir une longueur maximale telle qu'elle tienne dans une ligne standard du Basic (80 caractères maxi).

Les noms des variables chaînes de caractères ne peuvent être constitués que par une lettre de A à Z suivie par un dollar. Une chaîne de caractères peut être utilisée avec dimensionnement ou faire appel à un dimensionnement implicite ; dans ce cas, le compilateur réservera 32 octets de mémoire pour la chaîne et sa longueur ne pourra donc dépasser 32 caractères ; il vous faudra donc déclarer toute chaîne de caractères plus longue que 32 caractères. Par ailleurs, si vous voulez économiser de la RAM sur le programme compilé, vous avez intérêt à déclarer systématiquement toutes vos chaînes avec leurs longueurs réelles.

L'opérateur d'addition (+) peut être utilisé sur les chaînes de caractères, par exemple :
– « HAUT » + « PARLEUR » est égal à la chaîne de caractères « HAUT-PARLEUR ».

Une chaîne de caractères peut être initialisée à « rien » en début de programme de la façon suivante :
– A$="".

Cette possibilité d'initialisation est importante, car il ne faut pas oublier que plus tard cette chaîne sera constituée par plusieurs octets de RAM qui peuvent contenir n'importe quoi à la mise sous tension.

Les fonctions relatives aux chaînes de caractères

Dans les définitions qui vont suivre, N, M seront des constantes, des variables ou des expressions numériques tandis que X$ sera une variable ou une expression chaîne de caractères.

Les fonctions dont on dispose sont divisées en deux groupes, les fonctions produisant un résultat numérique et celles produisant un résultat chaîne de caractères. Nous allons commencer par les derniers.

– CHR$ (N) : produit un caractère dont N est le code ASCII.
– LEFT$ (X$, N) : produit une chaîne de caractères constituée par les N caractères de gauche de la chaîne X$.
– MID$ (X$, N, M) : produit une chaîne de caractères qui est le sous-ensemble de X$ commençant au nième caractère de X$ en partant de la gauche et comprenant M caractères.
– RIGHT (X$,N) : produit une chaîne de caractères constituée par les N caractères en partant de la droite de X$.
– STR$(N) : produit une chaîne de caractères dont le contenu est N, ainsi : STR$ (4567) = la chaîne de caractères « 4567 ».
– TRM$ (X$) : enlève les espaces se trouvant en fin d'une chaîne de caractères ; ainsi : TRM$
« BASIC » = « BASIC »

Les fonctions suivantes exploitent des chaînes de caractères pour produire des résultats numériques, ce sont :
– ASC (X$) : qui donne le code ASCII du premier caractère de la chaîne.
– LEN(X$) : qui donne la longueur de la chaîne de caractères concernée.
– SUBSTR (X$, Y$) : qui cherche la chaîne Y$ dans la chaîne X$; si la recherche s'avère positive, cette fonction fournit un nombre égal à la position du premier caractère de Y$ dans X$; sinon, la valeur 0 est affectée au résultat de cette fonction.
– VAL (X$) : a pour effet de convertir une chaîne de caractères en son équivalent numérique, ainsi VAL (« 1234 ») = 1234.

De toutes ces fonctions, VAL et STR$ peuvent produire des erreurs si les données fournies sont incompatibles avec la fonction, ces erreurs peuvent être traitées par un ON ERROR GOTO.

Ce Basic, de par le fait qu'il est compilateur, permet d'utiliser en plus de ce qui a été vu, une chaîne de caractères particulière qui est son « buffer » d'entrées/ sorties ; celui-ci s'appelle tout simplement BUF$ et il contient au maximum 129 octets. Les commandes suivantes peuvent, par exemple, être utilisées :
– Pour le lire : PRINT BUF$ ou READ #N, BUF$.
– Pour y écrire : INPUT BUF$ ou WRITE #N, BUF$.

Assignation boucles et sauts

Comme dans tout Basic, il existe une directive d'assignation de valeur : LET ; son utilisation est optionnelle et A = 2 aura le même effet que LET A = 2.
Il est également possible de placer une valeur en mémoire à une adresse spécifiée au moyen de la directive POKE, sous la forme POKE (X) = Y où Y est une expression qui est calculée puis tronquée à 8 bits et qui est ensuite placée à l'adresse spécifiée par l'expression X.

