

6GEI264 - Vérification et Validation Logicielle - Projet de conception

Jean-Luc Cyr

2017-05-15, rev 1.5

Table des matières

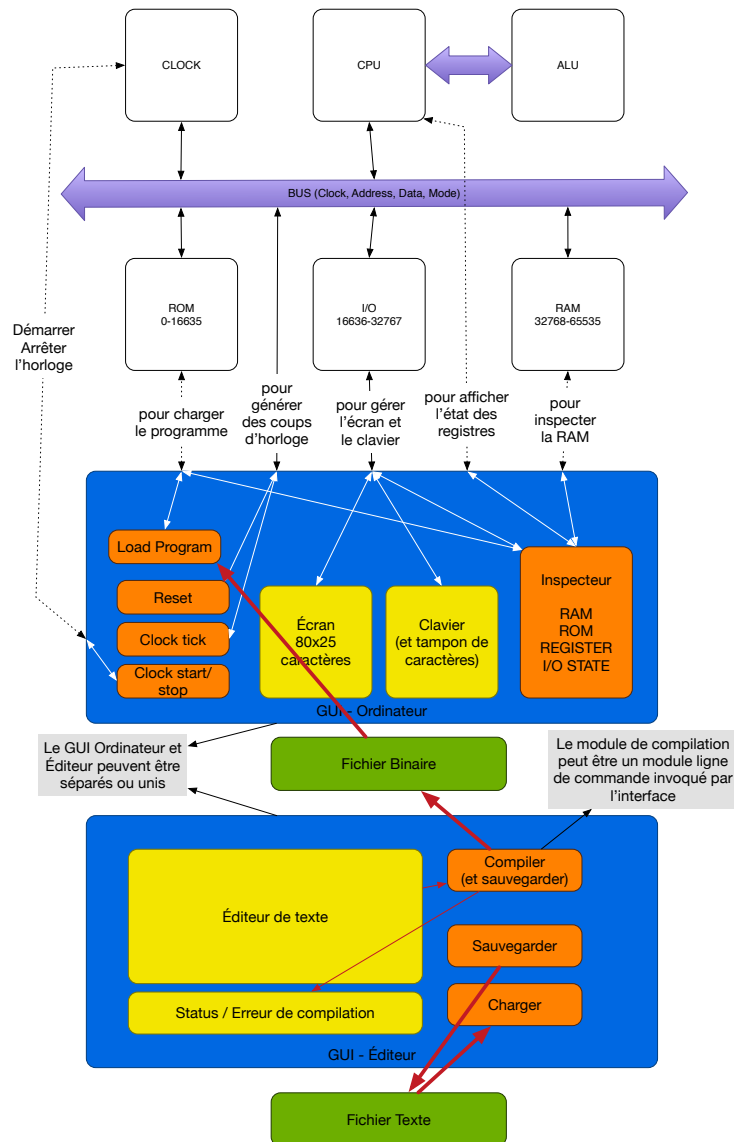
1	Description	4
1.1	Editeur	5
1.2	Assembleur	5
1.3	Ordinateur	5
2	Éditeur	6
3	assembleur	7
3.1	Assembleur - Fonctions	7
3.2	Assembleur - Adressage	7
4	processeur	8
4.1	Fonctionnement processeur	8
4.2	Spécifications	8
4.2.1	Registres	8
4.2.2	Bus	9
4.3	Entrées/sorties (memory map)	9
4.4	Clock	9
5	Assembleur	10
5.1	Data manipulation	10
5.1.1	DTA	10
5.2	Arithméric	10
5.3	Logic	10
5.4	Comparison	11
5.5	Flow control	11
5.6	Autres	11
6	Byte Code	12
6.1	Description	12
6.2	adressage	12
6.3	observations et exemples	13

