CACHAN
GEii
Électronique

TD et TP d'informatique industrielle
Geii1 – Module 2

Jusqu'aux vacances de la Toussaint

CACHAN
GEii
Électronique

TD d'informatique industrielle n° 1
Geii1 – Module 2
semaine du 18 au 23/10/99
Architecture des microprocesseurs

Objectifs

Connaître les principaux composants de l'architecture d'un microprocesseur.
Plus précisément, voici ce que vous devrez connaître après cette séance de TD.

- Connaître les étapes de l'exécution des instructions
  - extraction de l'instruction (instruction fetch)
  - décodage
  - exécution
  - accès mémoire
  - écriture registre

- Connaître le rôle des principaux composants d'un microprocesseur
  - Compteur programme
  - Mémoire cache d'instructions et mémoire cache de données
  - Registre d'instructions
  - Banc de registres
  - Unité Arithmétique et Logique

Jeu d'instruction

1/ Donnez la traduction en binaire du programme 1. A partir de l'étude du jeu d'instructions peut-on dire quel est le nombre de registres de ce micro ? Quelle est la taille de la mémoire ?

2/ Expliquez en français ce que font ces programmes.
A quel fragement de code C cela peut-t'il correspondre ?

programme 1 (exemple)

lw $2,10($1)	R2<= Mem[10+R1]	: 100011 00001 00010 0000000000001010
lw $3,11($1)	R3<=Mem[11+R1]	: ……
add $4,$3,$2	R4<=R2+R3;	: ……
sw $4,12($1)	Mem[12+R1]<=R4	: ……

Réponse :

addition de deux nombres et rangement du résultat :
Mem[12+R1] <= Mem[10+R1]+ Mem[11+R1]
code C : tab[12]=tab[10]+tab[11];

programme 2
lw $2,10($1)
lw $3,11($1)

```
add $4,$3,$2
lw $3,12($1)
add $4,$4,$3
lw $3,13($1)
add $4,$4,$3
sw $4,14($1)
```

programme 3

```
lw $2,10($1)
lw $3,11($1)
add $4,$3,$2
slt $5,$2,$3
beq $0,$5, +4        ; (le registre R0 vaut toujours 0 .
                     ; Il est uniquement accessible en lecture)
lw $3,12($1)
add $4,$4,$3
lw $3,13($1)
add $4,$4,$3
sw $4,14($1)
```

Contrôle

3/ Combien y a-t'il d'accès à la mémoire et d'accès aux registres pour une instruction
- de type registre ?
- de Saut ?
- de Chargement/Sauvegarde ?
- de branchement ?

4/ Dessinez, sur les schémas du microprocesseur fournis en annexe page 9, les mouvements de données pour chaque état de l'automate.

5/ Expliquez en français ce qui se passe pour chacun des états de l'automate.