Contrairement à beaucoup de Basic interprétés, il est possible de faire appel directement à des sous-programmes écrits en langage machine ; cela se fait au moyen de la directive CALL sous la forme : CALL ADRESSE ou ADRESSE représente l'adresse de début du sous-programme ainsi appelé. L'adresse, comme toutes les variables, peut être spécifiée en décimal ou hexadécimal si elle est précédée d'un dollar. Le sous-programme ainsi appelé doit impérativement se terminer par un RTS et doit rendre la pile telle qu'elle se trouvait lors de l'appel.

Les boucles s'exécutent, comme dans tout Basic, par une commande FOR NEXT de la façon suivante :
– FOR VARIABLE = EXPRESSION TO EXPRESSION STEP EXPRESSION ou VARIABLE

NUMERO	TYPE D'ERREUR
0 à 28	Erreur DOS, codes indiqués p. 189 n° 1692
29	Sans signification
30	Sans signification
31	Sans signification
32	Débordement lors d'une multiplication
33	Division par zéro
34	Erreur de conversion ASCII – binaire

Fig. 12. – Tableau des erreurs exploitables par la fonction ERR.

est une variable non indicée numérique et où EXPRESSION sont n'importe quelles expressions valides. Si STEP n'est pas indiqué, la valeur prise par défaut est 1.

De plus, en raison même du procédé de compilation qui produit un code machine très compact et rapide :
— La boucle sera toujours exécutée au moins une fois, quelle que soit la valeur précisée après TO.
— A la fin de la boucle, le compteur utilisé sera supérieur à la valeur finale spécifiée d'une quantité égale à STEP.
— Ces boucles peuvent être quittées et reprises quand vous le désirez.
— Lors de la compilation, vous pouvez avoir jusqu'à 16 boucles actives simultanément.
— Les valeurs initiales, finales et le pas (valeur indiquée après STEP) peuvent être positives ou négatives.

Les sauts peuvent faire appel aux directives GOSUB, IF THEN, ON ERROR GOTO, ON GOTO et ON GOSUB que nous allons étudier dans cet ordre.

GOSUB est très classique et sert à appeler un sous-programme en Basic, sous-programme qui doit impérativement se terminer par un RETURN. La syntaxe de GOSUB est très simple : GOSUB NUMERO DE LIGNE. La compilation de ces deux directives (GOSUB et RETURN) est très courte puisque GOSUB donne un JSR, tandis que RETURN donne un RTS.

IF THEN est très classique, son utilisation est la suivante :
— IF EXPRESSION RELATION EXPRESSION THEN NUMERO DE LIGNE ou RELATION représente un des six opérateurs de comparaison qui sont : inférieur (<), supérieur (>), égal (=), différent (<>), inférieur ou égal (= < ou < =) est supérieur ou égal (= > ou >).

Par ailleurs, le THEN peut être remplacé par un GOSUB.

Si la comparaison est vraie, le programme se branche à la ligne indiquée, et si le THEN est un GOSUB, le RETURN a pour effet de faire revenir à la ligne suivant immédiatement celle du IF GOSUB. Compte tenu de la possibilité de mettre plusieurs commandes sur la même ligne, le IF THEN peut jouer le rôle d'un IF THEN ELSE, par exemple :
— IF A> 2 THEN 1 000 ELSE 200 se traduira par :
— IF A> 2 THEN 1 000 : GOTO 200.

Le ON ERROR GOTO est un peu analogue à celui de l'interpréteur Basic étendu. Il s'utilise de la façon suivante : ON ERROR GOTO ou ON ERROR GOTO NUMERO DE LIGNE. Dans le premier cas, la fonction ON ERROR GOTO est désactive, tandis que, dans le second cas, toute erreur détectée à la suite de cette directive causera un branchement au numéro de ligne précisé après le ON ERROR GOTO.

Les erreurs qui peuvent être détectées par cette directive sont les suivantes :
— Division par zéro dans une expression arithmétique.
— Erreur de conversion ASCII — binaire dans un INPUT, READ, VAL (X$).
— Dépassement de capacité en multiplication dans une expression arithmétique.
— Erreur d'entrée/sortie disque.

Il faut alors utiliser la fonction ERR vue ci-avant pour savoir à quel type d'erreur l'on a à faire.