7	Composants	14
7.1	BUS	14
7.2	CPU	16
7.3	ALU	16
7.4	ROM	17
7.5	RAM	17
7.6	CLOCK	17
7.7	GUI	18
	7.7.1 Editeur	18
	7.7.2 Ordinateur	18
8	Rappel	20
9	Exemple de programme	21
9.1	Écrire salut à l'écran	21

version	description
1.0	version initiale
1.1	Précision sur les registres en 4.2.1 Ajout des schémas Ajout d'exemples de class
1.2	Mode d'adressage en 3.2 Ajout au byte code pour les types d'adressage en 6.1 et 6.2 Enrichissement de la section 7 décrivant les composants
1.3	Ajout fonction assignation et data à l'assembleur section 3.1 et 5.1 Déplacement des bouts de code aux sections appropriées
1.4	precision sur les nb negatifs (section 3) precision sur l'instruction DTA (section 3) ajustement table adressage mémoire
1.5 2017-04-03	spécification du format de fichier exemple de programme (section 9) spécification du format des données ajout table ASCII code complet du BUS

1 Description

Le projet consiste en la réalisation d'un micro-assembleur et d'un mini-ordinateur logiciel capable d'exécuter les programmes générés avec cet assembleur.



1.1 Editeur

Le module d'édition du code comprend minimalement, une fenêtre d'édition avec les fonctions pour sauvegarder le programme, charger un programme à partir d'un fichier, lancer l'assemblage du programme avec sa sauvegarde ou sa transmission vers la ROM de l'ordinateur. Idéalement chaque ligne d'instruction devrait afficher l'adresse correspondante en mémoire où elle se trouvera après assemblage dans le but de pouvoir indiquer les endroits de branchement (JMP).

1.2 Assembleur

Il est souhaitable de réaliser cet éditeur pour l'assembleur. Cependant, le module d'assemblage peut être réalisé de manière autonome et utilisable en ligne de commande.

1.3 Ordinateur

Le module mini-ordinateur peut être lancé à partir de l'éditeur ou de manière autonome en ligne de commande. Description des fonctionnalités complètes du projet. L'interface minimale du mini-ordinateur doit comprendre : un écran, un endroit pour voir l'état de l'ordinateur, une fonction pour charger un programme, une fonction pour lancer le programme, une fonction pour exécuter seulement une étape (instruction) du programme, la saisie des touches du clavier.

2 Éditeur

La partie fenêtre de saisie/édition du code source assembleur.

Fonctions disponibles :

- Assembler (compilation du programme de la source en code binaire)
- Load (charge un fichier texte dans l'éditeur)
- Save (sauvegarde le contenu de l'éditeur dans un fichier texte)
- Program (copie le "byte code" dans l'espace "ROM" de l'ordinateur)
- Execution mode (lance l'exécution du programme dans l'ordinateur)
 - Mode "clock mode" - une minuterie déclanche les changement d'étape
 - Mode "button click simulate clock" - un clic sur un bouton déclanche les changement d'étape
- Status Display (register content, screen output, ...) Permet de visualiser l'état de l'ordinateur
- Input buffer (read keyboard, stock key and status, ...) Gère les entrées clavier destinés à l'ordinateur

3 assembleur

L'ensemble des valeurs utilisés sont représentés par des entiers positifs de 16 bits.

Les instructions sont représentés avec leur opérants de destination par une valeur de 16 bits positive.

Les entrants, valeurs ou adresse sont également représentés par des valeurs de 16 bits positives.

Les opérations donnant des résultats négatifs (ex : $32 - 64 = -32$) seront représentés par leur valeur entière positive (ex : 32) et l'assignation du bit de registre de status "sign" à 1 pour signifier que le résultat est négatif.

L'assembleur permet le stockage de valeur avec l'instruction DTA. Cette instruction n'est pas interprétée par le CPU, elle sert uniquement à transférer des valeurs de l'assembleur dans le contenu du programme assemblé à être exécuté.

3.1 Assembleur - Fonctions

Instructions : Instruction Fixes : Préfix, Operand1, [Operand2]

— Data storage : Dta

Cette fonction permet d'insérer des valeurs dans le ROM pour leur utilisation ou leur déplacement en RAM par le programme.

— Data manipulation : Move/Load

— Arithmetic : Add/Sub/Mul/Div

— Logic : And/Or/XOr/Not

— Comparison : Gt/Lt/Eq/GtE/LtE/Neq

— Branch : JMP

— Conditional : BRA non zero, BRZ zero, BRO overflow

— Other : NOP

— PAS DE : sous-routines, utilisation de stack, interruption, etc.

