Carte a Microprocesseur 8086
Manuel utilisateur

Traduction MARTIN Hervé - France – herve.mt@free.fr - Version 1.0 – 21-08-2025

wichit.sirichote@gmail.com

Rev 1.0, Janvier 2022

Table des matières

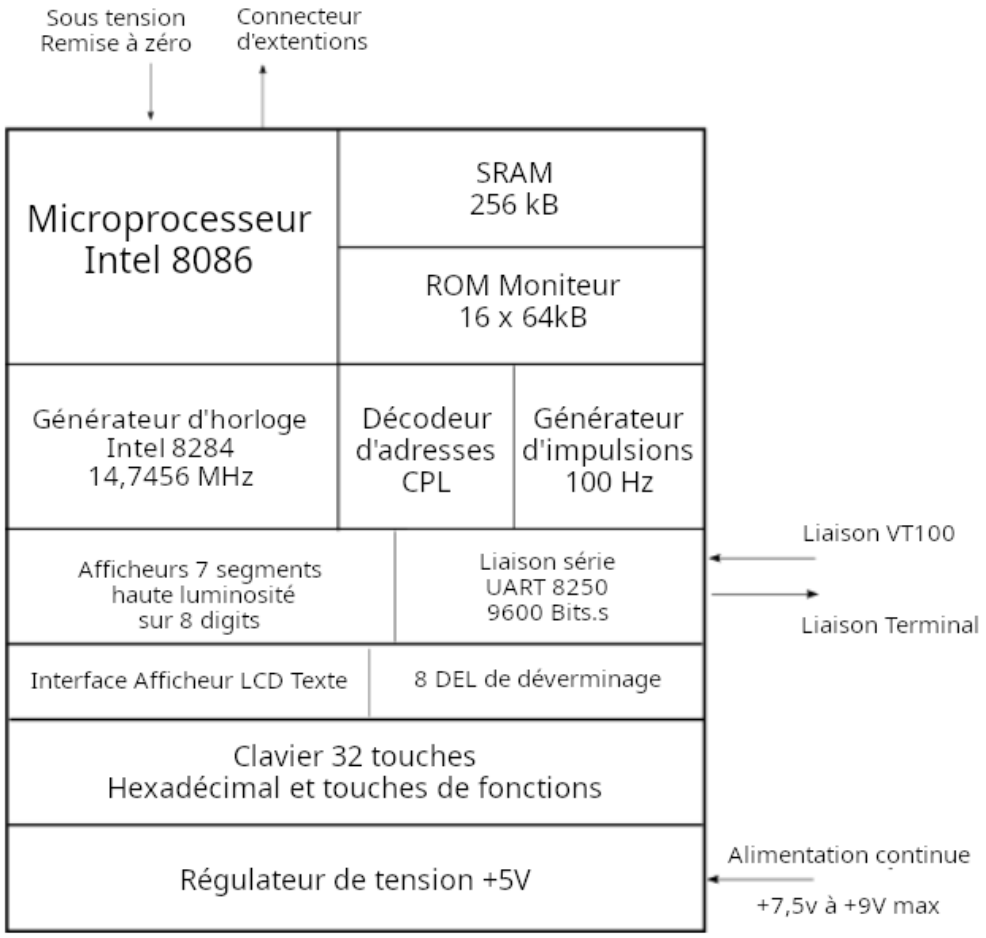
DESCRIPTIF DU MATÉRIEL
ET
PROGRAMME DE SUPERVISION

1 - APERÇU

Le kit microprocesseur 8086 est un ordinateur monocarte conçu pour l'auto-apprentissage des opérations du microprocesseur 16 bits. Le processeur est un 8086 avec une interface mémoire 16 bits. La ROM du moniteur fournit des fonctions permettant de saisir le code des instructions 8086 à l'aide d'une touche hexadécimale intégrée, de tester l'exécution du code, d'effectuer le programme pas à pas et d'afficher les registres utilisateurs.

Le kit dispose d'un générateur d'impulsions de 10 ms pour tester le fonctionnement des interruptions externes. Il est également équipé d'un port RS232C pour une interface de terminal à 9600 bauds.
Le fichier hexadécimal généré par l'assembleur peut être téléchargé facilement.

2 - SCHÉMA FONCTIONNEL



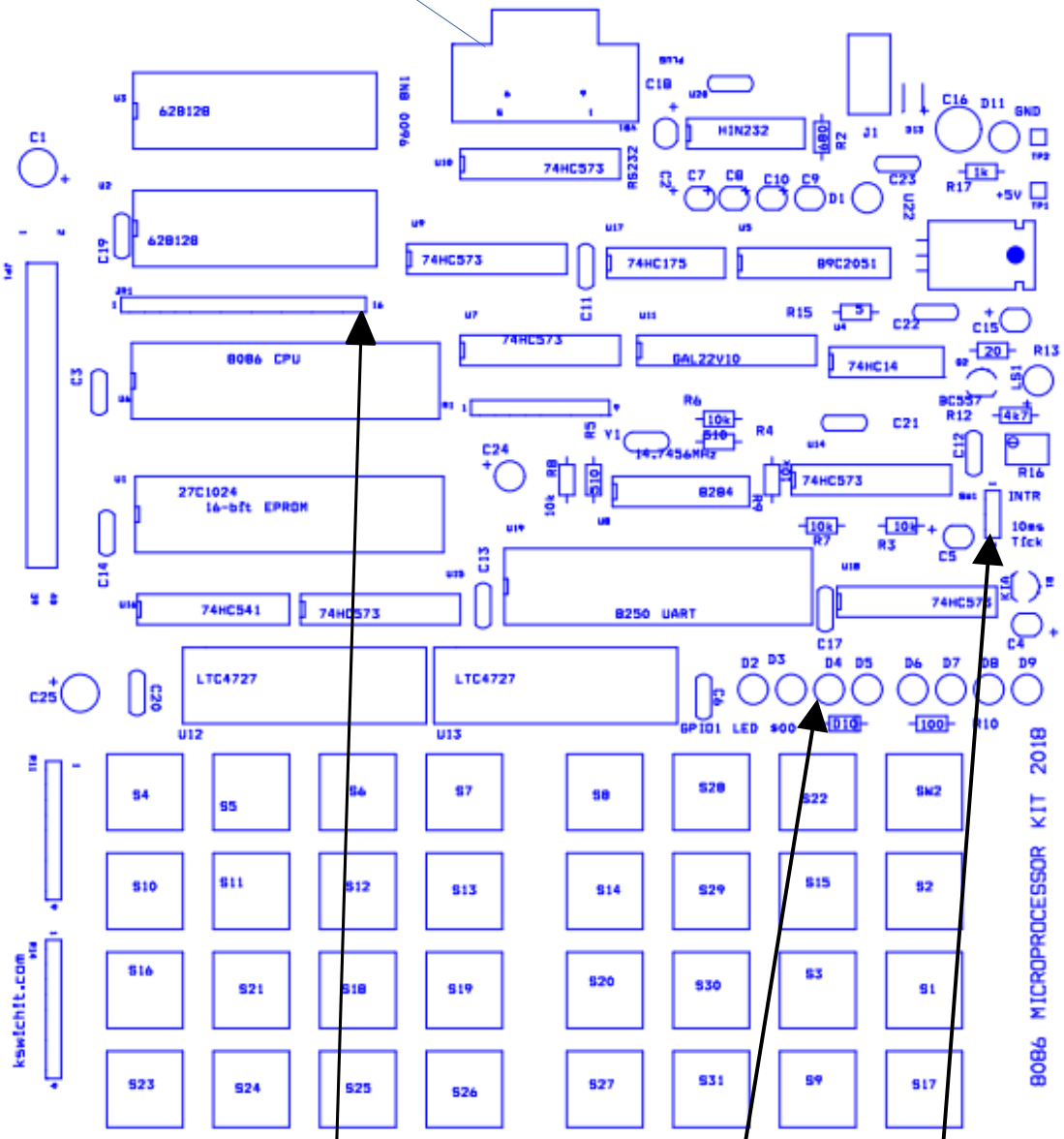
Notes

1. L'UART est réglé sur 9600 bit/s et contrôlé par la puce de communication asynchrone 8250.
2. Le kit dispose d'un bus d'interface pour module LCD 8 bits.
3. Le générateur d'impulsions à 100 Hz est destiné à l'expérimentation des interruptions.
4. Les ports pour l'affichage et l'interface du clavier ont été construits à l'aide de circuits intégrés logiques discrets.
5. Le décodeur d'adresses mémoire et de port est fabriqué à l'aide d'un Système à logique programmable (CPL).

3 - AGENCEMENT DU MATÉRIEL

Prise jack, +9Vcc

Connecteur RS232C
DB9 mâle



Connecteur LCD texte à 16 broches

DEL GPIO1, affichage binaire 8 bits (adresse 00H)

Sélecteur pour impulsions de 10 ms ou touche INTR

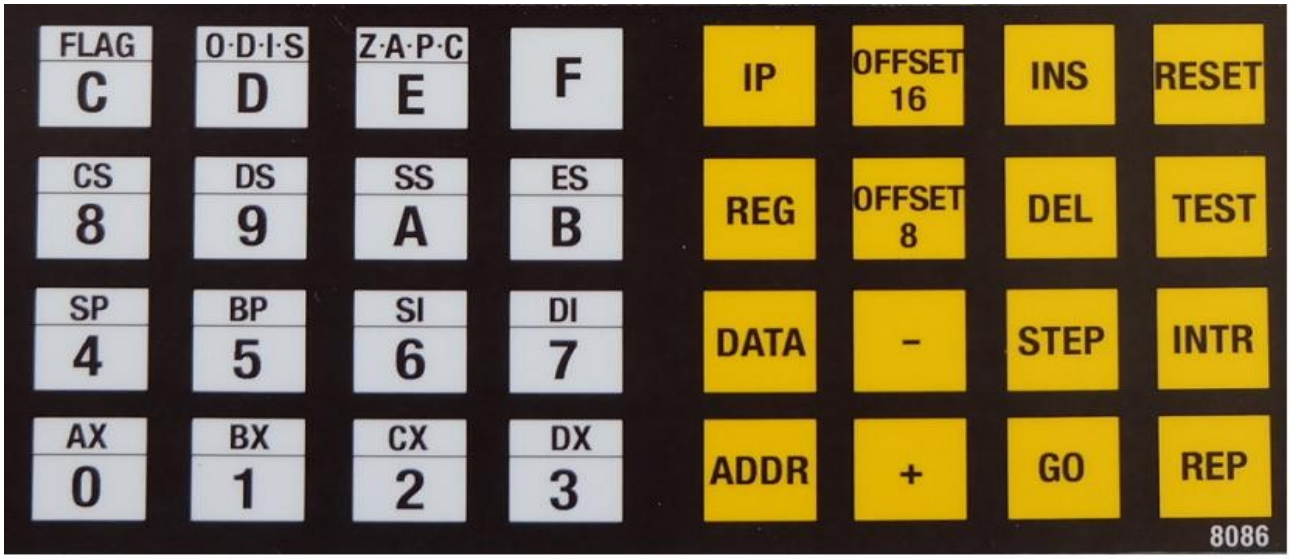
Remarques importantes

1 – L'insertion ou le retrait du module LCD doit être effectué lorsque le kit est hors tension !

2 – L'adaptateur secteur doit fournir environ +7,5 Vcc. Une tension plus élevée entraînera une surchauffe de la puce du régulateur de tension. (Une tension de +9 Vcc est acceptable.)

3 – En fonctionnement normal, le régulateur de tension sera **CHAUD**. Risque de brûlure au contact du dissipateur thermique.

4 - DISPOSITION DU CLAVIER



Touches HEXADÉCIMALES - Chiffres hexadécimaux de 0 à F avec registres utilisateur associés, AX, BX, CX, DX, SP, BP, SI, DI, CS, DS, SS, ES et affichage des indicateurs

Touches de fonction de la supervision

RESET	Réinitialisez le CPU et démarrez le programme de supervision.
TEST	Testez l'interruption de période 10 ms et le haut-parleur. SW1 doit être en position 10 ms.
INTR	Signal d'interruption externe. Les vecteurs d'interruption doivent être saisis par l'utilisateur.
REP	Répétez les touches de fonction lorsque vous appuyez dessus simultanément.
REG	Définir les registres utilisateurs affichés. À utiliser avec la touche hexadécimale pour un registre utilisateur donné. La touche D affiche les indicateurs O D I S (débordement, direction, interruption, signe). La touche E affiche les indicateurs Z A P C (zéro, auxiliaire, parité, report).
IP	Affichage l'adresse 400 après la réinitialisation, puis sur l'adresse IP actuelle après le point d'arrêt de service.
INS	Insérer un octet et décaler 512 octets vers le bas.
DEL	supprimer un octet et décaler 512 octets vers le haut.
STEP	En mode d'exécution pas à pas, les registres du processeur seront enregistrés dans les registres utilisateurs.
GO	Passage du programme de supervision au code utilisateur.
OFFSET8	Calculez l'octet de décalage 8 bits, utilisé avec les touches + et GO. Entrez l'adresse de départ, la destination avec la touche +, la touche GO calculera le décalage et écrira l'octet de décalage.
OFFSET16	Calculez le décalage 16 bits de l'octet, utilisé avec les touches + et GO. Entrez l'adresse de départ, la destination avec la touche +, la touche GO calculera le décalage et écrira le mot de décalage.