ON GOTO et ON GOSUB travaillent de la façon suivante : ON EXPRESSION GOTO (GOSUB) NUMERO 1, NUMERO 2, etc., où EXPRESSION représente une expression valide et NUMERO 1, NUMERO 2, etc., des numéros de lignes. L'expression est calculée et le résultat indique à quel numéro de ligne faire le branchement ; ainsi : ON A GOTO 100, 200, 300, 400 fera brancher le programme en 100 si A = 1, en 200 si A = 2, etc. Si le calcul de l'expression donne une valeur nulle, le programme passe à la ligne suivante.

STOP est un branchement un peu particulier, en ce sens qu'il arrête l'exécution du programme et cause un saut inconditionnel au dos en son point d'entrée « chaud » CD03.

Les directives machine

Un compilateur étant très près du langage machine, il est possible d'agir sur certains paramètres au moyen des directives suivantes. GEN permet de générer au sein d'un programme une suite de valeurs qui sont insérées telles quelles dans celui-ci lors de la compilation. Cette possibilité est utilisée pour ajouter quelques instructions lorsqu'elles sont plus faciles à écrire en langage machine qu'en Basic.

L'utilisation se fait de la façon suivante :
— GEN NOMBRE 1, NOMBRE 2, NOMBRE 3, etc., ou NOMBRE 1, NOMBRE 2, etc., sont des valeurs décimales ou hexadécimales qui seront placées dans le programme généré. Si un des nombres est inférieur à 255, il occupera quand même un octet, même si le nombre est précédé de zéros. Pour faire un branchement à l'adresse $ 1000 par exemple, vous pouvez écrire :
— GEN $ 7E (code du JMP), $ 10, $00.

Les entrées sorties

Les entrées sorties se font par un « buffer » de 129 octets qui peut donc contenir 128 caractères. Celui-ci est automatiquement placé après toutes les autres affectations mémoire, sauf le « buffer » de chaîne de caractères.

Les entrées se font au moyen d'un INPUT sous la forme : INPUT VARIABLE 1, VARIABLE 2, etc.

Un retour chariot, saut ligne et un point d'interrogation sont alors générés et le programme lit ensuite ce qui entre jusqu'à ce que son « buffer » soit plein ou qu'un retour chariot soit frappé. La frappe d'un CNTRL X fait imprimer « DEL » et efface complètement la ligne frappée, tandis qu'un CNTRL O renvoie le dernier caractère frappé et l'annule dans le « buffer » d'entrée.

Lorsque plusieurs variables sont attendues, elles doivent être séparées par une virgule ou un espace si ce sont des variables numériques, et par une virgule si ce sont des variables chaînes de caractères (les espaces étant comptés comme éléments de la chaîne).

PRINT s'utilise d'une façon analogue sous la forme :
— PRINT EXPRESSION 1 DELIMITEUR EXPRESSION 2 DELIMITEUR, etc., où EXPRESSION 1, EXPRESSION 2, etc., peuvent être des expressions numériques ou des chaînes de caractères tandis que DELIMITEUR peut être une virgule ou un point virgule. Dans le premier cas, les expressions à imprimer le sont dans des zones de 8 caractères de large ; la virgule faisant passer d'une zone à l'autre ; dans le deuxième cas les expressions se suivront sans espacement. Parmi les expressions : la fonction TAB peut être utilisée. Ce compilateur dispose en outre de possibilités d'entrées sorties sur disque avec création et accès aux fichiers.

Comme pour l'interpréteur BASIC étendu, les accès disque se font par des numéros de voies ou de canaux compris entre 0 et 9. Le compilateur tolère donc un maximum de 10 canaux ouverts simultanément.

Les directives d'accès aux fichiers disque dont on dispose sont les suivantes :
— CLOSE FILES : doit être utilisé lorsque tous les accès disque sont terminés. Cette commande a pour effet de fermer tous les fichiers qui pourraient être restés ouverts. Il est également conseillé de placer cette directive dans les programmes de gestion des erreurs lors des accès disque ; cela vous assurera d'avoir un disque parfaitement « propre », quoi qu'il arrive.
— OPEN : permet d'ouvrir un fichier disque pour une lecture ou une écriture. La syntaxe est : OPEN # NUMERO, NOM DE FICHIER où NUMERO est le numéro de voie (compris entre 0 et 9) qui sera affecté à ce fichier et où NOM DE FICHIER est un nom conforme à la syntaxe du DOS. Un OPEN doit être effectué préalablement à tout accès à un fichier, de plus un numéro de voie ne doit être affecté qu'à un fichier et à un seul à un instant donné. Enfin,

un fichier peut être ouvert en lecture ou en écriture. Lors de l'exécution de la directive OPEN, le BASIC cherche dans le répertoire des fichiers si celui-ci existe ou non. Dans le premier cas, le fichier est ouvert en lecture, dans le second cas, le fichier est ouvert en écriture. Pour savoir dans quel état se trouve l'ouverture d'un fichier donné, on dispose de la commande STATUS.