3.2 Assembleur - Adressage

Adressing

— Immediate (data value written in code)

— Direct (memory address written in code)

— Indirect (memory address taken from a register value)

— PAS DE : Relative (based on program counter)

— PAS DE : Index (memory address is a register value added by a value)

Pour ceux qui désirent implémenter d'autres modes d'adressage, voir la stratégie en section 6.1 et 6.2.

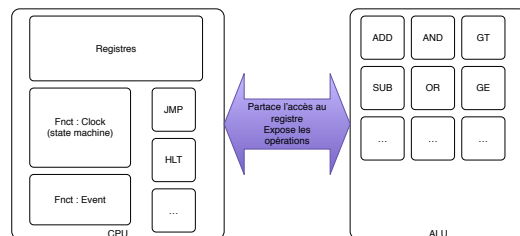
4 processeur

4.1 Fonctionnement processeur

Control Unit (program counter, instruction register) travaille sur 4 cycles

1. Fetch Instruction (based on PC, fill IR and increment PC)
2. Decode Instruction (Instruction in IR is decoded)
3. Read Address (Fill data registers)
4. Execute

Arithmetic logical Unit (accumulator)



4.2 Spécifications

4.2.1 Registres

Register	Description
P	Program Counter or Instruction Pointer (16 bits)
I	Instruction Register (16 bits)
A-D	General Purpose Register (4 x 16 bits)
S	Status Register (16 bits) 0x0000, 0, 0, 0, CND, Zero, Carry (OverFlow), Sign, Parity

Le registre de status est 16bits, tous les bits non-utilisés sont à 0.

À partir du bit le moins significatif (à droite) voici leur signification :

Parity le resultat de la dernière opération est pair ou impair (dernier bit)

Sign le résultat de la dernière opération est positif ou négatif

Carry le résultat de la dernière opération a généré un dépassement

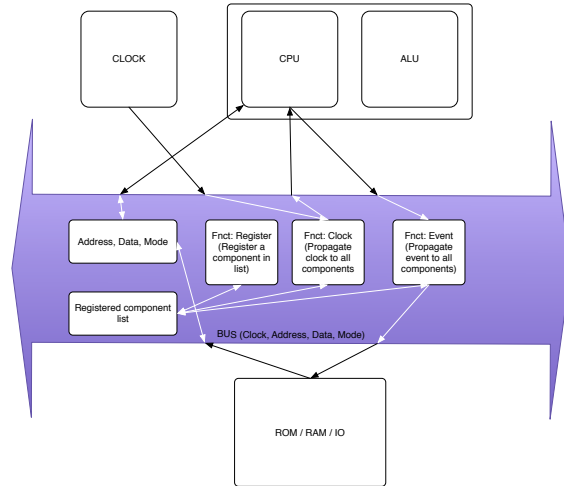
Zero le résultat de la dernière opération est zéro

CND le résultat de la dernière opération de COMPARAISON est vrai ou faux

- Les autres bits sont à 0

All register hold 16 bits values.

4.2.2 Bus



- data 16 bits
- address 16 bits
- mode 16 bits (wait, read, write)
- clock

4.3 Entrées/sorties (memory map)

All memory address contain a 16 bits value.

START	STOP	Description
0	16635	ROM Memory (16k byte Read Only - Program Area)
16636	32767	I/O Area (Read/Write)
16636	16636	Input device (keyboard Read Only)
16892	+2000	Output device (screen 80x25 2000 bytes direct address Write Only)
18892	+4096	Input/Output device (4k file Read/Write direct mapping)
32768	65535	RAM Memory (32k Read/Write - Data Area)

I/O should be restrain to 0-65535 values.

First 32k values (0-32767) is reserved for program (ROM) and I/O

Second 32k values (32768-65535) used as RAM

4.4 Clock

Sert à émettre un battement sur le bus pour déclancher le changement de la machine d'état du CPU.

5 Assembleur

À l'exception de LD et ST toutes les opérations sont effectuées uniquement sur des registres.

La dernière lettre de l'opérand est le registre de destination.