5 - CARACTÉRISTIQUES MATÉRIELLES

Caractéristiques matérielles :

Processeur : Microprocesseur 8086 - 16 bits - Compatible Intel x86
Oscillateur : Générateur d'horloge 8284, Quartz 14,7456 MHz (horloge du processeur 4,9152 MHz)
Mémoire : 256 Ko de RAM, 256 Ko d'EPROM
Puce de décodage mémoire et E/S : Dispositif logique programmable GAL22V10
Affichage : LED haute luminosité à 6 chiffres et 7 segments
Clavier : 32 touches
Port RS232 : UART 8250 9600 bits/s 8n1
Débogage : LED GPIO1 8 bits à l'adresse 00H
Signal périodique : Signal de période 10 millisecondes générées par le 89C2051 pour une simulation de déclenchement temporel de 1 seconde à l'état bas et 9 secondes à l'état haut
Interface LCD texte : Interface directe avec le bus du processeur pour écran LCD texte
Perte de tension : Puce de réinitialisation KIA7042 pour la réinitialisation en cas de baisse de tension
Consommation électrique : 500 mA à 9,0 V CC avec adaptateur secteur
Connecteur d'extension : Connecteur à 40 broches au format 2x20 espacement 2,54 mm
Dimensions mécaniques : 178 x 184 mm

6 - CARACTÉRISTIQUES DU PROGRAMME DE SUPERVISION

Caractéristiques du programme de supervision :

Entrez les instructions 8086 directement à l'aide du clavier hexadécimal.
Code de test s'exécutant en pas à pas ou par déclenchement sur point d'arrêt
Capture du statut des registres utilisateur
Insérer/supprimer un octet
Calcul du décalage 8/16 bits
Téléchargement de fichiers hexadécimaux Intel via un port RS232 à 9600 bits/s.

7 - MÉMOIRE ET PLAGES D'ENTRÉE/SORTIE

Le kit fournit 256 Ko de RAM à partir de l'emplacement 00000H - 3FFFFH et une ROM de surveillance à partir de E0000H - FFFFFH.

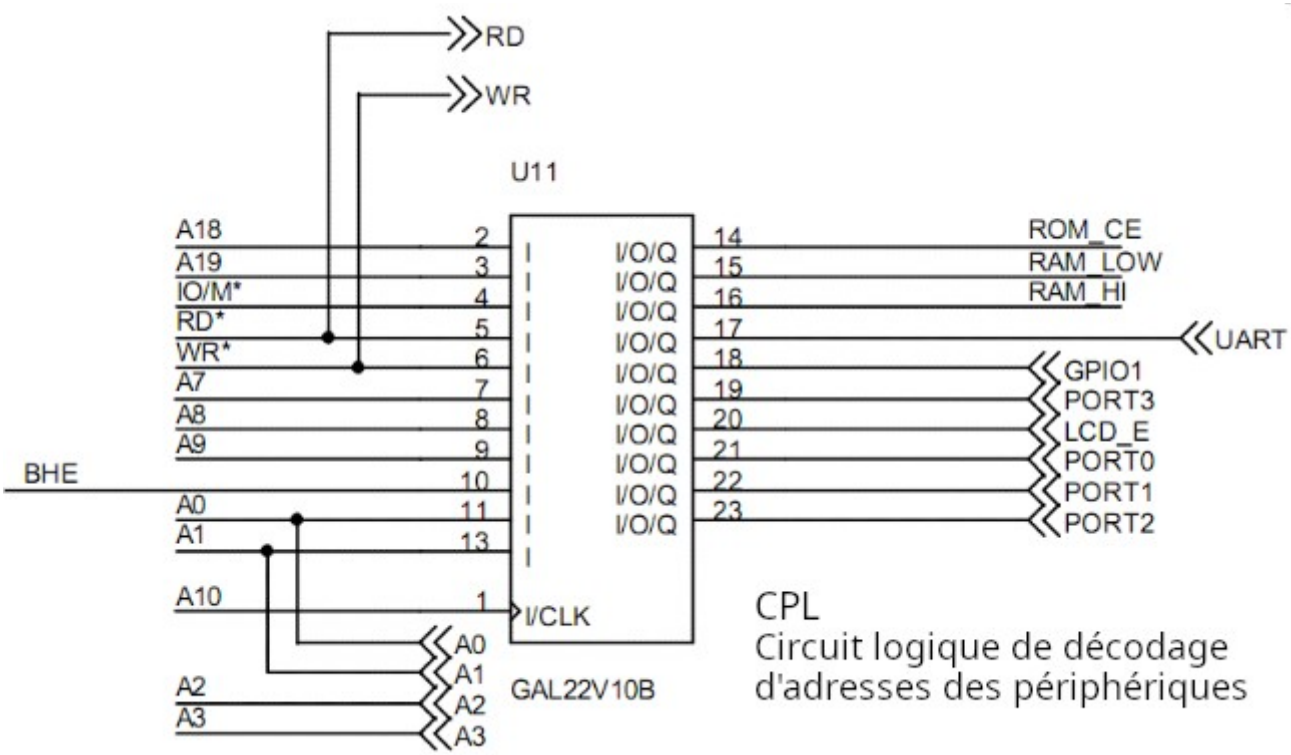
Les ports d'E/S sont situés entre 000H et 5FFH.

Plan d’adressage mémoire et dispositifs d’Entrées / Sorties

Adresses	Désignation
00000H - 3FFFFH	256kB RAM
E0000H - FFFFFH	Moniteur ROM
00H	GPIO1 LED
100H	PORT0 - Port en entrée (Clavier)
180H	PORT1 - Port en sortie (Afficheurs 7 segments - afficheur)
200H	PORT2 - Port en sortie (Afficheurs 7 segments - segments)
280H	PORT3 - 74HC175 (Haut parleur)
300H	UART 8250 - Adresse de base
400H	Connecteur d’extensions
500H - 506H	LCD registres LCD commande d’écriture (500H) LCD Écriture des données (502H) LCD Lecture des commandes (504H) LCD Lecture des données (506H)

8 - DÉCODEUR CPL

Les emplacements des mémoires et des périphériques d'E/S sont sélectionnés par le dispositif logique programmable, la puce CPL.



La puce CPL accepte des signaux d'entrée, effectue des opérations logiques et produit le signal d'activation pour chaque périphérique.

La partie gauche du GAL22V10B représente les signaux logiques d'entrée. Ces signaux se composent de signal d'adresse, des signaux de contrôle de la mémoire et des E/S.

La partie droite du GAL22V10B représente les signaux de sortie pour la sélection de la mémoire morte, de la mémoire vive et de chaque périphérique d'E/S.

Pour la sélection de la ROM, une seule ligne est nécessaire. Cependant, pour la RAM avec deux puces séparées, les signaux octets de poids fort et octets de poids faible sont nécessaires.

Le fichier pour la puce CPL ci-dessous est compilé avec la version 5.30.4 de WinCupL.

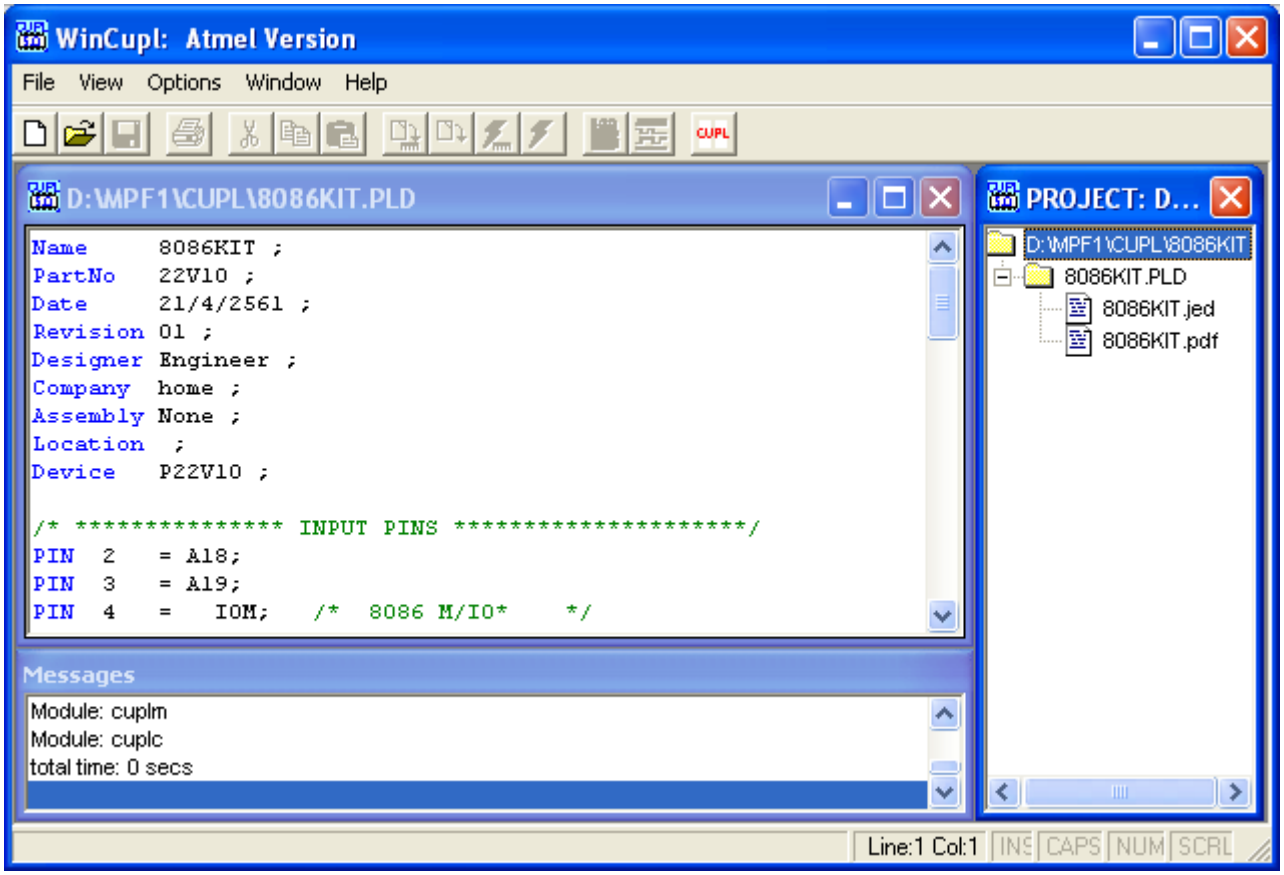
```
Name     8086KIT ;
PartNo   22V10 ;
Date     21/4/2561 ;
Revision 01 ;
Designer Engineer ;
Company  home ;
Assembly None ;
Location  ;
Device   P22V10 ;

/* *************** INPUT PINS *********************/
PIN 2 = A18;
PIN 3 = A19;
PIN 4 = IOM; /* 8086 M/IO* */
PIN 5 = RD;
PIN 6 = WR;
PIN 7 = A7;
PIN 8 = A8;
PIN 9 = A9;
PIN 10 = BHE;
PIN 11 = A0;
PIN 13 = A1;
PIN 1 = A10;

/* *************** OUTPUT PINS *********************/
PIN 14 = ROMCE;
PIN 15 = RAM_LOW;
PIN 16 = RAM_HI;
PIN 17 = UART;
PIN 18 = GPIO1;
PIN 19 = PORT3;
PIN 20 = LCD_E;
PIN 21 = PORT0;
PIN 22 = PORT1;
PIN 23 = PORT2;

ROMCE = !IOM # !A18 # !A19;
RAM_LOW = !IOM # A18 # A19 # A0;
RAM_HI = !IOM # A18 # A19 # BHE;
!GPIO1 = WR # IOM # A10 # A9 # A8 # A7 # A0;
PORT0 = RD # IOM # A10 # A9 # !A8 # A7 # A0;
!PORT1 = WR # IOM # A10 # A9 # !A8 # !A7 # A0;
!PORT2 = WR # IOM # A10 # !A9 # A8 # A7 # A0;
PORT3 = WR # IOM # A10 # !A9 # A8 # !A7 # A0;
UART = IOM # A10 # !A9 # !A8 # A7 # A0;
!LCD_E = (RD & WR) # IOM # !A10 # A9 # !A8 # A7;
```

Note : # OU logique, & ET logique, ! NON logique



Nous pouvons éditer l'équation CPL en utilisant directement l'IDE WinCupL.

Lors de la compilation, le fichier JEDEC, 8086KIT.jed sera produit.

9 - COMMENT DÉMARRER

Le kit accepte une alimentation en courant continu avec une tension minimale de +7,5V. Il consomme un courant continu d'environ 500mA. Cependant, il est possible d'utiliser une tension de +9VDC à partir de n'importe quel adaptateur CA. L'exemple d'un adaptateur CA est illustré ci-dessous.

La broche centrale est le potentiel positif (+). La broche extérieure est la masse (- ou MASSE).

La chute de tension au niveau du 7805 est d'environ +2 V. Pour obtenir +5 Vcc pour le kit, nous avons donc besoin d'une tension d'entrée **supérieur** à +7 Vcc.

