ELECINF 102 : Processeurs et Architectures Numériques

Le Nano processeur

Tarik Graba
tarik.graba@telecom-paris.fr

Plan

Un Programme

- Un processeur exécute un programme : une suite d'instructions sur un ensemble de données.

```
0.  112 + 3
1.  résultat précédent - 4
2.  si le résultat est nul, passer à l'étape 6,
    sinon continuer
3.  3 * 4
4.  résultat précédent + 9
5.  ouvrir la fenêtre
6.  résultat de l'étape 2 - 15
7.  passer à l'étape 12
8.  …
```

Un Programme

Données

- Les opérandes
  - `3,4,112,…`
- opérandes implicites
  - `Résultat précédent, résultat de l'étape 2, …`

Un Programme

Données

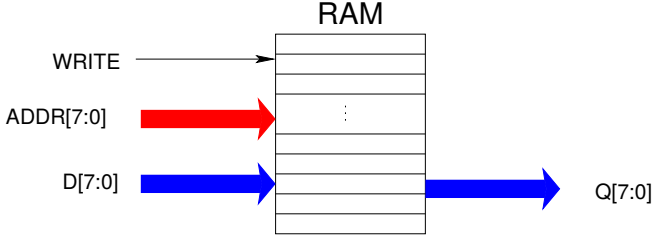
- Les opérandes
  - `3,4,112,…`
- opérandes implicites
  - `Résultat précédent, résultat de l'étape 2, …`

Instructions

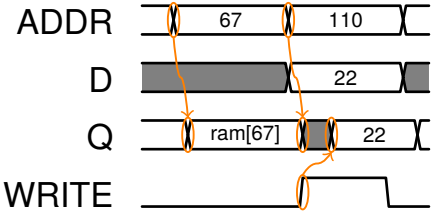
- Opérations arithmétiques et logiques
  - $+, -, \times, /, \ldots$
  - `And, Or, Not, Xor, …`
- Les tests
  - `Si le résultat précédent est nul…`
- Les sauts
  - Conditionné par un test : `Si …alors passer à l'étape 6`
  - Non conditionné : `Passer à l'étape …`
- instructions spéciales
  - `ouvrir la fenêtre, faire clignoter la LED, …`

La mémoire

- Une table
  - La position dans la table c'est l'adresse
- On ne peut lire ou modifier qu'une seule case à la fois
  - Chaque case contient un mot



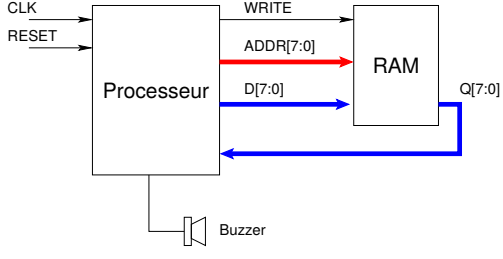
La mémoire
mémoire asynchrone



- un ``bus'' d'adresse
- la valeur sélectionnée apparait sur la sortie (Q)
- La donnée à écrire est présentée dur l'entrée (D)
- Un signal (WRITE) pour indiquer qu'on modifie la valeur

Le Système

- Le système est composé :
  - d'un processeur
  - d'une RAM (256 mots de 8 bits $\Rightarrow$ 8 bits d'adresse)
  - d'un périphérique (le buzzer)



Plan

1e étape : Automate linéaire basique

- Deux instructions :

Code	Instruction
`00000100`	`addition`
`00000110`	`soustraction`

- Structure linéaire de la mémoire

adresse	type du mot stocké	exemple
`0`	instruction	`+`
1	donnée (premier opérande)	`3`
2	donnée (deuxième opérande)	`4`
3	donnée (résultat)	`X`
4	instruction	`-`
5	donnée (premier opérande)	`12`
6	donnée (deuxième opérande)	`8`
7	donnée (résultat)	`X`

1e étape : Automate linéaire basique

- Deux instructions :

Code	Instruction
`00000100`	`addition`
`00000110`	`soustraction`

- Structure linéaire de la mémoire

adresse	type du mot stocké	exemple
`0`	instruction	`+`
1	donnée (premier opérande)	3
2	donnée (deuxième opérande)	4
3	donnée (résultat)	7
4	instruction	`-`
5	donnée (premier opérande)	12
6	donnée (deuxième opérande)	8
7	donnée (résultat)	4

Organisation du programme en mémoire
Attention

- La mémoire contient le programme et les données
- Les données et les instructions sont imbriquées et non différenciées
  - La valeur de l'instruction
  - La valeur de la donnée
- Mais…connaissance implicite de l'organisation

adresse	donnée
0	00000100
1	00000011
2	00000100
3	00000111
4	00000110
5	00001100
6	00001000
7	00000100

1e étape : Automate linéaire basique

Proposez une architecture, *i.e.* comment construire ce processeur ?