Les comparaisons sont faites sur le registre d'état

5.1 Data manipulation

Opération manipulation de données.

OP	REGISTER	VALUE	Description
DTA		16 bits value <value>	store <value> at current address
SET	A-D	16 bits value	set register <value>
LD	A-D	address	load content of memory at <address> into register A-D
ST	A-D	address	save content of register A-D to memory at <address>
MV	A-D	a-d	copy content of register a-d into register A-D

5.1.1 DTA

Cette fonction peut être élaborée pour permettre de conserver une simple valeur 16 bits ou une chaîne de caractère. De base, la valeur est conservée à l'adresse où placée la valeur dans le code.

Il serait intéressant de reconnaître une chaîne de caractère par exemple entre guillemets “” et de copier chacun des caractères en mémoire (16 bits par caractères) et terminer par une valeur 0. Cependant le support pour une chaîne de caractère n'est pas un requis du laboratoire.

Afin de ne pas introduire d'erreur dans le programme, elle devrait être utilisée à la fin du programme après une instruction HLT.

5.2 Arithmetic

Opération arithmétique entière sur le contenu de deux registres. Le résultat de l'opération est emmagasiné dans le registre A-D de l'opérateur. Les bits de conditions du registre de status S sont ajustés à 0 ou à 1 selon le résultat. (Zero, Parity, Overflow)

OP	REGISTER	REGISTER	Description
ADD	A-D	a-d	add content of register a-d to register A-D
SUB	A-D	a-d	subtract content of register a-d from register A-D
MUL	A-D	a-d	multiply content of register A-D by a-d
DIV	A-D	a-d	devide content of register A-D by a-d

5.3 Logic

Opération logique sur le contenu de deux registres. Le résultat de l'opération est emmagasiné dans le registre A-D de l'opérateur. Les bits de conditions du registre de status S sont ajustés à 0 ou à 1 selon le résultat. (Zero, Parity)

OP	REGISTER	REGISTER	Description
OR	A-D	a-d	A-D OU a-d resultat dans le registre A-D
AND	A-D	a-d	A-D ET a-d resultat dans le registre A-D
XOR	A-D	a-d	A-D OU EXCLUSIF a-d resultat dans le registre A-D
NOT	A-D	<none>	Négation du contenu du registre A-D

5.4 Comparison

Comparaison entre le contenu de deux registres. Les bits de conditions du registre de status S sont ajustés à 0 ‘faux’ ou à 1 ‘vrai’ selon le résultat. (CND)

OP	REGISTER	REGISTER	Description
LT	A-D	a-d	A-D est plus petit a-d
GT	A-D	a-d	A-D est plus grand a-d
LE	A-D	a-d	A-D est plus petit ou égal a-d
GE	A-D	a-d	A-D est plus grand ou égal a-d
EQ	A-D	a-d	A-D est égal a-d
EZ	A-D	<none>	A-D est égal à 0
NZ	A-D	<none>	A-D est différent de 0

5.5 Flow control

Branchement du programme à l’adresse indiquée selon la réalisation de la condition demandée.

OP	ADDRESS	REGISTER CONDITION	Description
JMP	address	<none>	branchement à l’adresse (aucune condition)
JMZ	address	S a le status Zero à 1	branchement à l’adresse
JMO	address	S a le status Overflow à 1	branchement à l’adresse
JMC	address	S a le status CND à 1	branchement à l’adresse

5.6 Autres

Autres instructions.

OP	ADDRESS	REGISTER CONDITION	Description
NOP	<none>	<none>	une opération qui n’effectue rien
HLT	<none>	<none>	arrête l’exécution du programme

6 Byte Code

6.1 Description

<reg> représente la valeur identifiant le registre A :01, B :02, C :03, D :04

<addr> représente une adresse mémoire hexadécimale (0000 à FFFF)

<val> représente une valeur hexadécimale (0000 à FFFF)