– CLOSE : permet de fermer un fichier et de libérer ainsi son numéro de voie pour un autre usage. La syntaxe en est : CLOSE # NUMERO où NUMERO est le numéro de voie à libérer. Cette commande agit que le fichier soit en lecture ou en écriture.

– WITE : permet d'écrire dans un fichier ouvert en écriture. La syntaxe en est : WRITE # NUMERO, LISTE où NUMERO est le numéro de la voie où écrire et où liste représente une liste de constantes, de variables ou d'expressions numériques ou chaînes de caractères, séparées par des virgules. Chaque élément de la liste est considéré comme un article de l'enregistrement ainsi défini. La longueur totale d'un enregistrement donné ne peut excéder 128 octets. Il faudra donc faire autant d'enregistrements que nécessaire si vous dépassez cette longueur. Tous les articles sont convertis en ASC II sur le disque et créent donc ainsi un fichier « texte ». La virgule séparatrice se retrouve dans le fichier ainsi créé.

La figure 15 clarifie un peu tout cela en montrant ce qui se trouve réellement sur le disque après un WRITE donné.

– RWRITE réalise la même fonction que WRITE, mais permet en plus de spécifier un numéro d'enregistrement sous la forme RWRITE # NUMERO, ENREGISTREMENT, LISTE où ENREGISTREMENT est le numéro évoqué ci-avant. NUMERO et LISTE ont la même signification que pour la commande WRITE vue ci-avant.

– READ : permet de lire le contenu d'un enregistrement existant de la façon suivante : READ # NUMERO, LISTE où NUMERO est un numéro de voie ouverte en lecture et où LISTE est une liste de variables séparées par des virgules. Le fichier correspondant à la voie ainsi spécifiée doit être un fichier texte et le nombre d'articles qu'il contient doit être compatible au nombre de variables demandées. S'il y a plus d'articles dans l'enregistrement que de variables dans la liste, ceux qui sont « en trop » seront ignorés. La liste de variables peut comporter tous les types admis par le Basic (numérique, chaîne de caractères, indices).

– RREAD est à READ, ce que RWRITE est à WRITE. L'utilisation et la syntaxe sont les mêmes que pour RWRITE vu ci-avant.

– CHAIN : permet au programme BASIC en cours d'exécution de charger et de lancer l'exécution d'un autre programme. La syntaxe est CHAIN NOM où NOM est le nom du programme à lancer. Les contraintes suivantes sont à respecter : le programme ainsi appelé doit avoir, au préalable, été sauvegardé sur disque sous forme binaire : c'est-à-dire avoir été assemblé ou compilé au préalable ; de plus, le programme doit être muni d'une adresse de transfert (voir mode d'emploi du DOS). Il faut être conscient du fait que, lors d'un CHAIN, le programme BASIC passe « la main » au DOS si bien que, si une erreur se produit (fichier non trouvé, disque pas prêt, etc.), le contrôle ne sera pas rendu au programme BASIC mais au DOS.

– RESTORE et SCRATCH : RESTORE permet de fermer un fichier ouvert en lecture ou écriture et de l'ouvrir à nouveau en lecture. La syntaxe est RESTORE # NUMERO 1, # NUMERO 2, etc., où NUMERO 1, NUMERO 2, etc., sont les numéros de voie des fichiers concernés.

SCRATCH joue le même rôle mais ouvre à nouveau le ou les fichiers concernés en écriture. La syntaxe est identique à RESTORE. Attention, de par sa fonction SCRATCH détruit le contenu des fichiers concernés, puisque ceux-ci sont rendus à nouveau disponibles pour une écriture.

– KILL : est à manier avec beaucoup de précautions. KILL NOM détruit de façon définitive le fichier dont le nom est spécifié.