Annexe A : Le MIPS R3000 simplifié

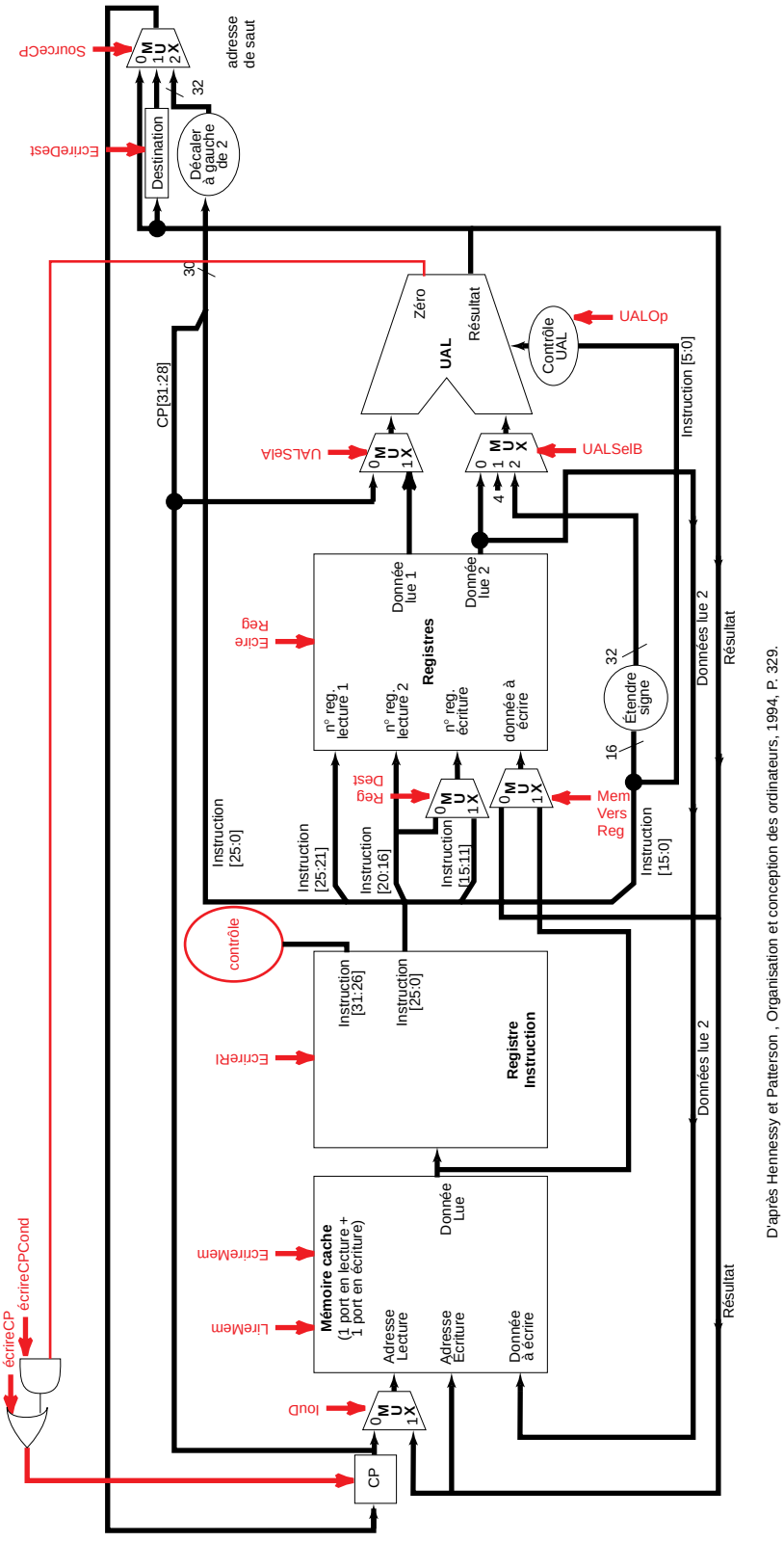


figure A1 : Architecture du R3000 simplifié

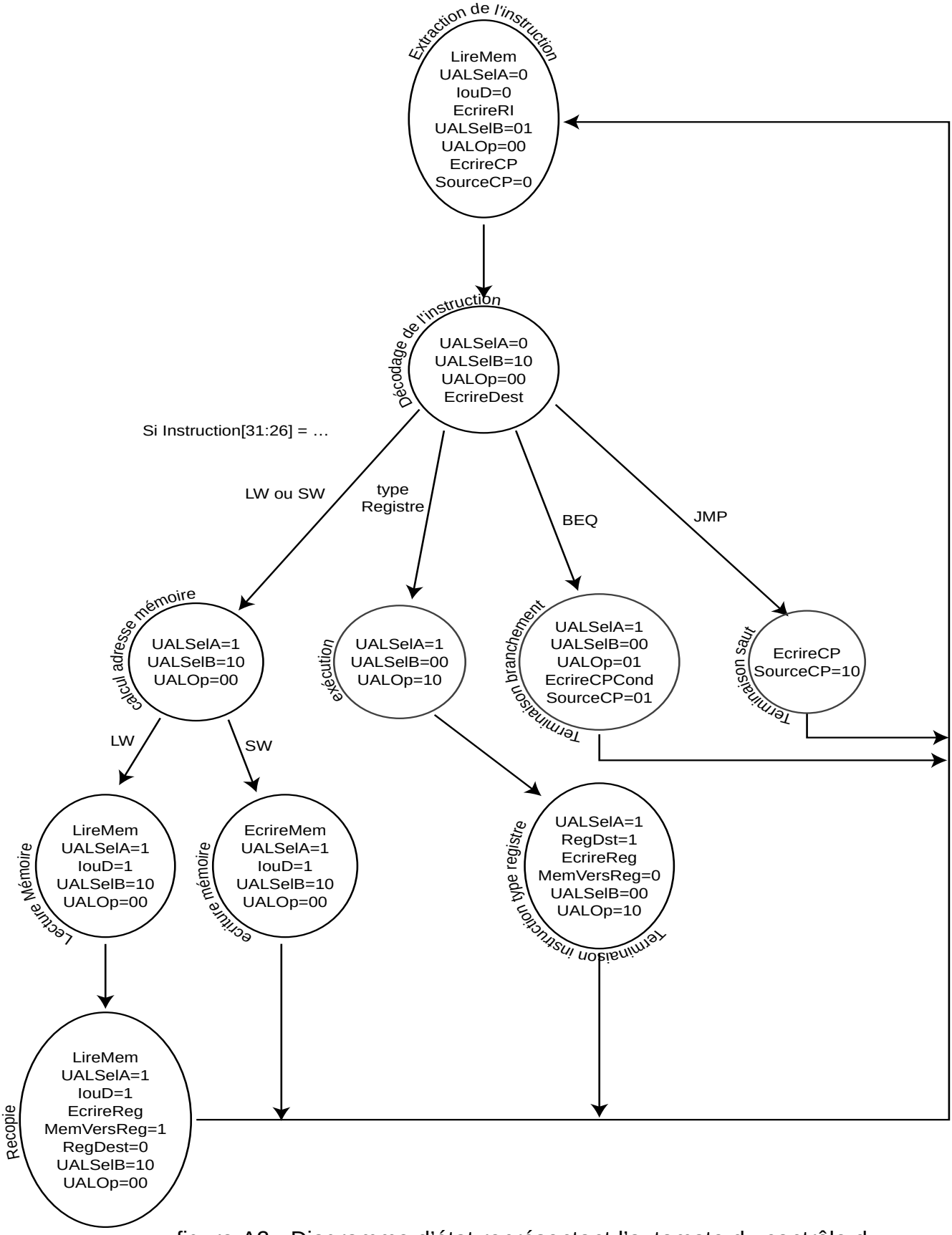


figure A2 : Diagramme d'état représentant l'automate du contrôle du R3000 simplifié.

Annexe A3 : Jeu d'instructions du MIPS R3000 simplifié

Toutes les instructions sont alignées sur des mots de 32 bits. Le premier champ de 6 bits (OpCode) permet d'effectuer le découpage en champ du reste de l'instruction. Les 4 formats d'instructions sont décrits ci-après.

Instruction type Registre

Il s'agit des instructions effectuant des opérations : *add sub and or* et *slt*.

Pour toutes ces instructions l'*Opcode* vaut 000000. *Rs* et *Rt* sont les registres sources alors que*Rd* est le registre destination.La fonction à réaliser par l'UAL (qui se trouve ordinairement dans le champ Opcode) est précisée pour ces instructions dans le champ *funct*. Le champ *resérvé* sert aux décalages. Il est ignoré ici.
Les opérations associées au champ *funct* sont les suivantes :

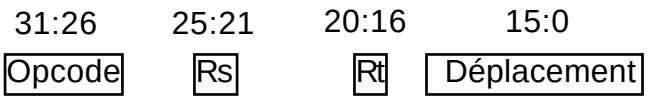
opération	*funct*	signification
add	10 0000	addition
sub	10 0010	soustraction
and	10 0110	opérateur et
or	10 0111	opérateur ou
slt	10 1010	positionner si inférieur

exemples :

Opcode	Rs	Rt	Rd	réservé	funct
00 0000	0 0011	0 0100	0 0010	00000	10 0000
00 0000	0 0111	0 1001	0 0101	00000	10 0111
00 0000	0 0010	0 0011	0 0001	00000	10 1010

```
add   $2, $3, $4          ;   R2<= R3 + R4
or    $5,$7,$9            ;   R5<= R7 ou R9
slt   $1,$2,$3            ;   si (R2<R3) alors R1<=1;
                           ;   sinon R1<=0
```

Instruction type load/store



Il s'agit des instruction de transfert entre un registre et la mémoire. Le registre considéré est Rt et l'adresse du mot mémoire est Rs + Déplacement.
sw (save word, Opcode = 101011) correspond à la sauvegarde de la valeur d'un registre en mémoire alors que *lw* (load word, Opcode = 100011) correspond au chargement d'un registre par une valeur contenue en mémoire.

exemples :

Opcode	Rs	Rt	Déplacement
10 0011	0 0011	0 0100	0000 0000 0010 0000
10 1011	0 0101	0 0100	0000 0000 0010 0001

```
lw $4,32($3)      ;   R4 <= Memoire[32+R3]
sw $4,33($5)      ;   Mémoire[33+R5] <= R4
```

Instruction de Branchement

31:26	25:21	20:16	15:0
Opcode	Rs	Rt	Déplacement (signée)

Il s'agit de l'instruction de branchement conditionnel : *beq* (Branch if EQual, Opcode 000100) Si le résultat de la comparaison est juste, le compteur programme est ainsi modifié :
CP <= CP + Déplacement * 4.
Déplacement est une valeur signée.

exemple :

Opcode	Rs	Rt	Déplacement
00 0100	0 0011	0 0010	0000 0000 0001 0000

beq $2,$3,16 ; si R2==R3 alors CP <= CP + adresse * 4.

Instruction de Saut

31:26	25:0
Opcode	adresse

Il s'agit de l'instruction de saut inconditionnel : *jmp* (JuMP, Opcode 000010) Le compteur programme est ainsi modifié :
PC[27:0] <= adresse * 4;
adresse est une valeur signée.

exemples :

Opcode	Adresse
00 0010	0000 0000 0000 0000 0000 0000 0010

jmp 2 ; PC[27:0] <= 2 * 4;

Annexe A4 : Contrôle de l'UAL

L'unité arithmétique et logique (UAL) reçoit en entrée un mot de contrôle de 3 bits indiquant l'opération à effectuer. Ce mot de trois bits est déterminé par le bloc contrôle de l'UAL à partir du champ funct des instructions type registre et de la sortie UalOp du séquenceur. La codage des opérations est donné dans le tableau ci-dessous.