1e étape : Automate linéaire basique

l'architecture

- Un compteur programme

1e étape : Automate linéaire basique

l'architecture

- Un compteur programme
- Un additionneur-soustracteur

1e étape : Automate linéaire basique

l'architecture

- Un compteur programme
- Un additionneur-soustracteur
- Un registre pour l'instruction

1e étape : Automate linéaire basique

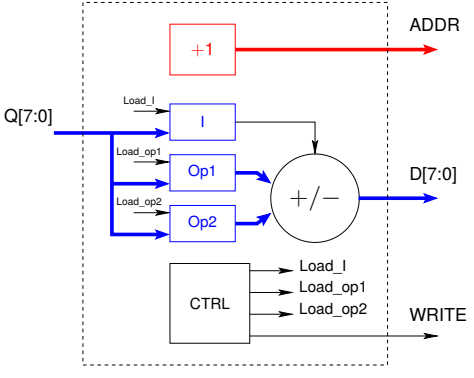
l'architecture

- Un compteur programme
- Un additionneur-soustracteur
- Un registre pour l'instruction
- Un registre pour chaque opérande

1e étape : Automate linéaire basique

l'architecture

- Un compteur programme
- Un additionneur-soustracteur
- Un registre pour l'instruction
- Un registre pour chaque opérande
- Un contrôleur



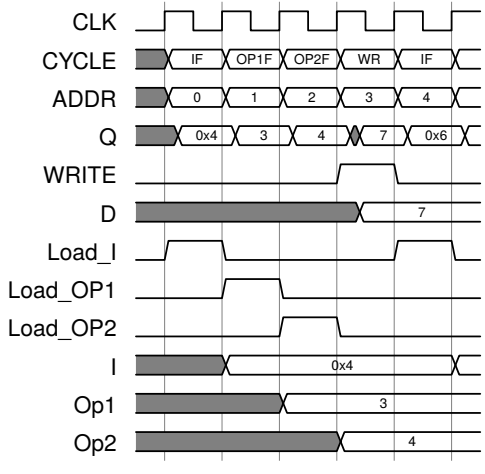
1e étape : Automate linéaire basique

- Comment se séquencent les opérations ?

adresse	type du mot stocké	exemple
0	instruction	+
1	donnée (premier opérande)	3
2	donnée (deuxième opérande)	4
3	donnée (résultat)	X
4	instruction	-
5	donnée (premier opérande)	12
6	donnée (deuxième opérande)	8
7	donnée (résultat)	X