Si la tension de sortie de votre adaptateur est réglable, choisissez une tension d'environ +7,5 Vcc. Cette tension évitera de faire surchauffer le régulateur LM7805.

Si le kit subit des déconnexions, Utilisez un transformateur avec une tension de 9 Vcc.

Une tension plus élevée entraînera une perte de puissance thermique plus importante au niveau du régulateur de tension 7805 et pourrait réduire sa durée de vie.

Lors de la mise sous tension, le message 8086 s’affiche au milieu des afficheurs 7 segments.

Appuyez sur la touche IP, l'écran affiche l'adresse de début du programme qui se trouve à l'adresse 0400.

Le champ de données affichera la donnée présente à cette adresse.

La touche + permet d'incrémenter l'adresse.

10 - COMMENT ENTRER UN PROGRAMME EN UTILISANT LE CODE HEXADECIMAL

Essayons d'entrer les instructions 8086 en utilisant le code hexadécimal ou le code machine.

Pour obtenir les codes hexadécimaux, il faudra utiliser un assembleur pour 8086 qui transformera les mnémoniques en code hexadécimal.

Adresses	Code Hexa	Étiquettes	Instructions	Commentaires
0400	B0 01	MAIN	MOV AL,1	Charge AL avec la valeur 1
0402	E6 00		OUT 0,AL	Transfert AL sur le port GPIO1 @00
0404	CC		INT 3	Retour au moniteur adresse @000C

Notre programme de test ne comporte que trois instructions.

La première instruction est : `MOV AL,1`
Charger le registre AL avec la valeur 01 d'une taille de 8 bits.
Cette instruction comporte deux octets de code hexadécimal, à savoir B0 et 01. B0 correspond à l'instruction « MOV AL,n » et 01 correspond à la variable « n ».

La deuxième instruction est : `OUT 0,AL`
Copie du registre AL sur le port de sortie GPIO1 qui correspond aux 8 DEL situées à l'adresse 00.
Le code machine de l'instruction est E6. L'emplacement de GPIO1 est à l'adresse 00.

La dernière instruction est INT 3, code hexadécimal CC C'est une interruption qui permet de revenir au programme de supervision à l'adresse 000C.

Ce programme de test ne comporte que 5 octets, B0, 01, E6, 00, CC.

Le premier octet sera entré à l'emplacement 0400, les suivants aux adresses 0401, 0402, 0403 et 0404.

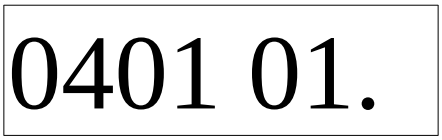
Voyons comment entrer ces codes dans la mémoire.

Étape 1 - Appuyez sur la touche RESET, puis sur la touche IP. L'écran affiche l'adresse de la mémoire actuelle et son contenu.

Ci-contre, l'emplacement 0400 contient la valeur hexadécimale 2F. Un petit point sur le champ de données indique que le champ actif est prêt à être modifié par une nouvelle valeur hexadécimale.

Étape 2 - appuyez sur la touche B, puis sur la touche 0. Le nouveau code hexadécimal B0 sera alors saisi à l'emplacement 0400.

Étape 3 - Appuyez sur la touche + pour incrémenter l'adresse de 0400 à 0401. Saisir sur le clavier la touche 1.



Répétez l'étape 3 jusqu'à ce que vous ayez atteint le dernier emplacement. Le code hexadécimal peut être vérifié à l'aide des touches + ou -.

Pour modifier l'emplacement de l'affichage, appuyez sur la touche ADDR. Le point se déplace vers le champ Adresse. Toute pression sur une touche hexadécimale modifie l'adresse affichée.

11 - AFFICHAGE DES REGISTRES D'UTILISATEURS

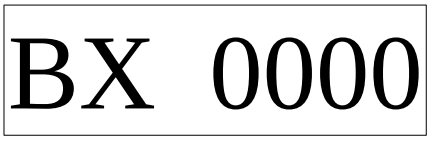
Avant de tester l'exécution du code, voyons comment examiner les registres de l'utilisateur.

Les registres de l'utilisateur sont des blocs mémoire de la RAM utilisé pour sauvegarder le contenu des registres de l'unité centrale après l'exécution d'un programme donné.

Nous pouvons examiner les registres de l'utilisateur pour vérifier l'exécution de notre code.

Appuyer sur la touche REG, puis sur la touche 0, pour afficher le contenu 16 bits du registre AX (AH:AL).

Appuyer sur la touche Reg puis 1, 2, 3, pour BX, CX et DX.



12 - EXEMPLES DE CODE DE TEST

Nous pouvons tester l'exécution du code ci-dessus en appuyant sur la touche GO. La dernière instruction INT 3 renverra l'unité centrale au programme de supervision pour enregistrer les registres de l'unité centrale dans les registres de l'utilisateur, ce qui nous permettra d'examiner le fonctionnement du programme.

Adresses	Code Hexa	Étiquettes	Instructions	Commentaires
0400	B0 01	MAIN	MOV AL,1	Charge AL avec la valeur 1
0402	E6 00		OUT 0,AL	Transfert AL sur le port GPIO1 @00
0404	CC		INT 3	Return to monitor 000C

Entrons maintenant le code, lorsque nous avons terminé, appuyons sur la touche IP puis sur GO.

Que se passe-t-il ?

La diode de GPIO1 s'allumera à la position indiquée par un 1 exemple : 00000001.

Pouvez-vous changer l'affichage de 0000 0001 en une autre valeur ? Comment ?

Un autre exemple est le programme itératif de déplacement des LED.

Adresses	Code Hexa	Étiquettes	Instructions	Commentaires
0400	B0 01	START	MOV AL,1	Charge AL avec la valeur 1
0402	E6 00	LOOP	OUT 0,AL	Transfert AL sur le port GPIO1 @00
0404	DC C0		ROL AL,1	Rotate LOOP forever
040B	B9 00 30	DELAY	MOV CX,3000H	Load CX with 3000
040E	E2 FE		LOOP $	Repeat here
0410	C3		RET	Return to main

Entrons maintenant le code.

Lorsque nous avons terminé, appuyons sur la touche IP puis sur GO.

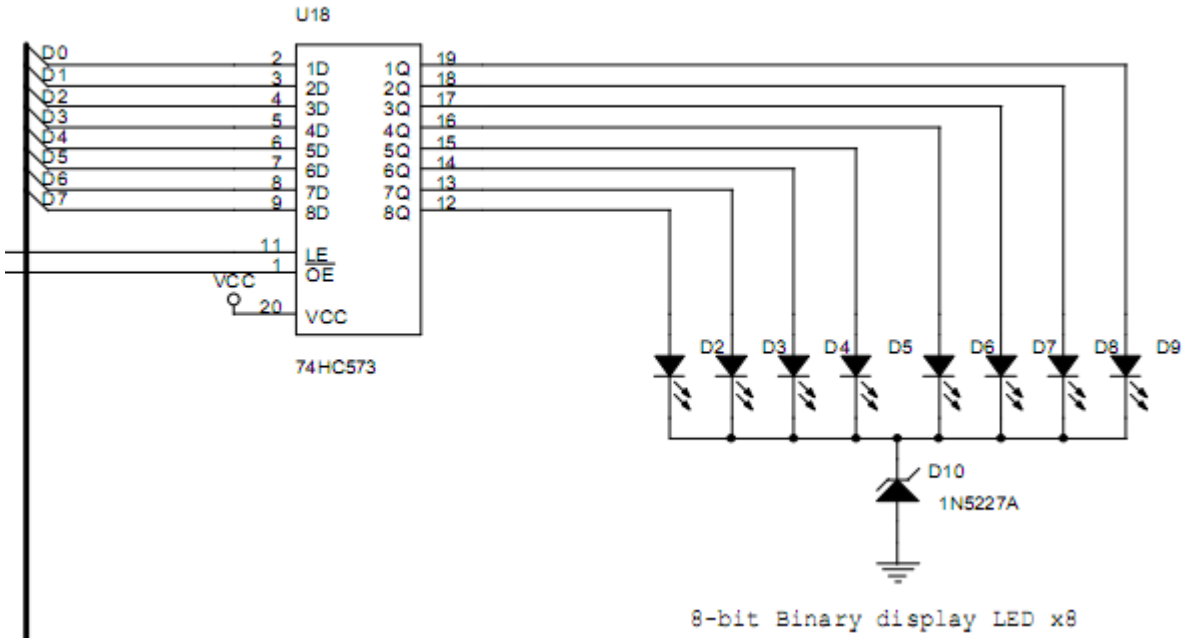
Que se passe-t-il ? Pouvez-vous modifier la vitesse de déplacement ? Comment ?

Pouvez-vous changer de direction ? Comment ?

13 - GPIO1 LED

Le kit fournit un affichage binaire utile de 8 bits. Il peut être utilisé pour déboguer le programme ou le code en cours de démonstration.

Le circuit est un CD74HC573, c'est un verrou 8 bits qui permet de conserver (verrouiller) une valeur binaire sur le port de sortie.



L'adresse du Circuit de verrouillage d'E/S est 00.

Nous pouvons utiliser l'instruction OUT 0,AL qui écrit des données de 8 bits dans AL pour les transférer sur les DELs à l'adresse 00.

Un « 1 » logique allume la DEL correspondant au bit « 1 » du code binaire. Un « 0 » logique éteint à la DEL correspondant au bit « 0 » du code binaire.

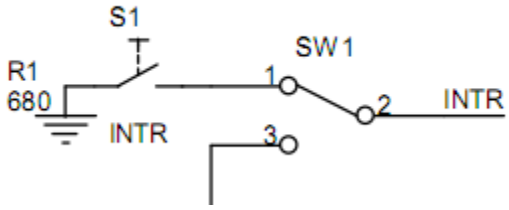
Le code hexadécimal pour OUT 0,AL est « E6 00 ». Seulement deux octets et facile à retenir !

Chaque fois que nous avons besoin de vérifier le contenu d'une variable de 8 bits, nous pouvons le copier dans le registre AL puis utiliser l'instruction OUT pour transférer facilement la valeur sur les LED du GPIO1.

14 - GÉNÉRATEUR D'IMPULSION 10 ms

SW1 est un sélecteur de source d'interruption entre la touche INTR et l'impulsion de 10 ms produit par le microcontrôleur 89C2051.

Le générateur d'impulsion est contrôlé par logiciel en utilisant l'interruption du timer0 du microcontrôleur 89C2051. Le signal d'impulsion actif au niveau bas est envoyé au travers de la broche P3.7. Le clignotement de la LED D1 indique que l'impulsion est active.



TICK est un signal périodique de 10 ms pour déclencher la broche INTR du 8086. Lorsque SW1 est positionné sur la position TICK, le CPU 8086 peut être déclenché par une interruption externe. L'impulsion de 100 Hz ou l'impulsion de 10 ms peuvent être utilisées pour produire des tâches qui s'exécutent avec un multiple de la période d'impulsion.

NOTE : La touche TEST, utilisera l'impulsion de 10 ms pour effectuer un comptage binaire à une fréquence de 100 Hz.

15 - EXEMPLE D'INTERRUPTION

Cet exemple de code met en évidence l'interruption masquable du 8086 en utilisant l'impulsion de 10 ms.

La résistance de tirage optionnel R1 du bus de données fournit une simple interruption de type numérique FF lorsque la puce 8086 accuse réception de l'INTR déclenché.

Le kit reçoit donc les vecteurs des emplacements 03FC (IP) et 03FE (CS).

Adresses	Code Hexa	Étiquettes	Instructions	Commentaires
03FC			ORG 3FCH	; INTERRUPTION FF
03FC	00		DFB SERVICE_INTERRUPT&0FFH	; IP = 0500
03FD	05		DFB SERVICE_INTERRUPT>>8	
03FE	00		DFB 0	; CS = 0000
03FF	00		DFB 0	
0400			ORG 400H	
0400	FB	MAIN	STI	; ENABLE INTERRUPT
0401	EB FE		JMP $	; JMP HERE
0500			ORG 500H	
0500		SERVICE_INTERRUPT		
0500	FE C4		INC AH	
0502	80 FC 64		CMP AH,100	
0505	75 06		JNZ SKIP	
0507	B4 00		MOV AH,0	
0509	FE C0		INC AL	
050B	E6 00		OUT 0,AL	
050D	CF	SKIP	IRET	

Nous voyons que les vecteurs de service d'interruption 0500 sont stockés à 03FC et 03FD. Le registre de segment de code, CS 0000, est stocké à 03FE et 03FF.