– EOF : permet de savoir si l'on a atteint la fin d'un fichier... La syntaxe en est EOF (# NUMERO) qui donne une valeur de 1 si l'on a atteint la

NUMERO	SIGNIFICATION
02	Numéro de ligne en double ou trop grand
03	Directive inconnue
04	Erreur de syntaxe
05	Nom de variable absent ou en erreur
06	Signe égal manquant
07	Référence à une ligne inexistante
08	Parenthèse droite absente ou en nombre incorrect
09	Opérande absent dans une expression
10	Numéro de ligne de destination absent ou en erreur
11	Nombre manquant
12	Mauvais enchevêtrement FOR-NEXT
13	Débordement de la table des symboles
14	Sans signification
15	Erreur sur un opérateur de relation
16	Délimiteur (, ou ;) absent
17	Guillemet absent en fin de chaîne
18	Compteur incorrect ou absent dans un FOR-TO
19	Tableau déjà défini
20	Erreur d'indice : indice manquant ou indices trop nombreux
21	Erreur d'indice : indice plus petit que 1 ou plus grand que 255
22	Débordement de la RAM affectée aux variables au-dessus de FFFF
23	Référence à un tableau non dimensionné
24	Erreur d'indice
25	Argument d'une fonction manquant ou incorrect (numérique)
26	Option illégale
27	Opérateur non reconnu dans une expression chaîne de caractères
28	Opérateur de concaténation (+) absent
29	Argument d'une fonction manquant ou incorrect (chaîne)
30	Nombre de boucles FOR-NEXT simultanées supérieur à 16
31	Débordement de la table des numéros de lignes
32	Débordement de la mémoire de programme au-dessus de FFFF
33	Sans signification
34	GOTO ou GOSUB manquant
35	Numéro de canal incorrect (non compris entre 0 et 9)
36	Erreur d'entrées/sorties disque

Fig. 13. – Liste des erreurs de compilation

NOM	FONCTION
OUT	Sortie de caractères
INP	Entrée de caractères
DSK	Entrées/sorties sur disquettes
MUL	Multiplication arithmétique
DIV	Division
RND	Générateur de nombres aléatoires
ST1	Fonction relative aux chaînes de caractères
ST2	Fonction relative aux chaînes de caractères
ARR	Multiplication de deux tableaux
IND	Chargement indirect d'un tableau

Fig. 14. – Noms et fonctions des sous-programmes de la bibliothèque du compilateur.

fin du fichier dont le numéro de voie est spécifié, ou une valeur nulle dans les autres cas.
– FILSIZ : permet de connaître la longueur d'un fichier en nombre de secteurs sur le disque. La syntaxe est : FILSIZ (# NUMERO) où NUMERO est le numéro de la voie du fichier concerné.
– STATUS : permet de savoir dans quel état se trouve un fichier. La syntaxe est : STATUS (# NUMERO) où NUMERO est le numéro de voie du fichier concerné. Cette commande donne une valeur nulle pour un fichier non ouvert, une valeur de 1 pour un fichier ouvert en lecture et une valeur de 2 pour un fichier ouvert en écriture.

Exemple d'utilisation : ON STATUS (# 2) GOTO 100, 200, fera passer à la ligne suivante si la voie 2 n'est pas ouverte, fera aller en 100 si elle est ouverte en lecture et en 200 si elle est ouverte en écriture.

Les directives de compilation

Ces directives n'ont aucun équivalent en BASIC interprété puisque l'implantation en mémoire du programme ne vous regarde pas. Ici, il n'en est pas de même et vous pouvez (et vous devez !) indiquer au compilateur où il devra implanter votre programme. Nous allons donc retrouver des directives analogues à celles utilisées en assembleur ; en effet, nous allons commencer par ORG.

ORG sert à définir l'emplacement où va être implanté le code produit par le compilateur. La syntaxe est tout simplement ORG = XXXX où XXXX représente une adresse exprimée en décimal ou hexadécimal (précédée d'un dollar dans ce dernier cas). Cette directive peut être utilisée autant de fois que vous le désirez dans un programme, ce qui vous permet de le découper ainsi en autant de blocs que vous le souhaitez.

Si vous ne précisez pas ORG, la valeur prise par défaut est $ 1 000. Une deuxième directive, un peu analogue à ORG, est à votre disposition. Il s'agit de BASE qui sert à définir l'adresse de début d'implantation des RAM utilisées pour les variables du programme.

Les remarques faites ci-avant pour ORG sont toutes valables, sauf la valeur prise par défaut qui est ici de $ 0030.

Ces directives étant nouvelles dans un BASIC, nous allons vous donner quelques conseils à leur sujet afin que vous puissiez les utiliser au mieux.