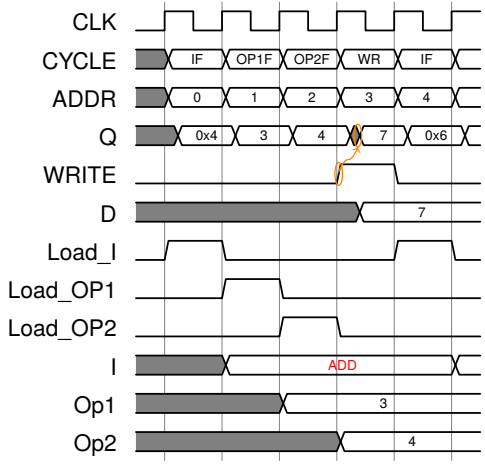
1e étape : Automate linéaire basique

Chronogramme



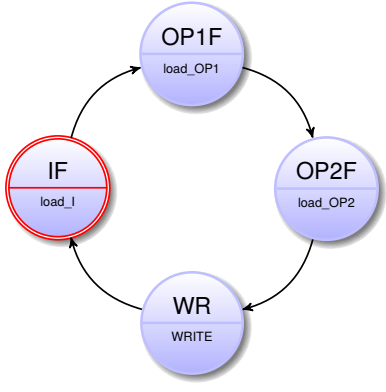
1e étape : Automate linéaire basique

Chronogramme

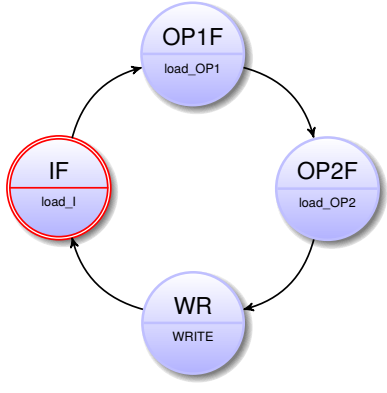


1e étape : Automate linéaire basique

le contrôleur



1e étape : Automate linéaire basique
le contrôleur

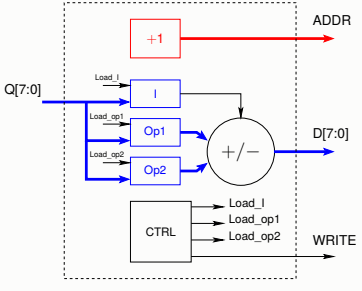


- Les 2 bits de poids faible de l'adresse permettent de différencier les états.

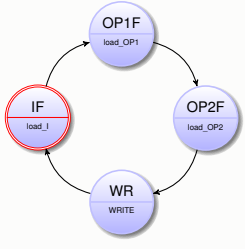
1e étape : Automate linéaire basique

Récapitulons

l'architecture



le contrôleur



2e étape : Automate avec accumulateur

- Comment chaîner des calculs (3+4−6+…) ?

2e étape : Automate avec accumulateur

- Comment chaîner des calculs (3+4−6+…) ?
- Mémoriser les résultats intermédiaires :

2e étape : Automate avec accumulateur

- Comment chaîner des calculs (3+4−6+…) ?
- Mémoriser les résultats intermédiaires :
  - Ajouter un registre pour garder le dernier résultat

2e étape : Automate avec accumulateur

- Comment chaîner des calculs (`3+4-6+…`) ?
- Mémoriser les résultats intermédiaires :
  - Ajouter un registre pour garder le dernier résultat
  - Toutes les instructions agissent sur un élément venant de la mémoire et le registre qui contient le résultat précédent

2e étape : Automate avec accumulateur

- Comment chaîner des calculs (3+4−6+…) ?
- Mémoriser les résultats intermédiaires :
  - Ajouter un registre pour garder le dernier résultat
  - Toutes les instructions agissent sur un élément venant de la mémoire et le registre qui contient le résultat précédent
  - Ajouter des instructions de chargement et enregistrement en RAM de ce registre

2e étape : Automate avec accumulateur

- Comment chaîner des calculs (3+4−6+…) ?
- Mémoriser les résultats intermédiaires :
  - Ajouter un registre pour garder le dernier résultat
  - Toutes les instructions agissent sur un élément venant de la mémoire et le registre qui contient le résultat précédent
  - Ajouter des instructions de chargement et enregistrement en RAM de ce registre
- Ce registre est appelé accumulateur

2e étape : Automate avec accumulateur

- Ajouter de nouvelles instructions

code (binaire 8 bits)	instruction
00000001	XOR
00000010	AND
00000011	OR
00000100	addition
00000110	soustraction
00001010	load
00001011	store

2e étape : Automate avec accumulateur

- Nouvelle structure du programme

adresse	type du mot stocké	exemple
0	instruction	`load`
1	donnée	3
2	instruction	`+`
3	donnée	4
4	instruction	`-`
5	donnée	1
6	instruction	`store`
7	donnée	`X`

2e étape : Automate avec accumulateur

- Nouvelle structure du programme

adresse	type du mot stocké	exemple
0	instruction	`load`
1	donnée	3
2	instruction	`+`
3	donnée	4
4	instruction	`-`
5	donnée	1
6	instruction	`store`
7	donnée	6

2e étape : Automate avec accumulateur

- Modifiez l'architecture précédente
  - Quels éléments faut--il ajouter ou supprimer ?