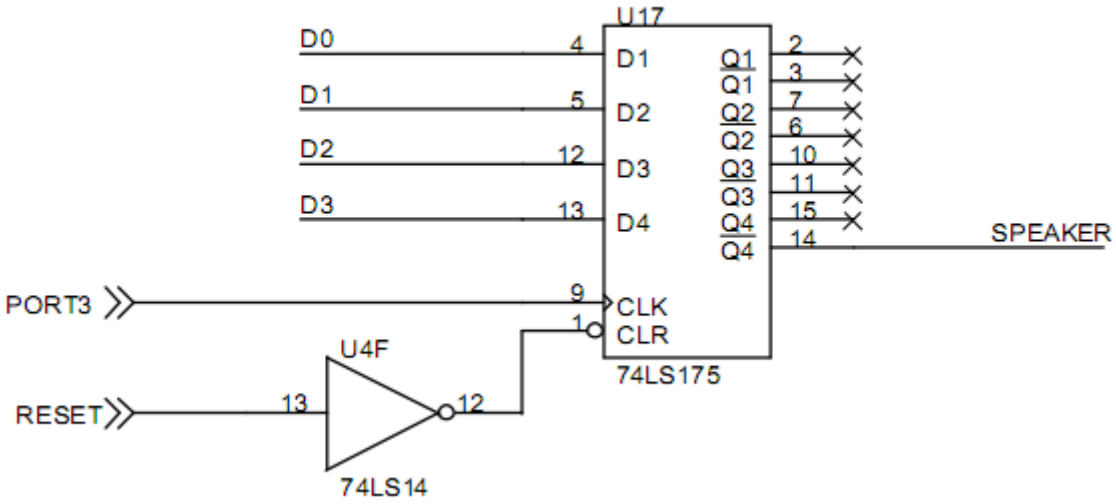
Dans le programme principal, il suffit d'activer l'interruption avec l'instruction STI. Ensuite, l'instruction « JMP » boucle sur sa propre adresse ($), en attendant l'interruption.

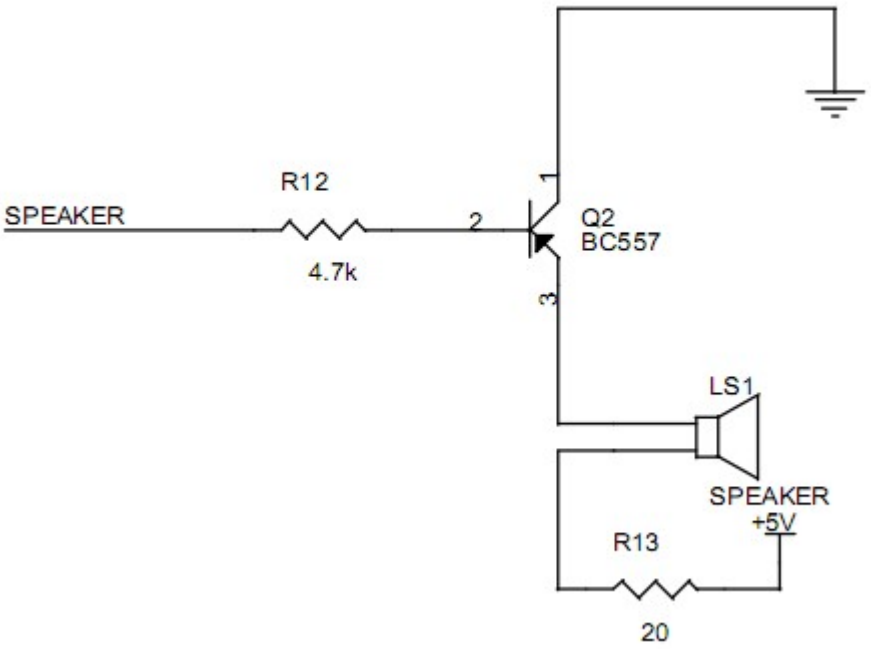
La routine de service utilise AH pour le comptage de 100 cycles. AL est incrémenté toutes les secondes. Nous voyons facilement le comptage sur la LED GPIO1.

Entrons le code hexadécimal et effectuons un test. Que se passe-t-il ? Pouvez-vous modifier le taux de comptage ? Comment ?

16 - PORT POUR HAUT-PARLEUR

Le kit est équipé d'un petit haut-parleur alimenté par le bit de sortie numérique 3 du PORT3. Nous pouvons produire un signal de tonalité en pulsant ce bit, ON et OFF. Le transistor PNP à petit signal alimente alors la charge du haut-parleur en produisant une tonalité.





Adresses	Code Hexa	Étiquettes	Instructions	Commentaires
0400			ORG 400H	
0400	BA 80 02		START MOV DX,PORT3	; charge DX avec 280
0403	B0 00	TONE	MOV AL,0	; charge AL avec 0
0405	EE		OUT DX,AL	; Ecrit AL sur PORT3
0406	E8 08 00		CALL DELAY	; call delay
0409	B0 08		MOV AL,8	; load AL with 8
040B	EE		OUT DX,AL	; write AL to PORT3
040F	EB F2		JMP TONE	; repeat forever
0411	B9 50 00	DELAY	MOV CX,50H	; load CX with 50
0414	E2 FE		LOOP $	; repeat here
0416	C3		RET	; return

Le programme ci-dessus produit une tonalité.

Entrons le code hexadécimal dans la mémoire, de l'adresse 400 à 416.

Appuyez sur IP, puis sur GO.

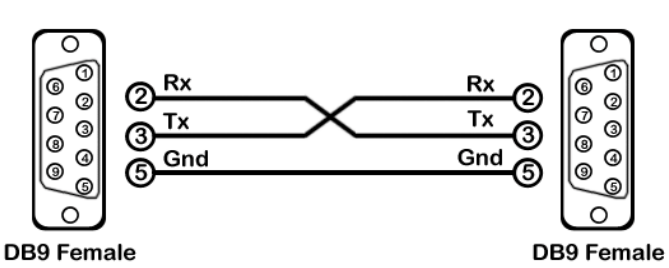
Que se passe-t-il ? Pouvez-vous augmenter ou diminuer la fréquence ? Comment ?

17 - PORT RS232C

Le port RS232C est destiné à la communication en série. On peut utiliser un câble croisé ou un câble NULL MODEM pour relier le kit au terminal. Les connecteurs des deux côtés sont des DB9 femelles.

Il est possible de le construire ou de l'acheter dans des magasins d'informatique.

—

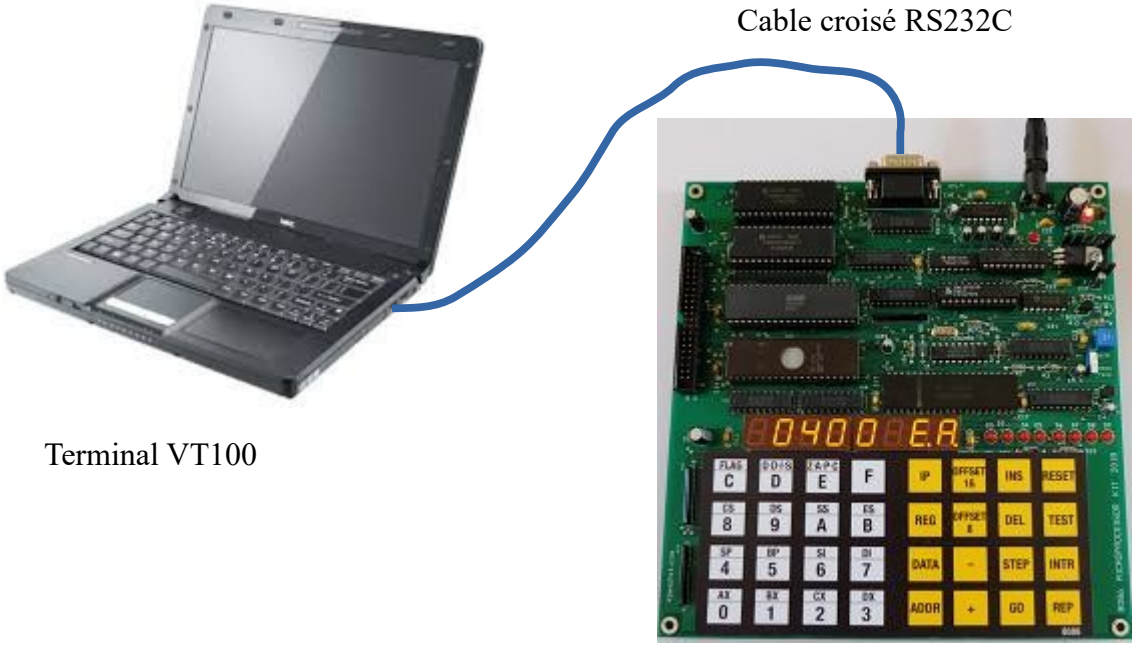


—

Pour les nouveaux PC ou ordinateurs portables sans port RS232. Il faut utiliser un convertisseur USB vers RS232 pour avoir le port RS232C.

18 - CONNEXION DU KIT 8086 AU TERMINAL

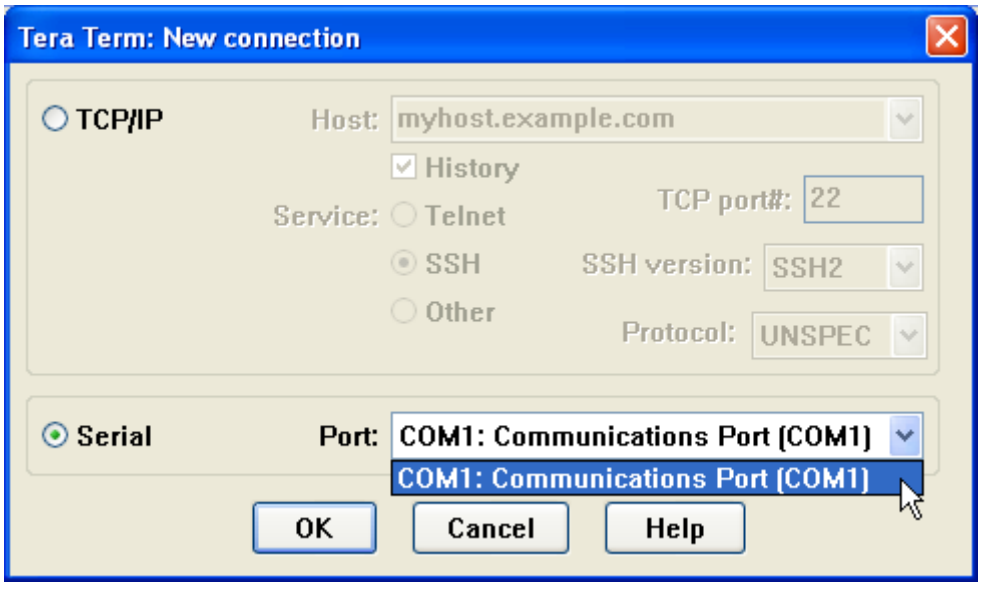
Nous pouvons connecter le kit 8086 à un terminal à l'aide d'un câble croisé RS232C.
Vous pouvez télécharger le programme de terminal gratuit, teraterm, à partir de l'URL suivante :
https://sourceforge.net/projects/tera-term/



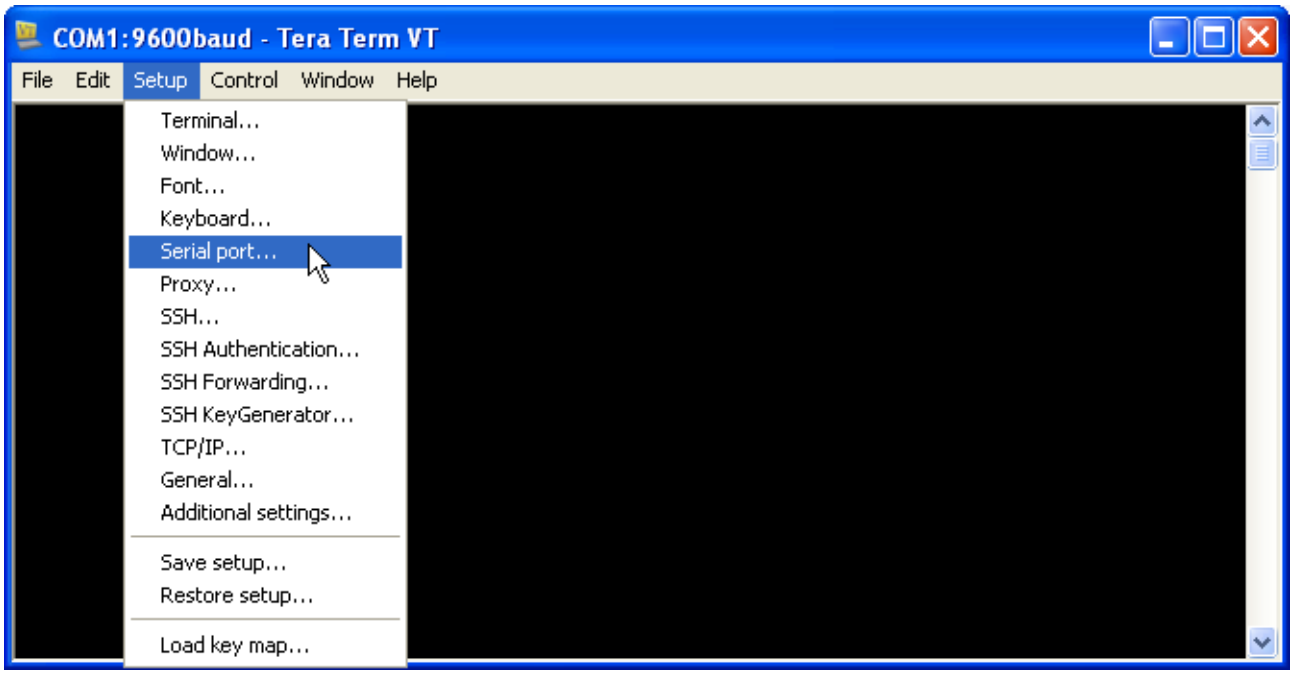
L'exemple montre la connexion d'un ordinateur portable avec le port COM1 au port RS232C du kit 8086. Pour un nouvel ordinateur portable sans port COM, nous pouvons utiliser l'adaptateur USB-RS232 pour convertir le port USB en port RS232.