OP	REGISTER	VALUE	OPCODE
SET	A-D	val (16 bits value)	0x05<reg> 0x<val>
LD	A-D*	address	0x06<reg> 0x<addr>
ST	A-D*	address	0x07<reg> 0x<addr>
MV	A-D	a-d	0x08<reg> 0x00<reg>
ADD	A-D	a-d	0x11<reg> 0x00<reg>
SUB	A-D	a-d	0x12<reg> 0x00<reg>
MUL	A-D	a-d	0x13<reg> 0x00<reg>
DIV	A-D	a-d	0x14<reg> 0x00<reg>
OR	A-D	a-d	0x21<reg> 0x00<reg>
AND	A-D	a-d	0x22<reg> 0x00<reg>
XOR	A-D	a-d	0x23<reg> 0x00<reg>
NOT	A-D	<none>	0x24<reg>
LT	A-D	a-d	0x31<reg> 0x00<reg>
GT	A-D	a-d	0x32<reg> 0x00<reg>
LE	A-D	a-d	0x33<reg> 0x00<reg>
GE	A-D	a-d	0x34<reg> 0x00<reg>
EQ	A-D	a-d	0x35<reg> 0x00<reg>
EZ	A-D	<none>	0x36<reg>
NZ	A-D	<none>	0x37<reg>
JMP	address	<none>	0x0100 0x<addr>
JMZ	address	S a le status Zero à 1	0x0200 0x<addr>
JMO	address	S a le status Overflow à 1	0x0300 0x<addr>
JMC	address	S a le status CND à 1	0x0400 0x<addr>
NOP	<none>	<none>	0x0000
HLT	<none>	<none>	0x0F00
DTA	<none>	0x<val>	place la valeur en ROM
DTA	<none>	"string"	place les valeurs en ROM

*Adressage différent, ajuster les bits supérieur du registre et remplacer la valeur <addr> par <reg> selon le cas.

Note : La fonction DTA ne génère pas d'instruction, elle copie simplement la valeur c'est pourquoi elle n'a pas de <op code> pas dans le tableau précédent.

Commandes sur 16 bits (8 bits pour la commande, 8 bits pour le registre)

6.2 adressage

Adressage Vous pouvez utiliser les bits supérieurs de la partie registre pour indiquer un mode d'adressage différent. Utiliser le tableau ci-dessous pour l'im-

plémenter.

00	L'argument est un registre
01	L'adresse à accéder est contenu dans le registre en argument
10	L'argument est une adresse
11	L'argument est l'adresse mémoire où aller chercher l'adresse

Cependant, seule les opération de déplacement d'information peuvent utiliser cette méthode, puisque les opérations effectuées par l'ALU n'ont pas accès à la mémoire, seulement aux registres. L'opération devrait alors utiliser un registre temporaire pour y placer la valeur avant d'exécuter l'opération, ce qui complexifie le fonctionnement de l'ordinateur.

6.3 observations et exemples

Remarquez que les commandes destinées à l'ALU ont une valeur > 0 dans le premier octet.

ex : SET[A-D] VALUE

SET = 0x05, A = 0x01, VALUE = 0x8000

0x0501 0x8000 seront stocké pour représenter SETA 0x8000 ou SETA 32768

LD = 0x06, A = 0x01, utiliser l'adresse dans le registre en argument = 0x10, argument 0x0001 (registre A)

0x0511 0x0001 seront stocké pour représenter LDA A ou Charger A à partir de la mémoire donc l'adresse est dans A

7 Composants

Votre programme principal devrait servir à créer les composants et les inter-relier.

```
1 import tkinter
2 import bus
3 import cpu
4 ...
5
6 if __name__ == '__main__':
7     # Creation du bus a la base de l'ordinateur pour les
8     # communications
9     bus = bus()
10    # Creation du cpu en le reliant au bus
11    cpu = cpu(bus)
12    # Creation de la rom en la reliant au bus
13    rom = rom(bus)
14
15    # Creation du GUI
16    ...
17
18    # Connexion du GUI aux composants
19
20    # Boucle sans fin du programme
```

7.1 BUS

Contient les éléments de communication entre les composants.

La liste des composants qui y sont reliés, et les valeurs transigées : adresse, valeur, mode et horloge.