2e étape : Automate avec accumulateur

l'architecture

- Un compteur programme

2e étape : Automate avec accumulateur

l'architecture

- Un compteur programme
- Une ALU (`add`, `sub`, `or`, `and`, `xor`)

2e étape : Automate avec accumulateur

l'architecture

- Un compteur programme
- Une ALU (`add`, `sub`, `or`, `and`, `xor`)
- Un registre pour l'instruction

2e étape : Automate avec accumulateur

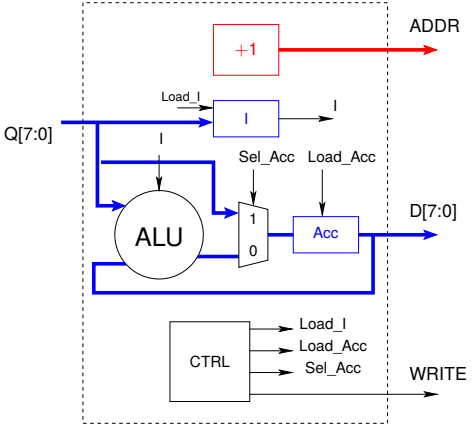
l'architecture

- Un compteur programme
- Une ALU (`add`, `sub`, `or`, `and`, `xor`)
- Un registre pour l'instruction
- Un registre pour l'accumulateur
- La possibilité de choisir l'entrée de ce registre

2e étape : Automate avec accumulateur

l'architecture

- Un compteur programme
- Une ALU (`add`, `sub`, `or`, `and`, `xor`)
- Un registre pour l'instruction
- Un registre pour l'accumulateur
- La possibilité de choisir l'entrée de ce registre
- Un contrôleur



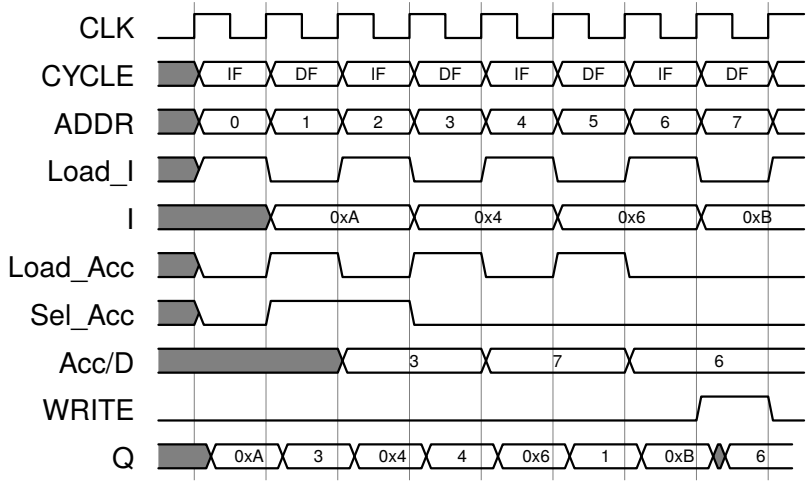
2e étape : Automate avec accumulateur

- Comment se séquencent les opérations ?

adresse	type du mot stocké	exemple
0	instruction	`load`
1	donnée	3
2	instruction	`+`
3	donnée	4
4	instruction	`-`
5	donnée	1
6	instruction	`store`
7	donnée	X

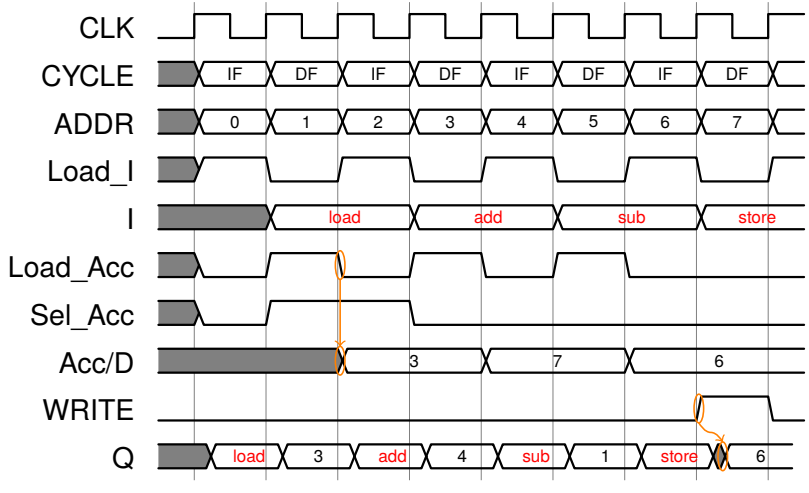
2e étape : Automate avec accumulateur

Chronogramme

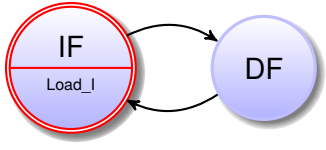


2e étape : Automate avec accumulateur

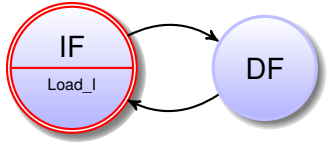
Chronogramme



2e étape : Automate avec accumulateur
le contrôleur



2e étape : Automate avec accumulateur
le contrôleur



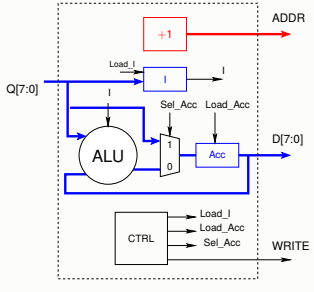
- Le bit de poids faible de l'adresse permet de différencier les états.