Il est conseillé de faire attention à ce que les zones définies pour la RAM ne recouvrent pas celles définies pour le programme. Les informations données par le compilateur en fin de compilation sont très utiles dans ce cas.

Une bonne pratique consiste à commencer un programme par une directive BASE suivie par des DIM de toutes les variables, ce qui fera d'une pierre deux coups. Ensuite la directive ORG marquera le début du programme proprement dit.

La directive DIM peut, elle aussi, être considérée comme une directive de compilation ; en effet, elle sert à déclarer les tableaux et, dans certains cas, les variables simples.

Les tableaux numériques peuvent être déclarés comme étant à une ou deux dimensions sous la forme DIM A (X, Y) où X et Y représentent la taille du tableau A. Ne pas oublier que les indices commencent à 1. Les chaînes de caractères ne peuvent être déclarées que comme variables à une dimension. Cependant, DIM sert aussi, dans ce cas, à indiquer la longueur de la chaîne. Ainsi DIM A$ (50) signifie que la chaîne A$ a une longueur de 50 caractères tandis que DIM A$ (15,30) considère la chaîne A$ comme étant un ensemble de 15 chaînes de 30 caractères. Attention ! dans le premier exemple ci-avant, A$ sera à utiliser sans indice puisque A$ sera une variable simple (DIM n'ayant servi qu'à définir sa longueur), tandis que dans le deuxième exemple, A$ sera à utiliser avec un indice puisque A$ comportera 15 éléments.

DIM peut aussi être utilisé pour déclarer des variables simples dans des cas particuliers. Ainsi, pour déclarer le registre d'état d'un ACIA en $ 8008 comme étant la variable A, ferons-nous :
– BASE = $ 8008
– DIM A (1)

Le registre d'état sera ainsi atteint lorsque l'on utilisera la variable A. Comme nous avons dit dans le paragraphe consacré aux variables que le fait d'écrire A était équivalent à A (1) ou à A (1,1), la définition de A réalisée ci-avant est très correcte.

La dernière directive de compilation doit impérativement être un END qui indique au compilateur qu'il doit s'arrêter. Ce END fait, par ailleurs, générer un saut au point d'entrée chaud du DOS par un JMP en CD03. Si l'endroit où se trouve le END ne correspond pas à la fin effective d'un programme (cas de sous-programmes écrits en fin de listing par exemple), la fin réelle du programme doit être matérialisée par un STOP qui fait aussi générer le même saut.

Les erreurs de compilation

Lors de la phase de compilation, les erreurs détectées sont indiquées sur le listing par un numéro de code conforme à la liste visible figure 13. De plus, une flèche est positionnée, sous la ligne en erreur, à l'emplacement approximatif de celle-ci. Cette dernière fonction ne peut cependant pas être à 100 % exacte et ne doit être considérée que comme une indication. Ces erreurs de compilation ne doivent pas être confondues avec les erreurs détectées par la fonction ERR. Ces dernières sont des erreurs dues à l'exécution du programme alors que celles de compilation sont des erreurs de syntaxe lors de l'écriture du programme.

Enfin, pour information, la figure 14 vous indique la liste des sous-programmes du compilateur qui peuvent apparaître en fin de listing.

Informations diverses

Les manettes de jeux dont nous avons parlé dans notre dernier article ne sont pas oubliées et seront décrites dès que possible, compte tenu de la place disponible et des projets en cours.

Nous avons commencé à recevoir et à ficher des réponses au sondage publié dans le numéro de mars, mais celles-ci sont encore trop peu nombreuses pour avoir une signification et permettre de constituer une liste valable de réalisateurs du système. Vous êtes, au 26-03.84, une centaine seulement à avoir répondu, ce qui fait moins de 10 % des réalisateurs du système.

Conclusion

Nous en resterons là pour aujourd'hui en espérant que la diversité des sujets abordés aura intéressé nombre d'entre vous. Nous vous donnons rendez-vous le mois prochain pour un article dont le contenu n'est pas encore défini, tant nous avons de choses à dire à propos de ce mini-ordinateur au fur et à mesure de son évolution.

C. TAVERNIER

```
90 N = 10
100 WRITE # 2,25 – 400, « TOTO », N
Produit un fichier qui contient :

32 35 2C 2D 34 30 30 20 54 4F 54 4F 2C 31 30  OD
 2  5     –  4  0  0     T  O  T  O     ,  1  0 EOF
```

Fig. 15. – Le contenu d'un fichier disque après un ordre WRITE.