UalOp	funct	contrôle UAL	opération
OO	XXXXXX	O10	addition
X1	XXXXXX	11O	soustraction
1X	XX0000	O10	addition
1X	XX0010	11O	soustraction
1X	XX0100	OOO	et
1X	XX0101	OO1	ou
1X	XX1010	11I	positionner si inférieur

CACHAN
GEii
Électronique

TD d'informatique industrielle n° 2
Geii1 – Module 2
semaine du 25 au 30/10/99
Bus et cycles bus

Objectifs

Comprendre la notion de bus partagé mono ou bidirectionnel et savoir analyser les chronogrammes représentant les cycles bus d'un microprocesseur.
Plus précisément, voici ce que vous devrez connaître après cette séance de TD :

- Composants
  - Ampli trois états, état haute impédance
  - Plots d'entrées-sorties
  - Séparateur de bus (244, 245)

- Bus partagé
  - Bus mono directionnel mono maître
  - Bus bidirectionnel mono maître

- Mémoire (SRAM)
  - Organisation, adresses d'octet
  - Signaux : bus d'adresse, bus de données, sélection, lecture écriture
  - Cycle de lecture, cycle d'écriture
  - Temps d'accès

- Cycles bus
  - Cycle de lecture
  - Cycle d'écriture
  - Synchronisation, états d'attente

1. Amplificateur trois états

1.1
A partir du schéma ci-dessous d'un amplificateur trois états (buffer tristate), définir pour chaque configuration des entrées le mode de fonctionnement (On/Off) des transistors MOSFET P1 (à canal P, normaly on) et N2 (à canal N, normaly off) ainsi que l'état de la sortie (H,L ou HZ).

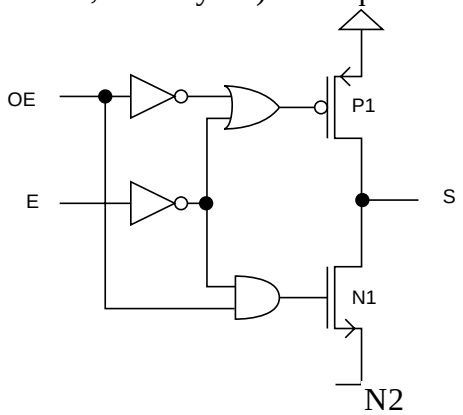


Figure 1 : Schéma d'un amplificateur trois états.

OE	E	P1	N2	S
L	L			
L	H			
H	L			
H	H			

1.2
A partir du schéma du buffer 8 bits 244 (cf. figure 2) , reconstruire sa table de fonctionnement.

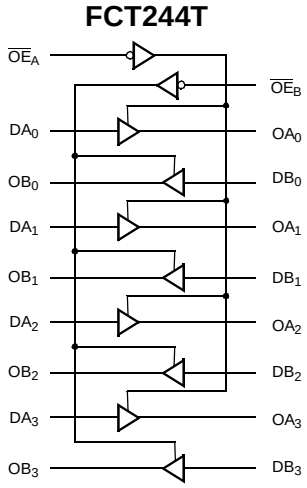


Figure 2 : Buffer 8 bits tristate (244)

Table de fonctionnement du 244

	/OEA	/OEA	D	O
	L	L	L	
	L	L	H	
	H	H	X	

1.3
Dans le tableau des caractèristiques électriques du circuit 244 ci-dessous, que signifie IOZH et IOZL ? Comment est-il possible que la sortie soit à l'état H ou L alors que le buffer est « off » ?

Electrical Characteristics Over the Operating Range

Parameter	Description	Test Conditions		Min.	Typ.[5]	Max.	Unit
V_{OH}	Output HIGH Voltage	V_{CC}=Min., I_{OH}=–32 mA	Com'l	2.0			V
		V_{CC}=Min., I_{OH}=–15 mA	Com'l	2.4	3.3		V
		V_{CC}=Min., I_{OH}=–12 mA	Mil	2.4	3.3		V
V_{OL}	Output LOW Voltage	V_{CC}=Min., I_{OL}=64 mA	Com'l		0.3	0.55	V
		V_{CC}=Min., I_{OL}=48mA	Mil		0.3	0.55	V
V_{IH}	Input HIGH Voltage			2.0			V
V_{IL}	Input LOW Voltage					0.8	V
V_H	Hysteresis[6]	All inputs			0.2		V
V_{IK}	Input Clamp Diode Voltage	V_{CC}=Min., I_{IN}=–18 mA			–0.7	–1.2	V
I_I	Input HIGH Current	V_{CC}=Max., V_{IN}=V_{CC}				5	µA
I_{IH}	Input HIGH Current	V_{CC}=Max., V_{IN}=2.7V				±1	µA
I_{IL}	Input LOW Current	V_{CC}=Max., V_{IN}=0.5V				±1	µA
I_{OZH}	Off State HIGH-Level Output Current	V_{CC} = Max., V_{OUT} = 2.7V				10	µA
I_{OZL}	Off State LOW-Level Output Current	V_{CC} = Max., V_{OUT} = 0.5V				–10	µA
I_{OS}	Output Short Circuit Current[7]	V_{CC}=Max., V_{OUT}=0.0V		–60	–120	–225	mA
I_{OFF}	Power-Off Disable	V_{CC}=0V, V_{OUT}=4.5V				±1	µA

Notes:

1. H = HIGH Voltage Level. L = LOW Voltage Level. X = Don't Care.
2. Unless otherwise noted, these limits are over the operating free-air temperature range.
3. Unused inputs must always be connected to an appropriate logic voltage level, preferably either V_{CC} or ground.
4. T_A is the "instant on" case temperature.
5. Typical values are at V_{CC}=5.0V, T_A=+25°C ambient.
6. This parameter is guaranteed but not tested.
7. Not more than one output should be shorted at a time. Duration of short should not exceed one second. The use of high-speed test apparatus and/or sample and hold techniques are preferable in order to minimize internal chip heating and more accurately reflect operational values. Otherwise prolonged shorting of a high output may raise the chip temperature well above normal and thereby cause invalid readings in other parametric tests. In any sequence of parameter tests, I_{OS} tests should be performed last.

2. Séparateurs de bus

2.1

A partir du schéma du circuit séparateur de bus (tranceiver) 245 ci-dessous, reconstruire sa table de fonctionnement.

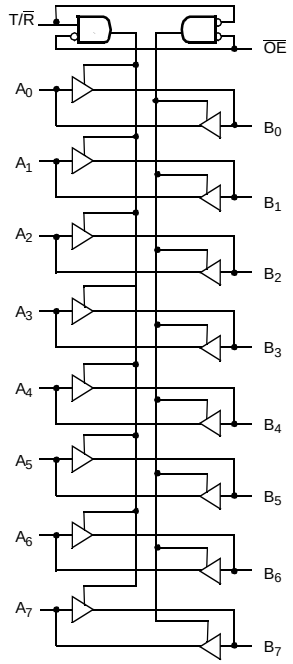


Figure 3 : schèma du 245

Table de fonctionnement du 245 :

/OE	T/R	Fonction
		Bus B vers bus A
		Bus A vers bus B
		HZ

3. SRAM

3.1

A partir du schéma bloc ci dessous de la SRAM 8 KByte CY7C199, retrouvez les configurations des signaux /CE, /WE, et /OE pour que les signaux I/O soit en entrée (écriture), en sortie (lecture), en haute impédance.