$$
\begin{aligned}
\text{Sel\_Acc} &= (\, I = \text{load} \,) \\
\text{Load\_Acc} &= (\, I \neq \text{store} \,) \;\&\&\; (\, \text{Etat} = \text{DF} \,) \\
\text{WRITE} &= (\, I = \text{store} \,) \;\&\&\; (\, \text{Etat} = \text{DF} \,)
\end{aligned}
$$

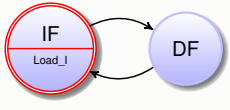
2e étape : Automate avec accumulateur

Récapitulons

l'architecture



le contrôleur



Sel_Acc = (I = load)
Load_Acc = (I ≠ store) && (Etat = DF)
WRITE = (I = store) && (Etat = DF)

3e étape : Automate avec accumulateur et indirection

- Si on veut :
  - Utiliser le même code sur des données différentes
  - Utiliser les mêmes données avec des codes différents
  - Utiliser le code sur des données générées par le code (on ne les connaît pas au chargement du programme en RAM)

3e étape : Automate avec accumulateur et indirection

- Si on veut :
  - Utiliser le même code sur des données différentes
  - Utiliser les mêmes données avec des codes différents
  - Utiliser le code sur des données générées par le code (on ne les connaît pas au chargement du programme en RAM)
- On doit :
  - Modifier les instructions de façon à avoir le code de l'instruction suivi de l'adresse de l'opérande (plutôt que sa valeur).

3e étape : Automate avec accumulateur et indirection

adresse	type du mot stocké	exemple	zone
0	instruction	load	zone de code
1	adresse de l'opérande	100	
2	instruction	+	
3	adresse de l'opérande	101	
4	instruction	store	
5	adresse de l'opérande	103	
6	instruction	load	
7	adresse de l'opérande	100	
8	instruction	-	
9	adresse de l'opérande	102	
10	instruction	store	
11	adresse de l'opérande	104	
…	…	…	
100	donnée	3	zone de données
101	donnée	4	
102	donnée	1	
103	donnée	X	
104	donnée	X	
…	…	…	

3e étape : Automate avec accumulateur et indirection

adresse	type du mot stocké	exemple	zone
0	instruction	load	zone de code
1	adresse de l'opérande	100	
2	instruction	+	
3	adresse de l'opérande	101	
4	instruction	store	
5	adresse de l'opérande	103	
6	instruction	load	
7	adresse de l'opérande	100	
8	instruction	-	
9	adresse de l'opérande	102	
10	instruction	store	
11	adresse de l'opérande	104	
…	…	…	
100	donnée	3	zone de données
101	donnée	4	
102	donnée	1	
103	donnée	7	
104	donnée	X	
…	…	…	

3e étape : Automate avec accumulateur et indirection

adresse	type du mot stocké	exemple	zone
0	instruction	load	zone de code
1	adresse de l'opérande	100	
2	instruction	+	
3	adresse de l'opérande	101	
4	instruction	store	
5	adresse de l'opérande	103	
6	instruction	load	
7	adresse de l'opérande	100	
8	instruction	-	
9	adresse de l'opérande	102	
10	instruction	store	
11	adresse de l'opérande	104	
…	…	…	
100	donnée	3	zone de données
101	donnée	4	
102	donnée	1	
103	donnée	7	
104	donnée	2	
…	…	…	

3e étape : Automate avec accumulateur et indirection

- Modifiez l'architecture précédente
  - Quels éléments faut--il ajouter ou supprimer ?