Pour télécharger le fichier Intel « .hex » généré par l'assembleur ou le compilateur C, réglez la vitesse du port série sur 9600 bits/s, 8 bits de données, pas de parité, pas de contrôle de flux, un bit d'arrêt.

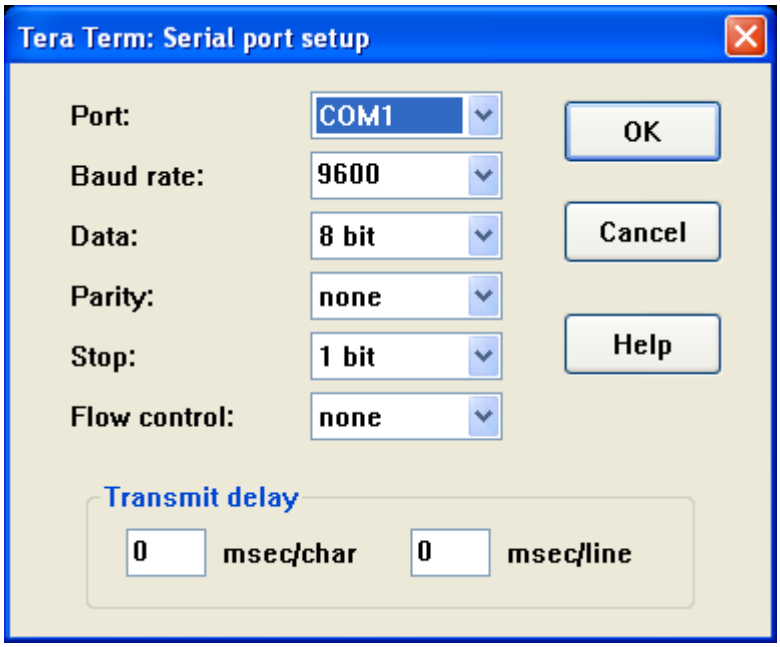
Étape 1 - Lancer Tera Term, puis cliquer sur Connexion série



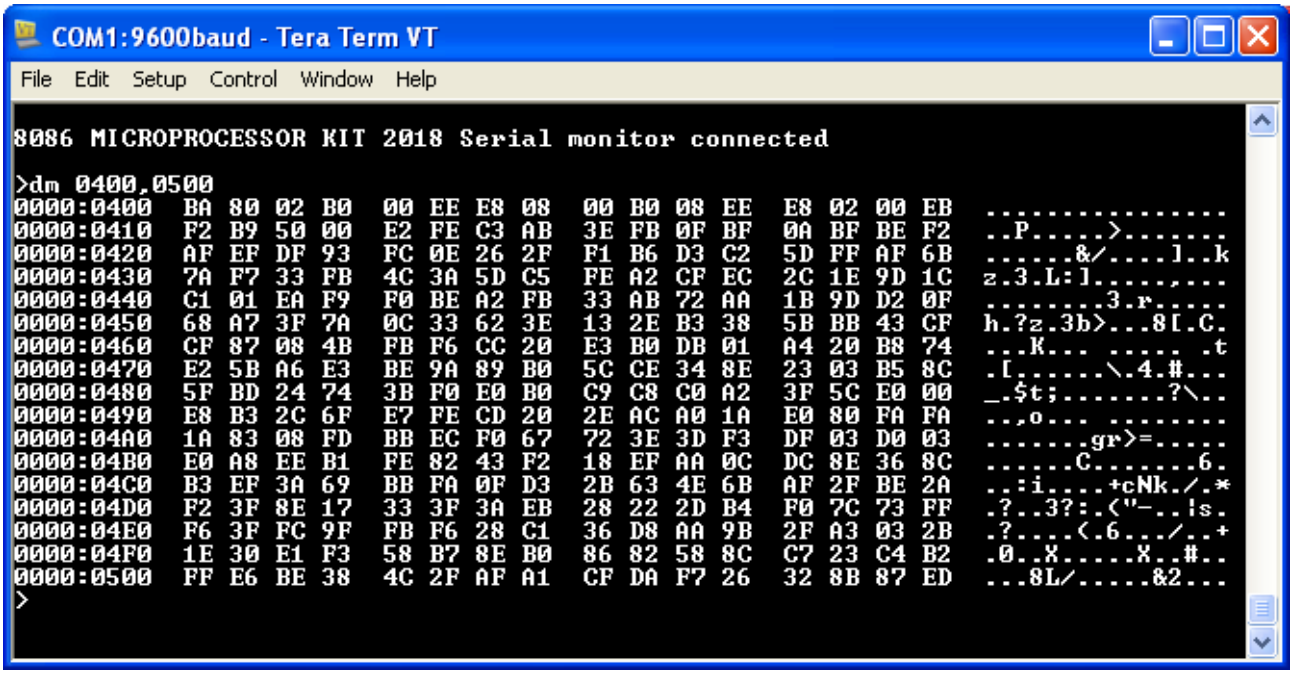
Étape 2 - Cliquez sur setup>Serial port.



Étape 3 - Régler la vitesse du port série sur 9600 et le format comme indiqué ci-dessous.



Étape 4 - Appuyez sur la touche ENTER du terminal. Le kit connectera automatiquement le terminal.

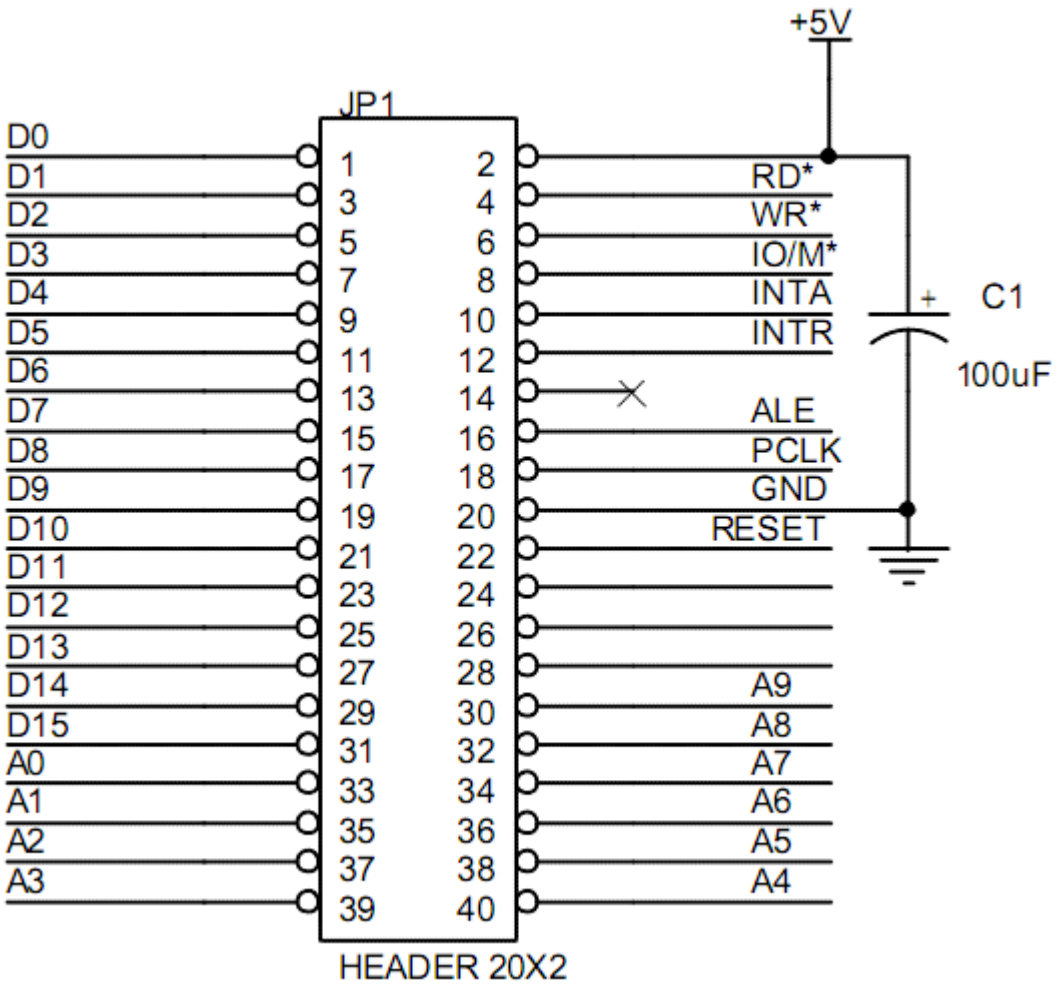


Appuyez sur la touche L pour télécharger le fichier hexadécimal. Touche ? pour accéder au menu d'aide.

Pour revenir à la supervision, appuyez sur la touche « RESET ».

19 - CONNECTEUR DE BUS D'EXTENSION

JP1, est un connecteur à 2x20 broches, il fournit des signaux du bus CPU pour l'extension ou l'interfaçage des E/S. Les étudiants peuvent apprendre à réaliser un simple port d'E/S, à interfacer un convertisseur analogique-numérique, à interfacer un moteur pas à pas ou des circuits d'alimentation en courant alternatif. Le bus de données est de 16 bits, de D0 à D15.



20 - CONNEXION DU MODULE LCD

JR1 est un connecteur à 16 broches pour la connexion du module de texte LCD. R15 est une résistance de limitation de courant pour le rétro-éclairage. R16 est un potentiomètre de réglage du contraste. Le module LCD est relié directement au bus 8086. Les registres de commande et de données sont situés dans l'espace E/S dont l'adresse est comprise entre 500H et 506H.

Registres LCD	Adresse E/S
Commande d'écriture	500H
Commande lecture	504H
Écriture de données	502H
Lecture des données	506H

Remarque : L'insertion ou le retrait du module LCD doit être effectué lorsque le kit est hors tension.

Tout écran LCD textuel avec un contrôleur compatible HD44780 peut être utilisé.

21 - ALIMENTATION DE LA SONDE LOGIQUE

Le kit fournit les points de test TP1(+5 V) et TP2(GND) pour l'utilisation de la sonde logique. Les élèves peuvent facilement apprendre les signaux logiques numériques à l'aide de la sonde logique.

La pince rouge correspond au +5 V et la pince noire à la masse.

SCHÉMA DU MATÉRIEL, LISTE DES PIÈCES

22 - LISTE DES PIÈCES

LISTE DES PIÈCES

Semi-conducteurs

U1 AM27C1024, 16-bit 256kB EPROM
U3,U2 HM628128A, 128kB SRAM
U4 74HC14, hex inverter
U5 AT89C2051, 8-bit tick generator
U6 8086, 16-bit Microprocessor
U7,U9,U10,U14,U15,U18 74HC573, 8-bit Latch
U8 8284, clock generator
U11 GAL22V10, PLD
U13,U12 LTC-4727, 7-segment LED
U16 74HC541, 8-bit tristate buffer
U17 74LS175, 4-bit latch
U19 8250, UART
U20 HIN232, RS232 voltage converter
U22 LM7805/TO_5, +5V voltage regulator

D1,D2,D3,D4,D5,D6,D7,D8, D9 LED

D10 1N5227A
D11 POWER
D13 TVS15V
Q1 KIA7042,
Q2 BC557, PNP transistor

Résistances (toutes sont 1/8W +/-5%)

R1 RESISTOR SIP 9
R2 680
R3,R6,R7,R8,R9,R16 10K
R4,R5 510
R10 100
R14,R11 10k RESISTOR SIP 9
R12 4.7k
R13 20
R15 5
R17 1k

Condensateurs

C1,C24,C25 100uF
C2,C4,C5,C8,C9,C10 10uF
C3,C6,C11 100nF
C7 10uF 10V
C12,C13,C14,C17,C18,C19, 0.1uF
C20,C21
C15 10uF 16V
C16 1000uF25V
C23,C22 0.1uF