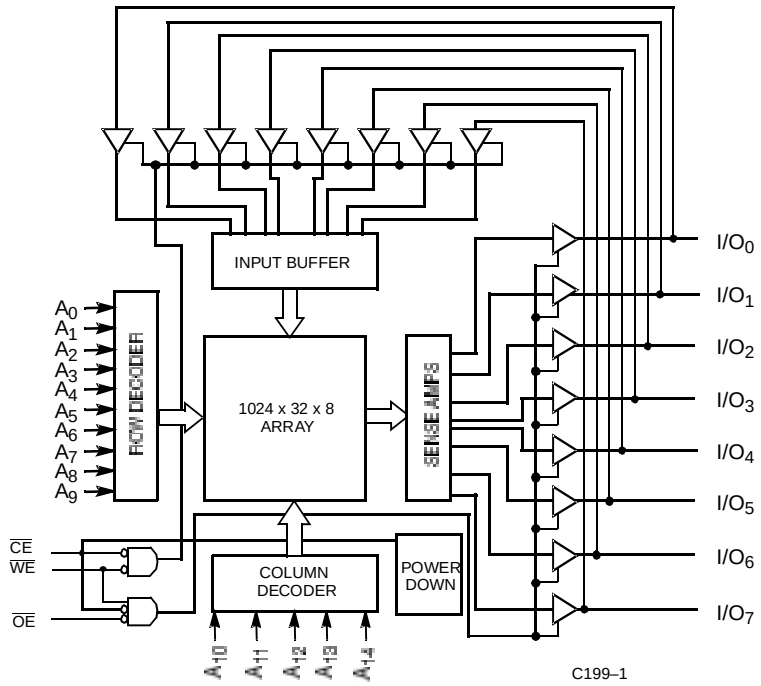


Figure 4 : Schéma bloc de la SRAM CY7C199.

Table de fonctionnement du CY7C199

/CE	/WE	/OE	I/O	Fonction
			IN	Écriture
			OUT	Lecture
			HZ	Sortie HZ
			HZ	Non selectionné

3.2

Indiquez sur la figure 5, sous forme de chronogramme, la séquence à appliquer sur les signaux de contrôle de chacune de ces 3 mémoires pour effectuer un transfert de la mémoire 1 vers la mémoire 2. On considère que le temps d'accès à ces mémoires en lecture comme en écriture est inférieur à 25ns. La fréquence de l'horloge est de 10Mhz. Peut-on doubler la fréquence d'horloge ?

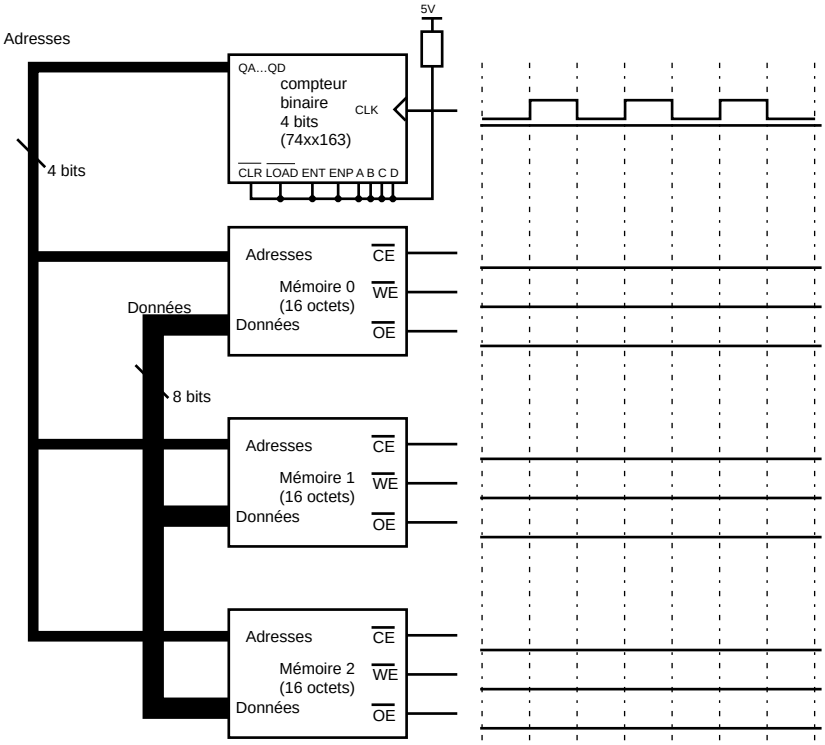


Figure 5 : Transfert de la mémoire 1 vers la mémoire 2.

4. Cycles bus d'un microprocesseur

L'architecture du 68000 simplifié est la décrite sur la figure A1 (cf. Annexe A).
Les cycles bus de lecture et d'écriture sont décrits par un diagramme d'état sur la figure A2. Vous allez vous y reporter pour les mettre sous la forme de chronogrammes.
A titre d'exemple, le premier exercice est résolu.

4.1
Dans le cas où

- Le signal DTACK est directement relié au signal AS, comme le montre la figure 6. Nous appellerons cette configuration "sans état d'attente".
- Le temps d'accès à la mémoire (compté à partir de l'affirmation du signal AS) est supposé inférieur à une période d'horloge.Complétez les chronogrammes suivants. Le premier chronogramme correspond à une lecture, le second correspond à une écriture.

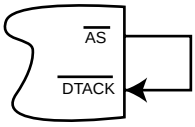


Figure 6: Câblage du signal DTACK sans état d'attente.

-

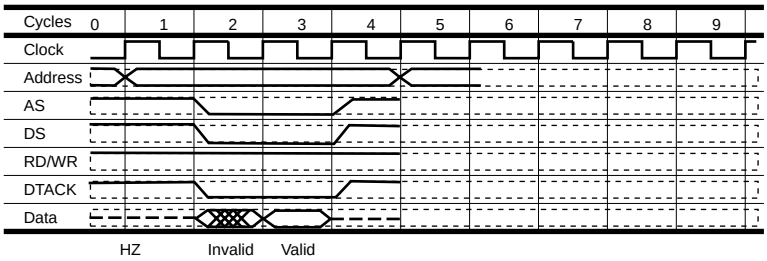


Figure 7 : Chronogramme d'un cycle de lecture sans état d'attente à compléter.

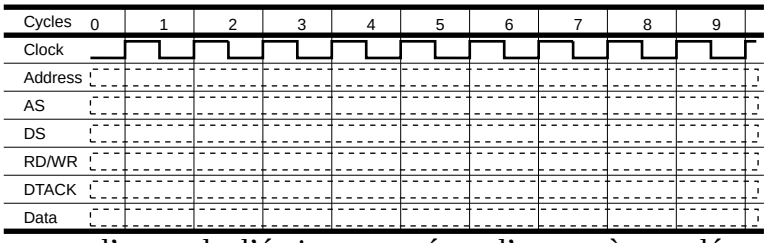


Figure 8 : Chronogramme d'un cycle d'écriture sans états d'attente à compléter.

4.2

Le temps d'accès à la mémoire est-il satisfaisant ? Quelle marge reste-t-il ?

……
……
……
……
……
……

Reprendre les mêmes questions dans le cas où

- Le signal DTACK correspond au signal AS est retardé conformément au schéma de la figure 9.
- Le temps d'accès à la mémoire est de trois périodes d'horloge.