3e étape : Automate avec accumulateur et indirection

l'architecture

- Un compteur programme

3e étape : Automate avec accumulateur et indirection

l'architecture

- Un compteur programme
- Une ALU

3e étape : Automate avec accumulateur et indirection

l'architecture

- Un compteur programme
- Une ALU
- Un registre pour l'instruction

3e étape : Automate avec accumulateur et indirection

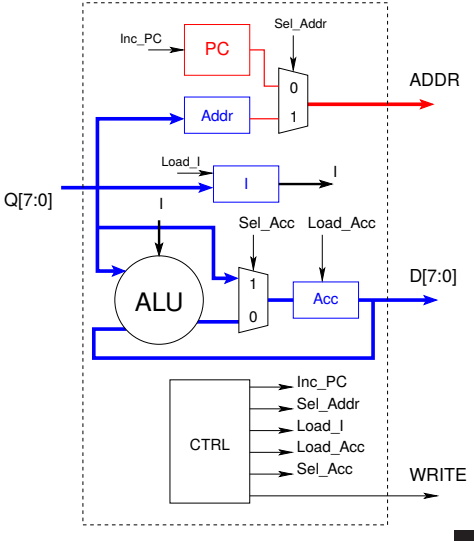
l'architecture

- Un compteur programme
- Une ALU
- Un registre pour l'instruction
- Un registre pour l'adresse des données
- La possibilité de choisir quelle adresse présenter

3e étape : Automate avec accumulateur et indirection

l'architecture

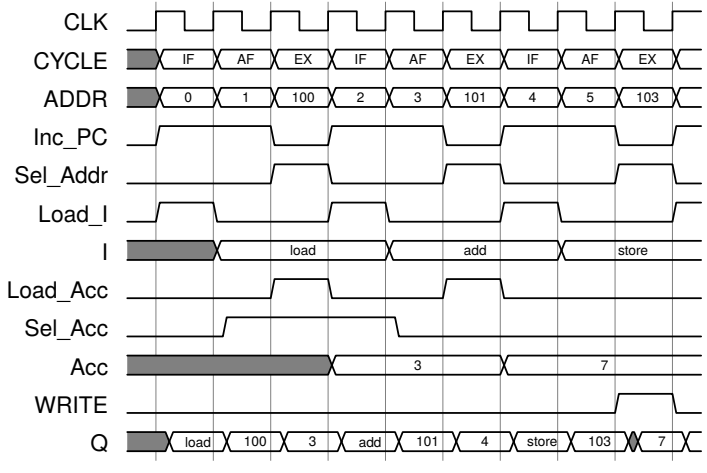
- Un compteur programme
- Une ALU
- Un registre pour l'instruction
- Un registre pour l'adresse des données
- La possibilité de choisir quelle adresse présenter
- Un registre pour l'accumulateur
- Un contrôleur



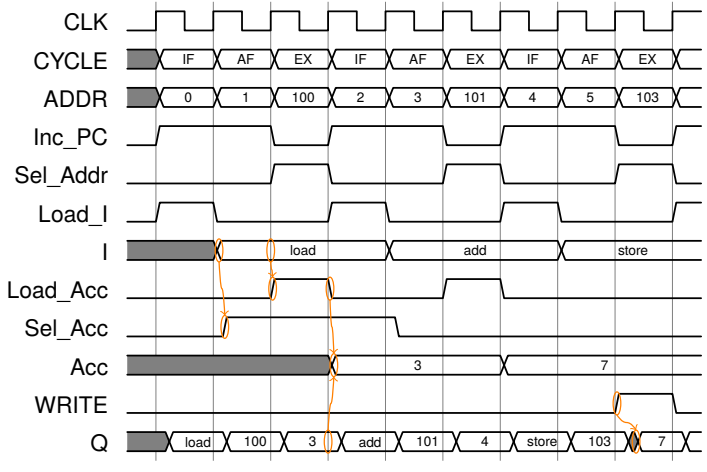
3e étape : Automate avec accumulateur et indirection

0	load
1	100
2	+
3	101
4	store
5	103
6	load
7	100
8	-
9	102
10	store
11	104
...	...
100	3
101	4
102	1
103	X
104	X
...	...