Pièces supplémentaires

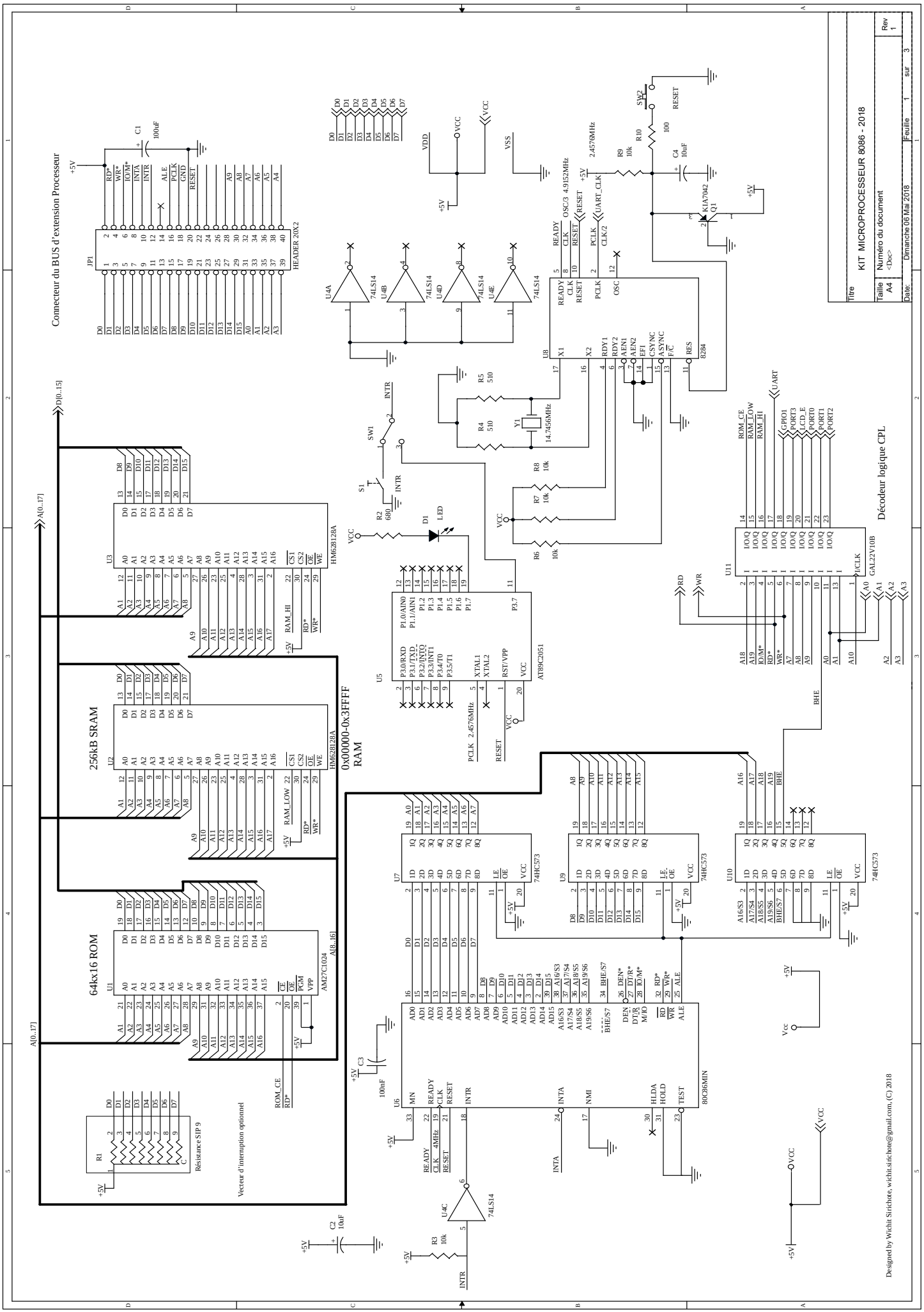
JP1 HEADER 20X2
JR1 CONN RECT 16
J1 DC Input
LS1 SPEAKER

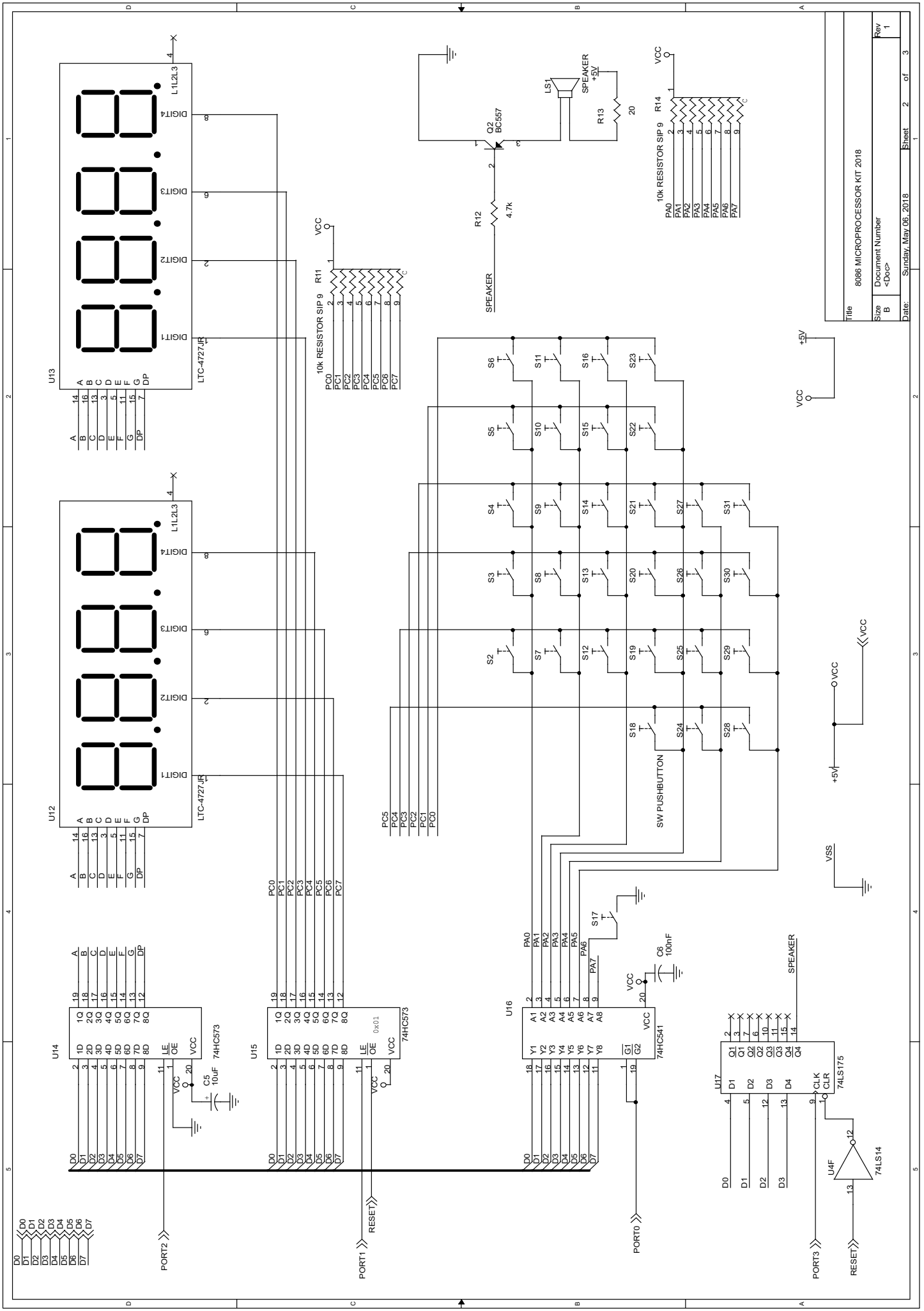
SW1 SW MAG-SPDT
SW2 RESET
S1 INTR
S2,S3,S4,S5,S6,S7,S8,S9, SW PUSHBUTTON
S10,S11,S12,S13,S14,S15,
S16,S17,S18,S19,S20,S21,
S22,S23,S24,S25,S26,S27,
S28,S29,S30,S31
TP1 +5V
TP2 GND

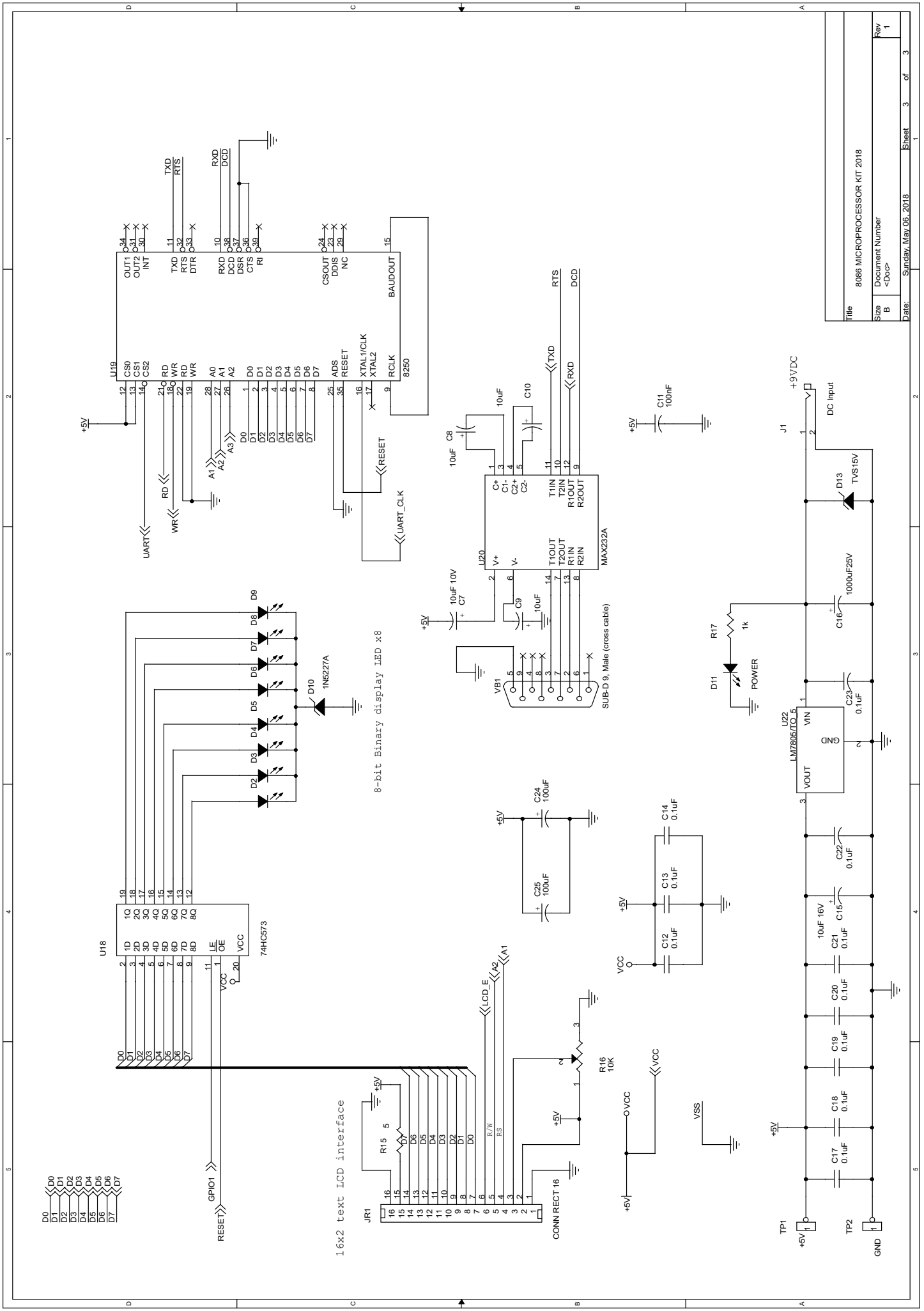
VB1 SUB-D 9, Male (cross cable)
Y1 14.7456MHz

PCB double side plate through hole
LED cover Clear RED color acrylic plastic
Autocollant clavier imprimable fichier SVG

23 - SCHÉMAS DE LA CARTE


Connecteur du BUS d'extension Processeur
JP1
HEADER 20X2
C1
100uF
RD*
WR*
IO/M*
INTA
INTR
ALE
PCLK
GND
RESET
A9
A8
A7
A6
A5
A4
D0..D7
A0..A15
VDD
VSS
VCC
+5V
U4A
U4B
U4D
U4E
74LS14
U8
8284
READY
CLK
RESET
PCLK
OSC
X1
X2
RDY1
RDY2
AEN1
AEN2
EFI
CSYNC
ASYNC
F/C
RES
OSC/3 4.9152MHz
2.4576MHz
UART_CLK
CLK/2
R9
10k
R10
100
C4
10uF
SW2
RESET
Q1
KIA7042
R5
510
R4
510
Y1
14.7456MHz
SW1
INTR
S1
R2
680
D1
LED
R8
10k
R7
10k
R6
10k
U3
HM628128A
256kB SRAM
U2
HM628128A
0x00000-0x3FFFF
RAM
U5
AT89C2051
P1.0/AIN0
P1.1/AIN1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.7
XTAL1
XTAL2
RST/VPP
PCLK 2.4576MHz
RESET
RAM_HI
RAM_LOW
RD*
WR*
CS1
CS2
OE
WE
U1
64kx16 ROM
AM27C1024
A18..16
CE
OE
PGM
VPP
ROM_CE
U11
GAL22V10B
I/O/Q
I/CLK
A0
A1
A2
A3
A10
A18
A19
IO/M*
BHE
ROM_CE
RAM_LOW
RAM_HI
UART
GPIO1
PORT3
LCD_E
PORT0
PORT1
PORT2
Décodeur logique CPL
U7
U9
U10
74HC573
LE
OE
A8
A9
A10
A11
A12
A13
A14
A15
A16
A17
A18
A19
BHE
A16/S3
A17/S4
A18/S5
A19/S6
BHE/S7
U6
80C86MIN
MN
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
AD8
AD9
AD10
AD11
AD12
AD13
AD14
AD15
DEN*
DT/R*
M/IO
RD*
WR*
ALE
READY
CLK 4MHz
RESET
INTR
INTA
NMI
HLDA
HOLD
TEST
C3
100nF
C2
10uF
R3
10k
U4C
74LS14
R1
Résistance SIP 9
Vecteur d'interruption optionnel
Titre
KIT MICROPROCESSEUR 8086 - 2018
Taille
A4
Numéro du document
<Doc>
Rev
1
Date:
Dimanche 06 Mai 2018
Feuille
1
sur
3
Designed by Wichit Stitchote, wichit.stitchote@gmail.com, (C) 2018