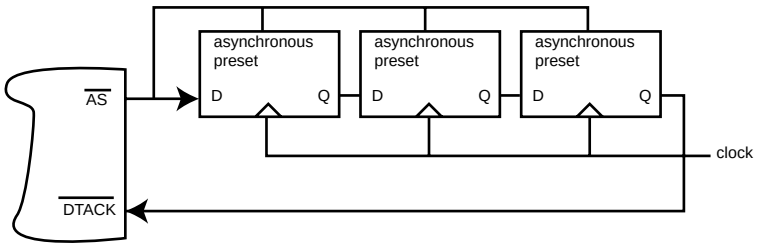


Figure 9 : Câblage du signal DTACK "avec état d'attente". Le signal preset (mise à un) est supposé asynchrone.

4.3

Quel est le fonctionnement de ce montage ? De combien de périodes d'horloge sont retardés les fronts descendants ? montants ?

4.4

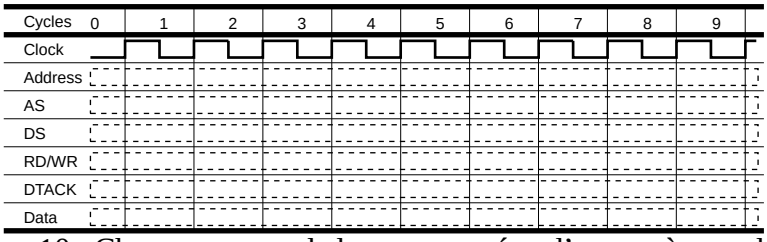


Figure 10 : Chronogramme de lecture avec état d'attente à compléter.

4.5

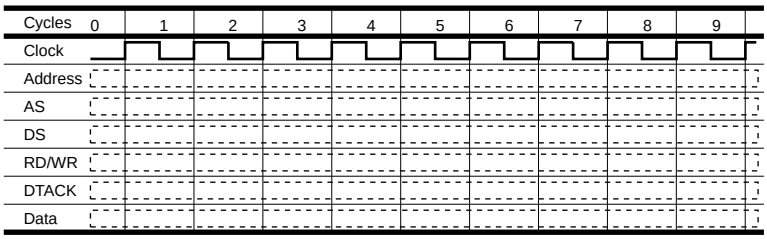


Figure 11 : Chronogramme d'un cycle d'écriture avec état d'attente à compléter.

4.6

Le temps d'accès à la mémoire est-il satisfaisant ?

……
……
……

4.7

Pourquoi parle-t-on d'état d'attente ? Quel en est le rôle ?

……
……
……
……

4.8

En 1999, dans un PC, la fréquence interne du microprocesseur avoisine les 450MHz alors que les DRAM ont un temps d'accès approximatif (en accès aléatoire, c'est-à-dire à des adresses non consécutives) de 50 ns. Combien y a-t-il d'état d'attente pour accéder à la DRAM ? Est-ce acceptable ? Quelle solution existe-t-il ?

4.9

Comparer les chronogrammes obtenus dans les questions précédentes avec ceux du 68000.

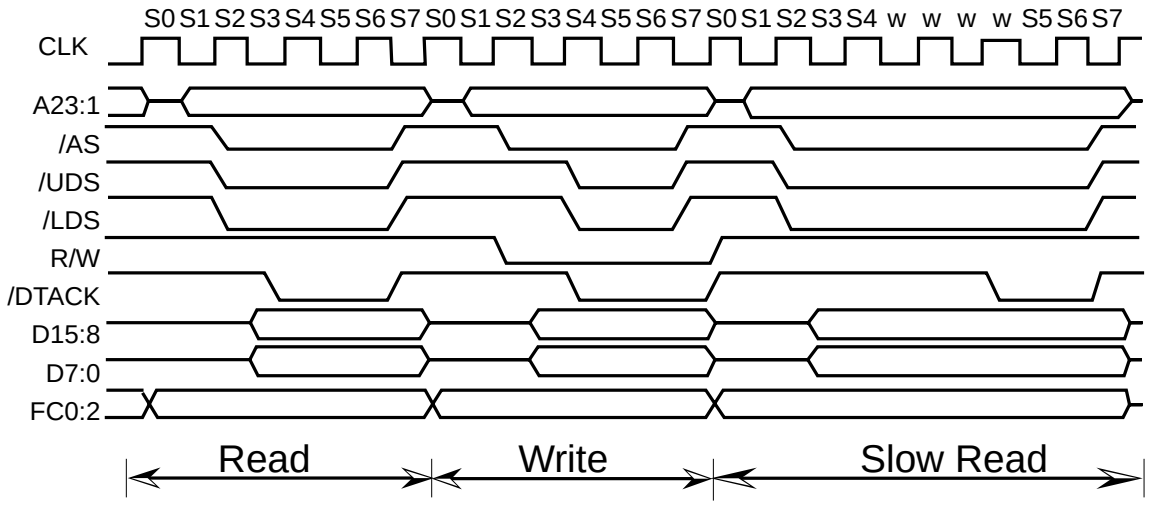


Figure 12 : cycles de lecture et d'écriture du 68000.