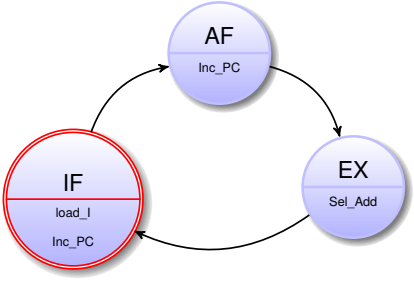
3e étape : Automate avec accumulateur et indirection



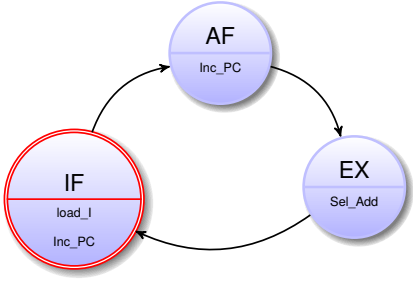
3e étape : Automate avec accumulateur et indirection



3e étape : Automate avec accumulateur et indirection



3e étape : Automate avec accumulateur et indirection

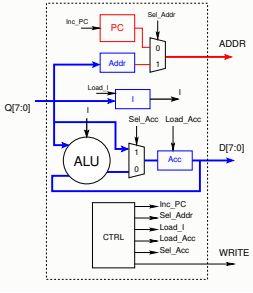


- L'adresse ne peut pas être utilisée, il faut un compteur modulo 3 en plus.

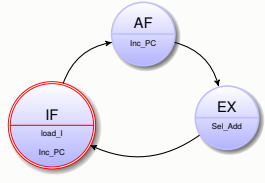
Inc_PC	=	Etat $\neq$ EX
Sel_Acc	=	(I $=$ load)
Load_Acc	=	(I $\neq$ store) && (Etat $=$ Ex)
WRITE	=	(I $=$ store) && (Etat $=$ Ex)
Sel_Add	=	Etat $=$ Ex

3e étape : Automate avec accumulateur et indirection

l'architecture



le contrôleur



Inc_PC	=	Etat $\neq$ EX
Sel_Acc	=	(I = load)
Load_Acc	=	(I $\neq$ store) && (Etat = Ex)
WRITE	=	(I = store) && (Etat = Ex)
Sel_Add	=	Etat = Ex

4e étape : Processeur complet

- Si on veut :
  - Exécuter les parties de code de façon conditionnelle
  - Organiser le code en fonctions réutilisables

4e étape : Processeur complet

- Si on veut :
  - Exécuter les parties de code de façon conditionnelle
  - Organiser le code en fonctions réutilisables
- On doit
  - Ajouter des instructions de saut
  - Ajouter des indicateurs pour les tests

4e étape : Processeur complet

code (binaire 8 bits)	instruction	effet
00000000	NOP	Rien
00000001	XOR	Acc = Acc XOR (AD)
00000010	AND	Acc = Acc AND (AD)
00000011	OR	Acc = Acc OR (AD)
00000100	ADD	Acc = Acc + (AD)
00000101	ADC	Acc = Acc + (AD) + C
00000110	SUB	Acc = Acc - (AD)
00000111	SBC	Acc = Acc - (AD) - C
00001000	ROL	Acc = {Acc[6:0], Acc[7]}
00001001	ROR	Acc = {Acc[0], Acc[7:1] }
00001010	LDA	Acc = (AD)
00001011	STA	(AD) = Acc
00001101	JMP	PC = AD
00001110	JNC	PC = AD si C=0
00001111	JNZ	PC = AD si Z=0

4e étape : Processeur complet

- Modifiez l'architecture précédente
  - Quels éléments faut--il ajouter ou supprimer ?

4e étape : Processeur complet

l'architecture

- Un compteur programme qu'on peut modifier

4e étape : Processeur complet

l'architecture

- Un compteur programme qu'on peut modifier
- Une ALU avec des indicateurs (flags)

4e étape : Processeur complet

l'architecture

- Un compteur programme qu'on peut modifier
- Une ALU avec des indicateurs (flags)
- Un registre pour l'instruction

4e étape : Processeur complet

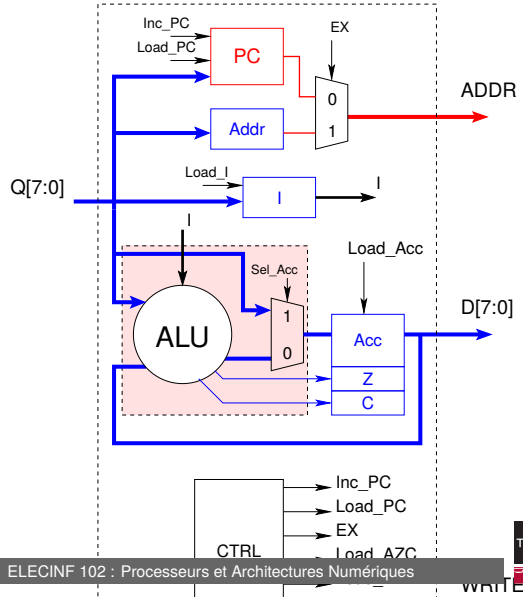
l'architecture

- Un compteur programme qu'on peut modifier
- Une ALU avec des indicateurs (flags)
- Un registre pour l'instruction
- Un registre pour l'adresse des données
- La possibilité de choisir quelle adresse présenter