Exemple complet du code du BUS dans la démonstration effectuée en classe.

```
1 class bus:
2     """
3     Composant BUS de donnee, servant a propager les
4     evenements entre les differents composants.
5
6     Les valeurs sont mise a jour par l'appellant puis la
7     fonction event() est appelee pour propager l'evenement.
8     - adress
9     - mode (1, 2, 8 ou 9)
10    - data (si ecriture)
11
12    La valeur resultante est disponible au retour de l'execution
13    de la fonction
14    - data (si lecture)
15
16    La fonction clock() sert a propager l'horloge vers les
17    composants.
18    """
19
20
```

```

22 components = [] # registered components
   address = 0    # 0-65535
   mode = 0       # 0:inactive 1:read 2:write 8: RESET 9: HALT
24   data = 0      # 0-65535
   def __init__(self):
26       # Init
       return
28
   def register(self, component):
30       # Register components
       self.components.append(component)
32       return
34
   def clock(self):
       # Propagate clock tick
36       if self.components:
           for i in self.components:
38               if i.clock:
                   i.clock()
40       return
42
   def event(self):
       # Propagate events
44       if self.components:
           for i in self.components:
46               if i.event:
                   i.event()
48       self.mode = 0
       return

```

Tous les composants sont reliés au bus. Dans leur initialisation ils reçoivent l'objet bus en paramètre et doivent s'y enregistrer.

```

1 def __init__(self, bus):
   # Initialisation du composant
3   self.bus = bus
   bus.register(self)
5   return

```

Tous les composants qui s'enregistre sur le bus doivent avoir des fonctions "clock" et "event" pour recevoir ces événements du bus.

```

1 def event(self):
   # Traitement d'un evenement du bus
3   if self.bus....
       # Traitement
5   return
7
   def clock(self):
       # Traitement d'un coup d'horloge
9
       return

```

Le bus quand à lui implémente ces fonctions qui pourront être appelées par les composants pour propager un battement d'horloge ou un évènement vers les autres composants.

7.2 CPU

contient les registres, les fonctions de bases du langage

```
class cpu:
2   register
   components
4
   def __init__(self, bus):
6       # Initialisation du composant
       bus.register(self)
8       return

10  def register(self, component):
    # Enregistre l'ALU ou autres extensions
12    ...
    return

14  def event(self):
16    ...
    return

18  def clock(self):
20    # Machine d'etat du CPU
    Fetch
22    Decode
    Execute
24    ...
    return

26  def ...:
28    # fonction
    return
```

7.3 ALU

Reliée seulement au CPU. Contient les opération arithmetiques et logiques, accède au registre du CPU.

L'ALU peut être déclarée dans le programme principal et enregistrée au CPU ou être déclaré directement dans le CPU.

```
1 import tkinter
import bus
3 import cpu
...
5
if __name__ == '__main__':
7     ...
```



```

    alu = alu()
9    cpu = cpu(bus, alu)
    ...

```

ou

```

1 class cpu:
    ...
3    alu

5    def __init__(self, bus):
        # Initialisation du composant
7        ...
        self.alu = alu()
9        ...

```

7.4 ROM

contient la mémoire du programme

```

1 class ram: (ou rom ou I/O)
    bus
3    data

5    def __init__(self, bus):
        # Initialisation du composant
7        bus.register(self)
        return

9    def event(self):
11       # Traitement d'un evenement du bus
        ...
13       return

15    def clock(self):
        ...
17       return

```

7.5 RAM

contient la mémoire utilisable par le programme

7.6 CLOCK

Émet à interval régulier un évènement sur le bus

Le bus propage cet évènement à chacun des composants enregistrés. Dans le cadre du laboratoire, seul le CPU doit exécuter une tâche sur l'évènement

“clock” soit de changer son cycle actif et d’exécuter les actions appropriées pour son nouvel état. Les autres composants devraient simplement retourner sans faire de traitement.

```
class ram: (ou rom ou I/O)
2   bus

4   def __init__(self, bus):
    # Initialisation du composant
6   bus.register(self)
    return

8   def run(self):
    # Emission d'un coup d'horloge
10  while(1):
    'sleep'
12    self.bus.clock()
14  return
```

7.7 GUI

Attention de distinguer les parties GUI de l'ordinateur et GUI de l'éditeur/-compilateur. Vous pouvez inclure les deux parties dans le même logiciel ou les conserver distinctes.