Annexe A : Le 68000 simplifié

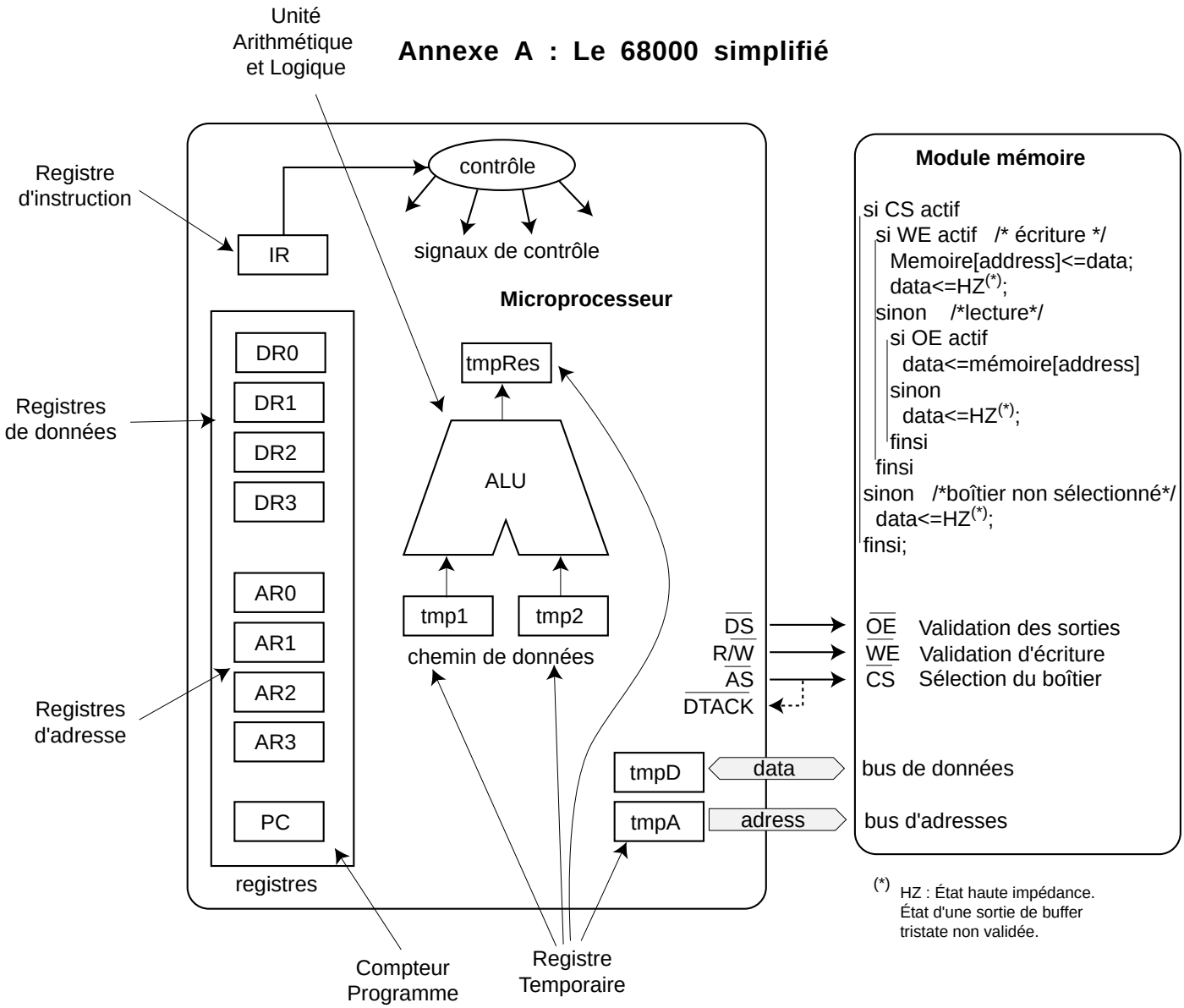


Figure A1 : Architecture simplifiée du 68000.

Éléments du microprocesseur

L'unité arithmétique et logique (ALU) réalise les opérations arithmétiques entières (addition, soustraction, multiplication, division) et les opérations logiques (et, ou, non, ou exclusif, décalage).
Le compteur programme (PC) est un pointeur vers la prochaine instruction à exécuter.
Le registre d'instruction contient l'instruction en cours d'exécution.
Les registres sont utilisés par le micro pour mémoriser temporairement des données ou des adresses fréquemment utilisées.

Mémoire

La mémoire contient les données et les instructions des programmes. Elle mémorise 2^N mots de données repérés par leur adresse de 0 à 2^N-1. N est la largeur en nombre de bit du bus d'adresse de la mémoire. La largeur de son bus de données détermine la taile de chaque mots. Nous utiliserons fréquemment des mémoires organisées en octets (mot de huit bits) dont la taille est de quelques Kilo Octets (KB). Le bus de données comprendra donc 8 bits et le bus d'adresse un peu plus de 10 bits.

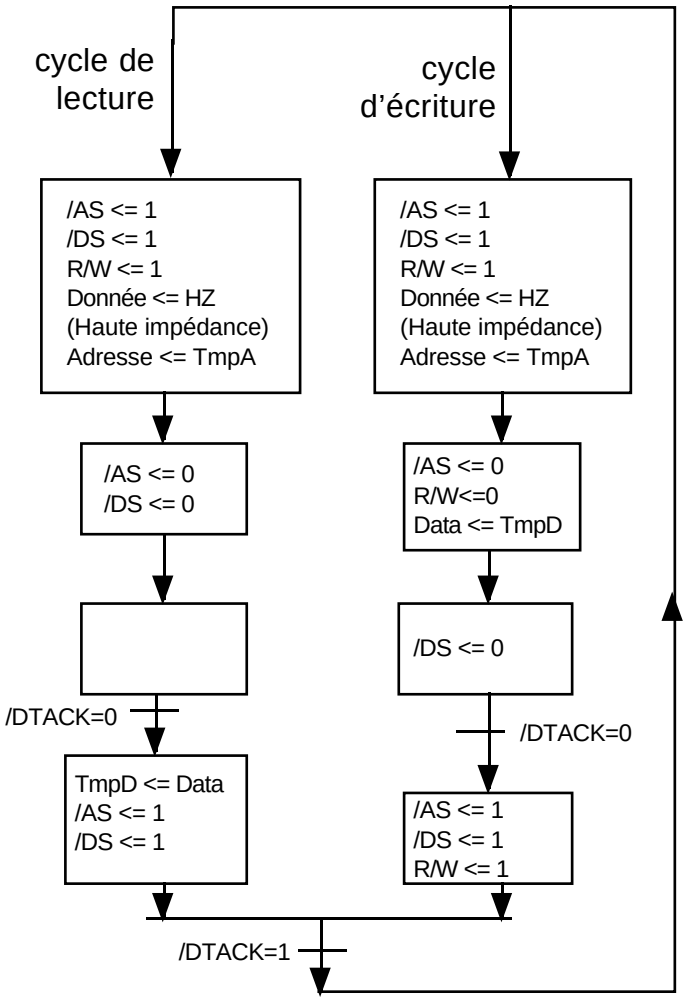


Figure A2 : Cycles de lecture et d'écriture du 68000 simplifié

Signaux

En affirmant le signal **AS**, le micro indique que son bus d'adresse est stable.

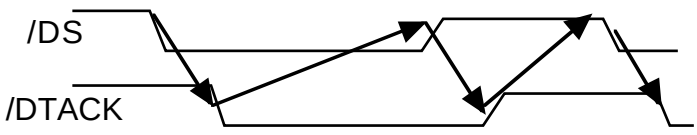
Lors d'un cycle de lecture, le micro met le signal **R/W** a l'état H. Lors d'un cycle d'écriture, le micro met le signal **R/W** à l'état L.

En affirmant le signal **DS** (validation du bus de donnée), le micro indique qu'il est prèt pour une transaction avec la mémoire. S'il s'agit d'un cycle de lecture, cela signifie qu'il a mis ses sorties vers le bus de données en haute impédance. S'il s'agit d'une écriture, cela signifie que la donnée est prète sur le bus.

Un front déscendant du signal **DTACK** (accusé de reception de donnée) informe le micro que la mémoire a eut le temps de répondre à sa requète. Lors d'une écriture, la donnée est écrite dans la mémoire, lors d'une lecture, la donnée est prète sur le bus de donnée.

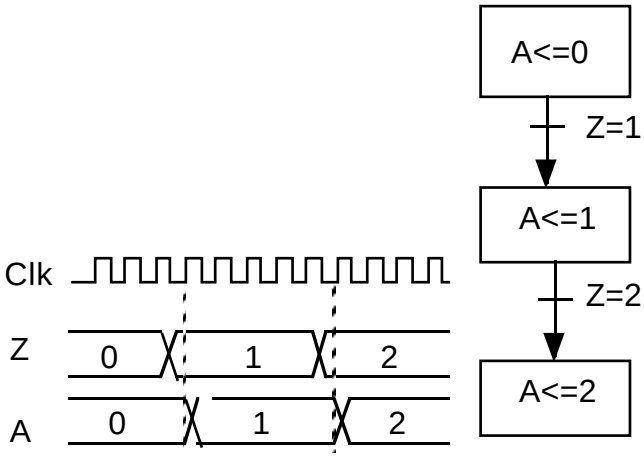
Handshake

Le couple DS et DTACK forme un *handshake* (poignée de main). Ces deux signaux permettent de synchroniser le micro et la mémoire (ou un périphérique). Ce procédé est couramment employé pour synchroniser des automates. Chaque transition sur un des signaux doit attendre la transition de son interlocuteur.