8086 MICROPROCESSOR KIT 2018
Sunday, May 06, 2018
Document Number
<Doc>
U12
U13
U14
U15
U16
U17
LTC-4727JR
74HC573
74HC541
74LS175
74LS14
SW PUSHBUTTON
SPEAKER
Q2 BC557
R12 4.7k
R13 20
10k RESISTOR SIP 9
PORT0
PORT1
PORT2
PORT3
RESET


8086 MICROPROCESSOR KIT 2018
Document Number
<Doc>
Sunday, May 06, 2018
8-bit Binary display LED x8
16x2 text LCD interface
SUB-D 9, Male (cross cable)
DC Input
+9VDC
MAX232A
74HC573
CONN RECT 16
UART
UART_CLK
RESET
GPIO1
POWER
TVS15V
1N5227A
LM7805/TO_5

LISTES DES COMMANDES DU PROGRAMME SUPERVISEUR

```
 1  / MON86.C
 2  // Monitor program developed with Micro-c for 8086
 3  // written by Wichit Sirichote, wichit.sirichote@gmail.com
 4  // 13 May 2018
 5
 6  #include <8086io.h>
 7
 8  #define port0 0x100           // input port keypad
 9  #define port1 0x180           //DIGIT CONTROL
10  #define port2 0x200           //SEGMENT
11  #define port3 0x280           // SPEAKER BIT3
12
13  #define user_io 0x400       // user expansion i/o ports
14
15  #define UART 0x300       // 8250 UART
16
17  #define gpio1 0               // debug LED
18
19
20  #define BUSY 0x80
21
22  #define LCD_command_write 0x500
23  #define LCD_command_read 0x504
24  #define LCD_data_write 0x502
25  #define LCD_data_read 0x506
26
27
28  char convert[16]= {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,
0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};
29
30  //char convert[16];
31
32  char buffer[8];             // display buffer
33  char buffer2[128];
34
35  unsigned char n;
36  int j;
37  unsigned char k;
38  char i,u,q,o,key;
39  unsigned int temp,temp2;
40  int x;
41  char hit;
42
43  char auto_key;
44
45  int temp16;
46  int PC, save_PC, display_PC;
47
```

```
48  int start,end,destination;
49
50  char state;
51
52  char *dptr;
53
54
55  int user_flag, save_stack;
56  int user_ip;
57  int user_cs;
58  int user_ds;
59  int user_es;
60  int user_ss;
61  int user_sp;
62
63
64  int user_ax;
65  int user_bx;
66  int user_cx;
67  int user_dx;
68  int user_bp;
69  int user_si;
70  int user_di;
71
72  int flag_low,flag_high;
73
74
75  //------------------- lcd drivers -----------------------------------
76
77  LcdReady()
78  {
79   while(in(LCD_command_read)&BUSY)
80   continue; // wait until busy flag =0
81  }
82
83  void clr_screen(void)
84  {
85   LcdReady();
86   out(LCD_command_write,0x01);
87  }
88
89
90  // if LCD connected, return 0, else return !=0
91  char check_LCD()
92  148  {
93    i=0;
94    while((in(LCD_command_read)&BUSY!=0) && ++i<100);
```

```
 95
 96        return (in(LCD_command_read)&BUSY);
 97  }
 98
 99  goto_xy(char x,char y)
100  {
101   LcdReady();
102   switch(y){
103    case 0 : out(LCD_command_write,0x80+x); break;
104    case 1 : out(LCD_command_write,0xC0+x); break;
105    case 2 : out(LCD_command_write,0x94+x); break;
106    case 3 : out(LCD_command_write,0xd4+x); break;
107   }
108  }
109  InitLcd()
110  {
111   LcdReady();
112   out(LCD_command_write,0x38);
113   LcdReady();
114   out(LCD_command_write,0x0c);
115   clr_screen();
116   goto_xy(0,0);
117  }
118
119  char *Puts(char *str)
120  {
121   unsigned char i;
122   for (i=0; str[i] != '\0'; i++){
123    LcdReady();
124    out(LCD_data_write,str[i]);
125   }
126   return str;
127  }
128
129  void putch_lcd(char ch)
130  {
131   LcdReady();
132   out(LCD_data_write,ch);
133  }
134
135  void LCDWriteText(char *txt) {
136   while(*txt)
137    putch_lcd(*txt++);
138  }
139
140  void LCDWriteConstText(const char *txt) {
141   while(*txt)
```

```
142     putch_lcd(*txt++);
143   }
144
145
146   //-----------------------------------------------------------------
147
148   dot_address()
149   {
150    buffer[3]=buffer[3]|0x80;
151    buffer[4]=buffer[4]|0x80;
152    buffer[5]=buffer[5]|0x80;
153    buffer[6]=buffer[6]|0x80;
154
155    buffer[7]= 0;
156
157    buffer[0]=buffer[0]&~0x80;
158    buffer[1]=buffer[1]&~0x80;
159    buffer[2]=0;          // could be 1MB physical location
160    //buffer[3]=0;
161   }
162
163
164   dot_data()
165   {
166
167    buffer[0]=buffer[0]|0x80;
168    buffer[1]=buffer[1]|0x80;
169    buffer[2]=0;
170   // buffer[3]=0;
171
172    buffer[3]=buffer[3]&~0x80;
173    buffer[4]=buffer[4]&~0x80;
174    buffer[5]=buffer[5]&~0x80;
175    buffer[6]=buffer[6]&~0x80;
176    buffer[7]=0;
177
178   }
179
180
181
182   hex4(int h)
183   {
184    temp16 = h;
185    buffer[3]= convert[temp16&0xf];
186    temp16>>=4;
187    buffer[4]= convert[temp16&0xf];
188    temp16>>=4;
```

```
189   buffer[5]=convert[temp16&0xf];
190   temp16>>=4;
191   buffer[6]=convert[temp16&0xf];
192
193   buffer[7]=0;
194  }
195
196  hex4low(int h)
197  {
198   temp16=h;
199   buffer[0]= convert[temp16&0xf];
200   temp16>>=4;
201   buffer[1]= convert[temp16&0xf];
202   temp16>>=4;
203   buffer[2]=convert[temp16&0xf];
204   temp16>>=4;
205   buffer[3]=convert[temp16&0xf];
206  }
207
208  address_display()
209  {
210
211  temp16 = PC;
212
213  hex4(temp16);
214
215  }
216
217  data_display()
218  {
219   dptr =PC;
220
221   n = *dptr;
222
223   temp16 = n;
224
225   buffer[0]= convert[temp16&0xf];
226   temp16>>=4;
227   buffer[1]= convert[temp16&0xf];
228
229   buffer[2]=0;
230
231  }
232
233  read_memory()
234  {
235   address_display();
```

```
236    data_display();
237   }
238
239
240  key_address()
241  {
242   state = 1;
243   read_memory();
244   dot_address();
245   hit=0;
246  }
247
248  key_data()
249  {
250
251   read_memory();
252   dot_data();
253   hit=0;
254   state=2;
255
256  }
257
258
259
260  key_PC()
261  {
262   PC=save_PC;
263   key_data();
264  }
265
266  key_plus()
267  {
268
269   if(state==1 || state==2)
270   {
271    PC++;
272    display_PC=PC;
273    read_memory();
274    key_data();
275   }
276
277  if(state==4)
278  {
279   start = display_PC;
280   state = 5;
281   buffer[0] = 0x5e;
282   hit =0;
```

```
283
284
285
286     }
287
288     if(state==6)
289     {
290      start = display_PC;
291      state = 7;
292      buffer[0] = 0x5e;
293      hit =0;
294
295
296
297     }
298
299
300
301
302   }
303
304   key_minus()
305   {
306    if(state==1 | state ==2)
307    {
308     PC--;
309     display_PC=PC;
310     read_memory();
311     key_data();
312    }
313   }
314
315   data_hex()
316   {
317
318    dptr = PC;
319    x = *dptr;
320    if(hit==0) x=0;
321    {
322     hit =1;
323     x = x << 4;
324     x = x|key;
325
326     *dptr = x;
327
328     read_memory();
329
```

```
330    dot_data();
331   }
332  }
333
334  hex_address()
335  {
336   if(hit==0) PC=0;
337   {
338    hit=1;
339
340    PC<<=4;
341    PC |= key;
342    read_memory();
343    dot_address();
344   }
345  }
346
347  /* insert word and shift 512 words down */
348
349  insert()
350  {
351   dptr=PC;
352   for(j=512; j>0; j--)
353   {
354    *(dptr+j)=*(dptr+j-1);
355   }
356   *(dptr+1)=0; /* insert next byte */
357   PC++;
358   read_memory();
359   dot_data();
360   state=2;
361  }
362
363
364  /* delete current word and shift 512 words up */
365
366  cut_byte()
367  {
368   dptr=PC;
369   for(j=0; j<512; j++)
370   {
371    *(dptr+j)=*(dptr+j+1);
372   }
373   read_memory();
374   dot_data();
375   state=2;
376  }
```

```
377
378
379  reg_ax()
380  {
381
382   temp16 = user_ax;
383
384   hex4low(temp16);
385
386   buffer[4]=0;
387   buffer[5]=0x64;
388   buffer[6]=0x77;
389   buffer[7]=0;
390  }
391
392  reg_bx()
393  {
394
395   temp16 = user_bx;
396
397   hex4low(temp16);
398
399   buffer[4]=0;
400   buffer[5]=0x64;
401   buffer[6]=0x7c;
402   buffer[7]=0;
403  }
404
405  reg_cx()
406  {
407
408   temp16 = user_cx;
409
410   hex4low(temp16);
411
412   buffer[4]=0;
413   buffer[5]=0x64;
414   buffer[6]=0x39;
415   buffer[7]=0;
416  }
417
418  reg_dx()
419  {
420
421   temp16 = user_dx;
422
423   hex4low(temp16);
```

```
424
425   buffer[4]=0;
426   buffer[5]=0x64;
427   buffer[6]=0x5e;
428   buffer[7]=0;
429  }
430
431
432  reg_sp()
433  {
434
435   temp16 = user_sp;
436
437   hex4low(temp16);
438
439   buffer[4]=0;
440   buffer[5]=0x73;
441   buffer[6]=0x6d;
442   buffer[7]=0;
443  }
444
445  reg_bp()
446  {
447
448   temp16 = user_bp;
449
450   hex4low(temp16);
451
452   buffer[4]=0;
453   buffer[5]=0x73;
454   buffer[6]=0x7c;
455   buffer[7]=0;
456  }
457
458  reg_si()
459  {
460
461   temp16 = user_si;
462
463   hex4low(temp16);
464
465   buffer[4]=0;
466   buffer[5]=0x30;
467   buffer[6]=0x6d;
468   buffer[7]=0;
469  }
470
```

```
471  reg_di()
472  {
473
474   temp16 = user_di;
475
476   hex4low(temp16);
477
478   buffer[4]=0;
479   buffer[5]=0x30;
480   buffer[6]=0x5e;
481   buffer[7]=0;
482  }
483
484  reg_cs()
485  {
486
487   temp16 = user_cs;
488
489   hex4low(temp16);
490
491   buffer[4]=0;
492   buffer[5]=0x6d;
493   buffer[6]=0x39;
494   buffer[7]=0;
495  }
496
497  reg_ds()
498  {
499
500   temp16 = user_ds;
501
502   hex4low(temp16);
503
504   buffer[4]=0;
505   buffer[5]=0x6d;
506   buffer[6]=0x5e;
507   buffer[7]=0;
508  }
509
510  reg_ss()
511  {
512
513   temp16 = user_ss;
514
515   hex4low(temp16);
516
517   buffer[4]=0;
```

```
518   buffer[5]=0x6d;
519   buffer[6]=0x6d;
520   buffer[7]=0;
521  }
522
523  reg_es()
524  {
525
526   temp16 = user_es;
527
528   hex4low(temp16);
529
530   buffer[4]=0;
531   buffer[5]=0x6d;
532   buffer[6]=0x79;
533   buffer[7]=0;
534  }
535
536  reg_flag()
537  {
538
539   temp16 = user_flag;
540
541   hex4low(temp16);
542
543   buffer[4]=0;
544   buffer[5]=0x38;
545   buffer[6]=0x71;
546   buffer[7]=0;
547  }
548
549  reg_flag_low()
550  {
551   flag_low=0;
552
553   temp16 = user_flag;
554
555   if(temp16&1) flag_low|=1;
556   if(temp16&4) flag_low|=0x10;
557   if(temp16&0x10) flag_low|=0x100;
558   if(temp16&0x40) flag_low|=0x1000;
559
560   hex4low(flag_low);
561
562   buffer[4]=0;
563   buffer[5]=0x38;
564   buffer[6]=0x71;
```

```
565   buffer[7]=0;
566  }
567
568  reg_flag_high()
569  {
570   flag_high=0;
571
572   temp16= user_flag;
573
574   if(temp16&0x80) flag_high|=1;
575   if(temp16&0x200) flag_high|=0x10;
576   if(temp16&0x400) flag_high|=0x100;