Note : Si les deux parties sont réunies, le transfert du code binaire peut se faire directement dans l'espace "ROM" de l'ordinateur à partir de l'éditeur lors de la compilation (sans utiliser un fichier binaire).

7.7.1 Editeur

Editeur de texte avec affichage des numéro de ligne correspondant aux adresses mémoire où les instructions seront stockées lors de l'exécution.

N'oubliez pas de tenir compte de la longueur des instructions et leur valeurs en mémoire (1 ou 2 fois 16 bits) pour ajuster le numéro de la ligne suivante.

Boutons :

- Chargement d'un programme texte
- Sauvegarde d'un programme texte
- Assemblage du texte pour produire un binaire et le sauvegarder dans un fichier ou le place en mémoire ROM.

Affichage : status de l'assemblage, erreur ou autre.

7.7.2 Ordinateur

Ecran : 25 lignes de 80 colonnes (accessible au travers du module Entrée/-Sortie)

Clavier : Saisie des touches, accumulation dans un tampon accessible par le CPU via le BUS (accessible au travers du module Entrée/Sortie).

Boutons :

- Chargement d'un logiciel binaire à partir d'un fichier
- Execution du logiciel avec l'horloge (clock on/off)
- Execution d'un pas du logiciel (un clock d'horloge)
- Reset du CPU (remise à 0 du registre pointeur d'instruction)

Affichage : contenu d'un ou plusieurs éléments mémoire ou registre utilisé pour l'inspection d'un adresse memoire ou d'un registre.

8 Rappel

Remember power of 2 : 1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768,65536
Remember odd number have last bit set to 1

Remember bases conversion

base 16	base 10	base 2
0x0	0	0b0000
0x1	1	0b0001
0x2	2	0b0010
0x3	3	0b0011
0x4	4	0b0100
0x5	5	0b0101
0x6	6	0b0110
0x7	7	0b0111
0x8	8	0b1000
0x9	9	0b1001
0xA	10	0b1010
0xB	11	0b1011
0xC	12	0b1100
0xD	13	0b1101
0xE	14	0b1110
0xF	15	0b1111

Table de caractères ASCII

16	10	Caractère
0x0000	48	0
0x0001	49	1
0x0009	57	9
0x0041	65	A
0x0042	66	B
0x005A	90	Z
0x0061	97	a
0x0062	98	b
0x007A	122	z
0x00AA	170	

9 Exemple de programme

9.1 Écrire salut à l'écran

CODE	VALEUR HEX	VALEUR DEC	DESC
SETA 83	0x0501 0x0053	1281 83	Mets 83 (S) dans le registre A
STA 16892	0x0701 0x41FC	1793 16892	Mets le contenu du reg. A à l'adresse 16892
SETA 65	0x0501 0x0041	1281 65	Mets 65 (A) dans le registre A
STA 16893	0x0701 0x41FD	1793 16893	Mets le contenu du reg. A à l'adresse 16893
SETA 76	0x0501 0x004C	1281 76	Mets 76 (L) dans le registre A
STA 16894	0x0701 0x41FE	1793 16894	Mets le contenu du reg. A à l'adresse 16894
SETA 85	0x0501 0x0055	1281 85	Mets 85 (U) dans le registre A
STA 16895	0x0701 0x41FF	1793 16895	Mets le contenu du reg. A à l'adresse 16895
SETA 84	0x0501 0x0054	1281 84	Mets 84 (T) dans le registre A
STA 16896	0x0701 0x4200	1793 16896	Mets le contenu du reg. A à l'adresse 16896
HLT	0x0F00	3840	Arrêt de l'exécution

Le fichier correspondant généré par l'éditeur et chargé par l'ordinateur devrait donc contenir les valeurs 16 bits suivantes :

1281 83 1793 16892 1281 65 1793 16893 76 16894 85 16895 84 1793 16896 3840

Les valeurs sauvegardés dans le fichier seront séparées par un espace et/ou un changement de ligne. Chaque valeur lue sera remise sous forme d'entier positif 16 bits.