Automate synchrone

Le séquenceur représenté par ce diagramme d'état est **synchrone**. Toutes les sorties sont synchrones avec l'horloge et toutes les entrées sont échantillonnées sur les fronts montants de l'horloge.



CACHAN
GEii
Électronique

TP d'informatique industrielle n° 1
Geii1 – Module 2

semaine du 18/10 au 23/10/99
Outils de développement croisés

Objectifs

Objectifs généraux
Maîtriser les outils de développement croisé pour le kit ColdFire (SBC5206)
Objectifs précis
- Créer un exécutable :
- Rôle des différents fichiers (makefile, fichier de commande du linker, startup.icd, mem, map)
- Procédure pour ajouter un fichier source
- Procédure pour créer/utiliser une librairie
- Utiliser le fichier MAP /MEM pour connaître le résultat de l'édition de lien
    - Savoir si un objet a été chargé ou non
    - Trouver l'adresse du symbole d'une fonction ou d'une variable
- Identifier l'outil qui a provoqué d'une erreur et sa cause probable : préprocesseur, compilateur, éditeur de liens, exécution.
- Télécharger par le port ICD ou par la liaison série.
- Exécuter en pas à pas.
- Poser un point d'arrêt et visualiser la trace
- Visualisation du programme et des données en mémoire
- Visualisation des registres

Exercices

1 Etapes de la création d'un exécutable

À partir d'un unique fichier source C, compiler et éditer les, charger par le port DEBUG et exécuter.

```
//  Un petit filtre RIF
//

#include <cf_tk99a_io.h>

typedef       signed char    signal_t[16];


void initalise (signal_t);
void filtre (signal_t entree, signal_t sortie);
void display (signal_t);
signal_t      signal_1;
signal_t      signal_2;

//sin = sin(Fe/16)
const signal_t sinus = {0,52,95,122,127,111,75,27,-27,-75,-111,-127,-122,-95,-52,0};

void main (void){
```

```
	initalise (signal_1 ) ;
	filtre (signal_1, signal_2 );
	display (signal_1);
	display (signal_2);
}

//   sin (Fe/16)) + sin (Fe /4)
void initalise (signal_t s){
	unsigned long int i;
	for (i=0;i<16;i++) {
		s[i] = sinus[i]+sinus[(4*i)%16];
	}
}

#define MIN(x,y) ((x<y) ? x : y)
#define MAX(x,y) ((x>y) ? x : y)

//filtre FIR gaussien sigma = 0.6
void filtre (signal_t entree, signal_t sortie){
	long int j,i;
	long int x[16];

	int gauss[] =  {2567 ,15527, 45723, 65536 , 45723 ,15527,2567};
	// gauss  = round(exp(-((t-3)*0.6)^2)*2^16)

	for (i=0;i<16;i++)
		x[i] = 0 ;


	for (i=-3;i<=3;i++)
		for (j= MAX(0,i);j<MIN(16,16+i);j++)
			x[j] += entree[j-i]*gauss[i+3];

	for (i=0;i<16;i++)
		sortie[i]=x[i]>>16;
}
void display (signal_t s){
	int i;
	for(i=0;i<16;i++)cout << s[i] ;
	cout << endl;
}
```

2 Compiler et vérifier ce programme.

```
//////////////////  top.cpp
// Ce programme cherche un caractère “z“  parmi les 256 premiers octets de la mémoire.

#include <cf_tk99a_io.h>
#include "prototypes.h"

int Erreur ;

int main (void){
	char y;
	char z = 78;
```

```
	Erreur = 0;
	y = search (z);
	if (Erreur != 0 ){
		cout <<"Rien trouve"<<endl;
	}
	else{
		cout << "Trouve en "<< (int)y << "eme position " << endl;
	}
	cout << "Fin du programme"<<endl;
}

//////////////////  prototype.h

extern char search (char);

//////////////////    my_fonc.cpp
#include <prototype.h>

extern int Erreurs ;

char search (char c){
char i;
char * ptr = 0;
for (i=0;i<256;i++)
	if (   ptr[i] = c)
		return i;
	else {
		Erreurs = 1;
		return -1;
	}
}
```

3 Le moniteur, exécution en pas à pas, modification de la mémoire.

Écrire un programme qui attend que l'octet à l'adresse 0x30000 (hexadécimale) passe à 1. L'octet est initialisé à zéro et le moniteur est utilisé pour changer sa valeur au cours de l'exécution.

CACHAN
GEii
Électronique

TP d'informatique industrielle n° 2
semaine du 25/10 au 30/10/99
TD d'informatique industrielle n° 3
semaine du 8/11 au 13/11/99
Geii1 – Module 2
Jeu d'instructions et modes d'adressage

Objectifs

Objectifs généraux
Savoir lire l'assembleur généré par le compilateur. Les aspects liés à la pile seront vus au prochain TD.
Connaître les principales instructions et les principaux modes d'adressage.
Objectifs précis
Syntaxe d'une instruction assembleur
- Étiquette, mnémonique, source, destination

Jeu d'instruction
- MOVE, MOVEA, LEA
- ADD, SUB, MUL,ANDI, ORI
- CMP , BRA, BCC

Mode d'adressage
- registre
- immédiat
- direct
- indirect par registre d'adresse
- indirect avec déplacement
- indexé
- indexé avec déplacement

Formats 8, 16 ou 32 bits

Exercices

1 Modes d'adressages.
À l'aide du jeu d'instructions fourni en annexe et de quelques exemples, examinez les mécanismes des modes d'adressage par registres, indirectes, indirectes post-incrémentés et pré-décrémentés, indirectes avec déplacement, indirectes indexés avec déplacement.

2 Suivre l'évolution des registres dans le programme ci-dessous

```
/* exo1asm.s */
/* td tp asm mode d'@ */
              .text
              .globaldebut
debut :
              /* byte word et long */
              move.b #0x12,d0
              move.w #0x3456,d1
              move.w #0x789a,d2
              move.l #0x89abcdef,d3

              /* registres d' adresse */
              move.w d0,a0
```

```
                move.w d1,a1
                move.w d2,a2
                move.w d3,a3

                move.l d0,a4
                move.l d1,a5
                move.l d2,a6
                move.l d3,a7

                /* signe */
                move.b #0x12,d0
                ext.w  d0
                ext.l  d0

                move.b #0x12,d0
                ext.w  d0
                ext.l  d0


                move.l #0x97,d0
                ext.l  d0

ici :           bra.s ici
```

En analysant le programme ci-dessous implanté à partir de l'adresse 10000, indiquez le contenu des bus d'adresses et de données à chaque instruction exécutée.

```
   Fichier source :
                move.w #1234,d0
                lea    0x20000,a0
                move.b  d0,(a0)+
                move.w d0,(a0)

   Listing après l'édition  des liens :
   10000:     303c 04d2            movew #1234,%d0
   10004:     41f9 0002 0000       lea 20000 ,%a0
   1000a:     10c0                 moveb %d0,%a0@+
   1000c:     3080                 movew %d0,%a0@
```

3 On veut observer la configuration suivante sur le bus du microprocesseur :
Adresse = 0001 0300
Donnée = 1234 zzzz
R/W = 0
/WE3 = 1
/WE2 = 1
/WE1 = 0
/WE0 = 0

Quel programme (ou quelle instruction) faut-il écrire en assembleur ? En C ?