577   if(temp16&0x800) flag_high|=0x1000;
578
579   hex4low(flag_high);
580
581   buffer[4]=0;
582   buffer[5]=0x76;
583   buffer[6]=0x71;
584   buffer[7]=0;
585  }
586
587
588  reg_display()
589  {
590
591   if(key==0) reg_ax();
592   if(key==1) reg_bx();
593   if(key==2) reg_cx();
594   if(key==3) reg_dx();
595   if(key==4) reg_sp();
596   if(key==5) reg_bp();
597   if(key==6) reg_si();
598   if(key==7) reg_di();
599   if(key==8) reg_cs();
600   if(key==9) reg_ds();
601   if(key==0xa) reg_ss();
602   if(key==0xb) reg_es();
603   if(key==0xc) reg_flag();
604   if(key==0xd) reg_flag_high();
605   if(key==0xe) reg_flag_low();
606
607
608  }
609
610  key_reg()
611  {
```

```
612   buffer[7]=0;
613   buffer[6]=0x50;
614   buffer[5]=0x79;
615   buffer[4]=0x6f;
616   buffer[3]=0;
617   buffer[2]=0;
618   buffer[1]=0;
619   buffer[0]=0;
620
621   state = 3; /* register display state = 3 with hex key */
622
623  }
624
625
626
627  key_go()
628  {
629
630   if(state==1 || state==2)
631   {
632
633    asm {
634
635          MOV AX,SP
636          MOV save_stack,AX
637
638          MOV AX,user_sp
639          MOV SP,AX  ; LOAD SP WITH USER STACK
640
641          MOV AX,user_cs
642          PUSH AX
643          MOV AX,PC  ;user_ip
644          PUSH AX
645
646          MOV AX,user_di
647          PUSH AX
648          MOV AX,user_si
649          PUSH AX
650          MOV AX,user_bp
651          PUSH AX
652          MOV AX,user_dx
653          PUSH AX
654          MOV AX,user_cx
655          PUSH AX
656          MOV AX,user_bx
657          PUSH AX
658          MOV AX,user_ax
```

```
659          PUSH AX
660
661
662
663          POP AX
664          POP BX
665          POP CX
666          POP DX
667          POP BP
668          POP SI
669          POP DI
670          RETF  ;JUMP TO USER PROGRAM WITH CS = 0000
671         }
672
673
674    }
675
676
677
678    if(state==5)
679    {
680
681     destination = display_PC;
682     temp = destination-(start+3);
683     hex4(temp);
684     dptr = start+1;
685     *(dptr) = (char)temp&0xff;
686     *(dptr+1)= (char)(temp>>=8);
687     PC = start+1;
688     read_memory();
689
690     state = 1;
691
692
693
694
695    }
696
697    if(state==7)
698    {
699
700     destination = display_PC;
701     temp = destination-(start+2);
702     hex4(temp);
703     dptr = start+1;
704     *(dptr)= (char)(temp&0xff);
705     PC = start+1;
```

```
706    read_memory();
707
708    state = 1;
709
710
711
712
713   }
714
715
716
717
718  }
719
720
721
722  service_break()
723  {
724   asm{
725
726       MOV user_ax,AX
727       MOV user_bx,BX
728       MOV user_cx,CX
729       MOV user_dx,DX
730       MOV user_bp,BP
731       MOV user_si,SI
732       MOV user_di,DI
733
734       POP AX
735       MOV save_PC,AX
736       MOV PC,AX
737
738       POP AX
739       MOV user_cs,AX
740       POP AX
741       MOV user_flag,AX
742       MOV user_sp,SP
743
744       CALL read_memory
745
746       MOV AX,save_stack
747       INC AX
748       INC AX
749       MOV SP,AX
750
751       RET
752
```

```
753     }
754
755  }
756
757
758  key_step()
759  {
760
761
762
763  asm {
764
765       MOV AX,SP
766       MOV save_stack,AX
767
768       MOV AX,user_sp
769       MOV SP,AX    ; LOAD SP WITH USER STACK
770
771       PUSHF
772       MOV BP,SP
773       OR >[BP],#$0100    ; set trap flag
774
775       MOV AX,user_cs
776       PUSH AX
777       MOV AX,PC    ;user_ip
778       PUSH AX
779
780       MOV AX,user_di
781       PUSH AX
782       MOV AX,user_si
783       PUSH AX
784       MOV AX,user_bp
785       PUSH AX
786       MOV AX,user_dx
787       PUSH AX
788       MOV AX,user_cx
789       PUSH AX
790       MOV AX,user_bx
791       PUSH AX
792       MOV AX,user_ax
793       PUSH AX
794
795
796
797       POP AX
798       POP BX
799       POP CX
```

```
800       POP DX
801       POP BP
802       POP SI
803       POP DI
804       IRET    ;JUMP TO USER PROGRAM WITH IRET, TRAP FLAG WAS SET
805      }
806
807
808  }
809
810
811  offset16()
812  {
813    buffer[0]=0x6d;
814    buffer[1]=0;
815    display_PC=PC;
816    state = 4;      // enter word address
817    hit=0;
818
819
820  }
821
822  offset8()
823  {
824
825    buffer[0]=0x6d;
826    buffer[1]=0;
827    display_PC=PC;
828    state = 6; // enter word address
829    hit=0;
830
831
832  }
833
834  word_enter()
835  {
836
837   if(hit==0) display_PC=0;
838   {
839    hit=1;
840    display_PC<<=4;
841    display_PC |= key;
842    hex4(display_PC);
843    dot_address();
844   }
845  }
846
```

```
847  key_test()
848  {
849
850    test_sound();
851
852    enable();
853    for(;;)
854    ;
855
856  }
857
858
859  key_exe()
860  {
861
862    if( key>15)
863    {
864     if(key==0x13) key_address();
865     if(key==0x12) key_data();
866     if(key==0x17) key_plus();
867     if (key==0x16) key_minus();
868     if (key==0x10) key_PC();
869     if (key==0x21) key_go();
870     if (key==0x11) key_reg();
871     if (key==0x18) insert();
872     if (key==0x19) cut_byte();
873     if (key==0x14) offset16();
874     if (key==0x15) offset8();
875     if (key==0x20) key_step();
876     if (key==0x22) key_test();
877
878    }
879
880   else
881    {
882
883     if(state==1) hex_address();
884     if(state==2) data_hex();
885     if(state==3) reg_display();
886     if(state==4 || state ==5 || state ==6 || state==7) word_enter();
887
888
889
890    }
891
892  }
893
```

```
894
895
896
897  /* return internal code hex keys and function keys */
898
899  char key_code(char n)
900  {
901   if(n == 3) return 0;
902   if(n == 0x22) return 1;
903   if(n == 0x1c) return 2;
904   if(n == 0x16) return 3;
905   if(n == 2) return 4;
906   if(n == 0xf) return 5;
907   if(n == 0x21) return 6;
908   if(n == 0x1b) return 7;
909   if(n == 7) return 8;
910   if(n == 1) return 9;
911   if(n == 0x1a) return 0xa;
912   if(n == 0x14) return 0xb;
913   if(n == 0xc) return 0xc;
914   if(n == 6) return 0xd;
915   if(n == 0) return 0xe;
916   if(n == 0x19) return 0xf;
917
918   if(n == 0x13) return 0x10;
919   if(n == 0xe) return 0x11;
920   if(n == 0x15) return 0x12;
921   if(n == 0x10) return 0x13;
922
923   if(n == 0x23) return 0x14;
924   if(n == 0x1d) return 0x15;
925   if(n == 0x17) return 0x16;
926   if(n == 0x11) return 0x17;
927
928   if(n == 9) return 0x18;
929   if(n == 8) return 0x19;
930   if(n == 0x12) return 0x20;
931   if(n == 0xd) return 0x21;
932
933   if(n == 0x18) return 0x22;
934   if(n == 0x30) return 0x23;
935
936
937  }
938
939  chk_serial()
940  {
```

```
941
942     if(chkchr()==0x0d)
943     {
944
945      out(port2,0xff);
946      out(port1,0x00);
947
948      asm {
949
950           JMP $c000
951          }
952
953
954         // out(gpio1,0xaa);
955
956          }
957
958     }
959
960
961
962
963
964
965
966
967
968
969
970
971   delay(int d)
972   {
973      for(x=0; x<d; x++)
974      ;
975
976   }
977
978   char scan()
979   {
980    k=1;
981    u=0;
982    key=-1; // if no key pressed key=-1
983    q=0; // key code
984
985    for(i=0; i<8; i++)
986    {
987     out(port1,~k); // write digit
```

```
 988
 989   out(port2,buffer[i]); // write segment
 990
 991  // delay(10);
 992
 993  if(buffer[i] != 0x06 && buffer[i] != 0x86) delay(10);
 994   else delay(1);
 995
 996  out(port2,0);
 997    o= in(port0); // read keypad
 998   for(n=0; n<6; n++)
 999   {
1000    if((o&1)==0) key=q;
1001    else q++;
1002    o>>=1;
1003   }
1004
1005    k<<=1;
1006
1007   }
1008
1009
1010   return key;
1011  }
1012
1013  // scan1 repeat scan display and keyboard
1014
1015  void scan1()
1016  {
1017   temp2=0;
1018
1019
1020
1021   while((scan()!= -1) && ((in(port0)&0x40) !=0))
1022
1023   {
1024
1025    chk_serial();
1026
1027
1028   }
1029   delay(10);
1030
1031
1032
1033
1034  while(scan()== -1)
```

```
1035  {
1036   chk_serial();
1037   auto_key=0;
1038  }
1039   delay(10);
1040
1041   key = scan();
1042  // check key in range
1043   if(key>=0 && key <0x31)
1044   {
1045
1046    key= key_code(key);
1047
1048    key_exe();
1049    // out(gpio1,key);
1050
1051   }
1052
1053  }
1054
1055  service_timer()
1056  {
1057
1058   out(gpio1,i++);
1059
1060   asm{
1061       POP BP
1062       IRET
1063   }
1064
1065  }
1066
1067  test_sound()
1068  {
1069
1070   for(j=0; j<2000; j++)
1071   {
1072    out(port3,j);
1073    delay(1);
1074   }
1075   out(port3,0); // turn off
1076
1077  }
1078
1079
1080  insert_vectors()
1081  {
```

```
asm{

    MOV AX,#service_break+3
    MOV $000c,AX     ; SERVICE ADDRESS FOR INT 3
    MOV AX,#$F000    ; CS SEGMENT FOR MONTIOR ROM F000, RAM 0000
    MOV $000E,AX

* INSERT VECTOR FOR INT 1 TRAP

    MOV AX,#service_break+3
    MOV $0004,AX     ; SERVICE ADDRESS FOR INT 2
    MOV AX,#$F000    ; CS SEGMENT FOR MONTIOR ROM F000, RAM 0000
    MOV $0006,AX

* INSERT VECTOR FOR EXTERNAL INTR

    MOV AX,#service_timer
    MOV $3FC,AX
    MOV AX,#$F000    ; CS SEGMENT FOR MONTIOR ROM F000, RAM 0000
    MOV $3FE,AX

 }
}


init_UART()
{

 out(UART+6,0x83);
 out(UART+0,16); // 9600 8n1
 out(UART+2,0);
 out(UART+6,3);
 out(UART+2,0);
 out(UART+8,3);

}



main()
{
 disable(); // disable interrupts

 out(port3,0); // turn off speaker


 PC = 0x400;
```

```
1129   save_PC = 0x400;
1130   user_sp = 0xff00;
1131   user_cs = 0x0000;
1132   user_ds = 0x0000;
1133   user_ss = 0x0000;
1134   user_es = 0x0000;
1135
1136
1137   init_UART(); // 9600 8n1
1138
1139   strcpy(buffer2,"\r\n8086 MICROPROCESSOR KIT 2018");
1140
1141   putstr("\r\n8086 MICROPROCESSOR KIT 2018");
1142
1143   if(check_LCD()==0)
1144   {
1145    InitLcd();
1146    Puts("8086 MICROPROCESSOR KIT 2018");
1147    goto_xy(0,1);
1148    Puts("256kB RAM, 256kB ROM, UART");
1149   }
1150
1151   out(gpio1,0); // tur off gpio1
1152
1153   buffer[7]=0;
1154   buffer[6]=0;
1155   buffer[5]= convert[8];
1156   buffer[4]= convert[0];
1157   buffer[3]= convert[8];
1158   buffer[2]= convert[6];
1159   buffer[1]=0;
1160   buffer[0]=0;
1161
1162   insert_vectors();
1163
1164   while(1)
1165   {
1166
1167   scan1();
1168
1169   }
1170
1171
1172  }
1173
1174
```

NOTE