4 Dans le programme ci-dessous, expliquez le rôle des directives .text, .data, .global, .asciz, ds et dc.

```
                .text
                .global debut
debut :
                lea     source,a0
                lea     dest,a1

boucle :
                move.b (a0)+,(a1)+
                bne.s  boucle

                nop
                nop
                nop

ici :           bra.s  ici

                .data
                .global source
                .global dest
source :        .asciz "Il etait une fois"

                .align 4
dest :                  ds.b    20

                .global top
top :           dc.b    0
```

5 Tapez le programme de la question 4. Quelles sont les adresses des différents tableaux et variables ? Suivez l'évolution des différents registres et cases mémoires.

6 Créez un programme qui initialise la case mémoire d'adresse 10100 de type word avec la valeur 55AA en utilisant les modes d'adressage absolu et indirect.
Quels auraient été les programmes C équivalents ?

7 Créez un programme en assembleur qui initialise les 20 éléments d'un tableau de type word avec une valeur quelconque en utilisant les modes d'adressage indirectes post-incrémentéés et indirectes avec déplacement.
Quels auraient été les programmes C équivalents ?

8 Créez un programme qui additionne deux tableaux de type word de 10 éléments et qui range le résultat dans un tableau de type long.
Quel aurait été le programme C équivalent ?

Annexe
Jeu d'instructions simplifié du cœur du ColdFire

Description simplifiée du cœur du Coldfire

Modèle de programmation (registres)

8 registres de données (32 bits)

D0	D1	D2	D3	D4	D5	D6	D7

8 registres d'adresses (32 bits)

A0	A1	A2	A3	A4	A5	A6	A7

Compteur programme (32 bits)

PC

Indicateur de conditions (4 bits)

CC

bit	3	2	1	0
cc	N	Z	V	C
condition/résultat	negatif	nul	dépassement (signé)	retenue générée

Jeu d'instructions

Additions

Mnémonique	**OPCode**	**Registre**	**OpMode**	**Mode**	**Registre**	**Description**
ADD ea,Dn	1101	xxx	010	mmm	yyy	Dxxx <- Dxxx+ *ea
ADD Dn,ea	1101	xxx	110	mmm	yyy	*ea <- *ea+Dxxx
ADDA ea,An	1101	xxx	111	mmm	yyy	An<-An+ *ea
ADDI #data,Dn	0000	011	010	000	yyy	Dn<-Dn+data
	Upper	Word	of	Data		
	Lower	Word	of	Data		
ADDQ	0101	Data	010	mmm	yyy	*ea <- data + *ea

Soustractions

Mnémonique	**OPCode**	**Registre**	**OpMode**	**Mode**	**Registre**	**Description**
SUB ea,Dn	1001	xxx	010	mmm	yyy	Dxxx <- *ea - Dxxx
SUB Dn,ea	1001	xxx	110	mmm	yyy	*ea <- Dxxx - *ea
SUBI #data,Dn	0000	010	010	000	yyy	Dxxx<-Dxxx-data
	Upper	Word	of	Data		
	Lower	Word	of	Data		
SUBQ	0101	Data	010	mmm	yyy	*ea <- data - *ea

Comparaison

Mnémonique	OPCode	Registre	OpMode	Mode	Registre	Description
COMP ea,Dn	1011	xxx	010	mmm	yyy	CC<- Dxxx - *ea
COMPA ea,An	1011	xxx	111	mmm	yyy	CC <- Axxx - *ea
COMPI #data,Dn	0000	110	010	000	yyy	CC<-Dxx-data
	Upper	Word	of	Data		
	Lower	Word	of	Data		

Multiplications

Mnémonique	OPCode	Registre	OpMode	Mode	Registre	Description
MULS ea,Dn	1100	xxx	111	mmm	yyy	Dxxx <- *ea * Dxxx
MULU ea,Dn	1100	xxx	011	mmm	yyy	Dxxx <- *ea * Dxxx

Mouvements de données

Mnémonique	OPCode	Mode (dest.)	Registre (dest.)	Mode (sour.)	Registre (sour.)	Description
MOVE ea,ea	11 ss	mmm	yyy	mmm	yyy	dest. <- sour.
MOVEA ea,ea	idem	mmm	yyy	mmm	yyy	dest. <- sour.

ss pour size :
01 : 8 bits, octet (b)
10 : 16 bits, mot (w),
10 : 32 bits, long word (l)

Chargement d'une adresse

Mnémonique	OPCode	Registre	OpMode	Mode	Registre	Description
LEA ea,An	0100	xxx	111	mmm	yyy	Axxx <- ea

Ici il s'agit bien de ea et non de *ea. LEA est à l'assembleur ce que & est au langage C.

Branchement relatif

Mnémonique	OPCode	cond (cc)	déplacements sur 8 bits	Description
BRA label	0110	0000	dddd dddd	PC <- PC + depl
BRA label	0110	cccc	dddd dddd	si cc alors PC <- PC + depl

Saut

Mnémonique	OPCode	Registre	OpMode	Mode	Registre	Description
JMP dest	0100	111	111	mmm	yyy	PC <- ea

Modes d'adressage

Mode d'adressage	mode mmm	reg. yyy	syntaxe	description
par registre de données	000	N° reg	Dn	Contenu de Dn
par registre d'adresses	001	N° reg	An	Contenu de An

indirecte	010	N° reg	(An)	Memoire[An]
post-incrémenté	011	N° reg	(An)+	Mémoire[An + +]
pré-décrémenté	100	N° reg	-(An)	Mémoire[- - An]
indirecte avec déplacement	101	N° reg	(d16,An)[1]	Memoire[An+dépl]
indexé avec déplacement	110	N° reg	(d8,An,Xn)[2]	Mem[An+dépl+Xn]
absolu court (adresse 16 bits)	111	000	address [3]	Mem[address]
absolu long(adresse 32 bits	111	001	address[4]	Mem[address]
immédiat	111	100	#data	valeur data
relatif au PC	111	010	(d16,PC)[5]	Memoire[PC+dépl]
indéxé et relatif au PC	111	011	(d8,PC,Xn)[6]	Mem[PC+dépl+Xn]

Éventuel mot d'extension pour les adressages indexés (d8,An,Xiii) ou (d8,PC,Xiii).

D/A	Reg	W/LW	scale	EV	d8
b	iii	1	ss	0	dddd dddd
X est un registre de données (0) ou d'adresses (1)	registre d'indexe (X)		byte(00), word(01), ou long word (11)		déplacement sur 8 bits (signé)

Codage des conditions de branchement (cc)

condition	cccc	syntaxe	signification
toujours vrai	0000		true
toujours faux	0001		false
pas de retenue détéctée	0100	CC	carry clear
retenue détéctée	0101	CS	carry set
égalité détectée	0111	EQ	equal
supérieur ou égal	1100	GE	greater or equal
supérieur	1110	GT	greater than
inférieur ou égal	1111	LE	less or equal
inférieur	1101	LT	less than
différent	0110	NE	not equal
pas de dépassement (signé) détecté	1000	VC	overflow clear
dépassement (signé) détecté	1001	VS	overflow set

[1] Le déplacement d16 est donné par un mot d'extension sur 16 bits.
[2] Le mot d'extension suit le format donné par le tableau qui suit.
[3] L'adresse est donnée par un mot d'extension sur 16 bits.
[4] L'adresse est donnée par deux mot d'extension sur 16 bits chacun.
[5] Le déplacement est donné par un mot d'extension sur 16 bits
[6] Le mot d'extension suit le format donné par le tableau qui suit.