4e étape : Processeur complet

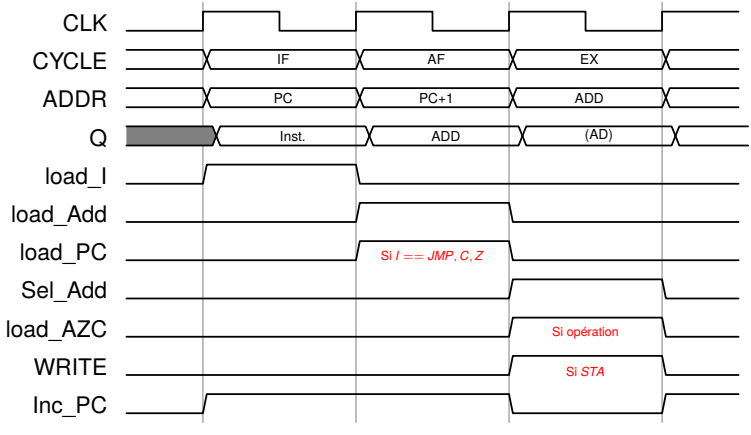
l'architecture

- Un compteur programme qu'on peut modifier
- Une ALU avec des indicateurs (flags)
- Un registre pour l'instruction
- Un registre pour l'adresse des données
- La possibilité de choisir quelle adresse présenter
- Un registre pour l'accumulateur et les flags
- Un contrôleur

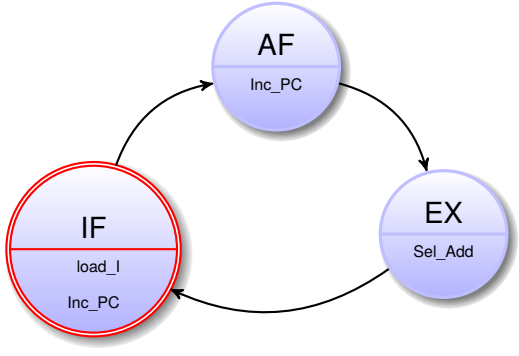


4e étape : Processeur complet

Chronogramme



4e étape : Processeur complet
le contrôleur

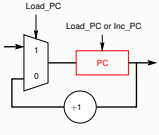


Sel_AZC	=	(I = LDA)
Load_AZC	=	(I ≠ (STA, JMP, NOP) && (Etat = Ex)
WRITE	=	(I = STA) && (Etat = Ex)
Load_PC	=	(I=JMP || (I=JNZ && Z=0) || (I=JNC && C=0)) && (Etat = AF)

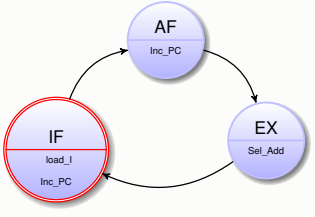
4e étape : Processeur complet

Récapitulons

l'architecture



le contrôleur

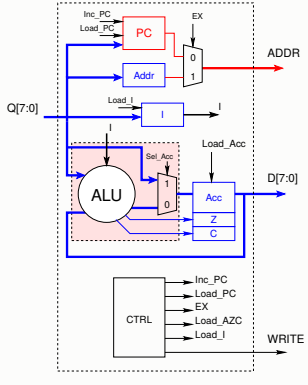


Sel_AZC = (I = LDA)
Load_AZC = (I ≠ (STA, JMP, NOP) && (Etat = Ex)
WRITE = (I = STA) && (Etat = Ex)
Load_PC = (I=JMP || (I=JNZ && Z=0) || (I=JNC && C=0)) && (Etat = AF)

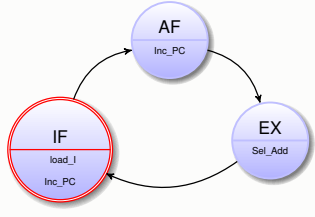
4e étape : Processeur complet

Récapitulons

l'architecture



le contrôleur



Sel_AZC	=	(I = LDA)
Load_AZC	=	(I ≠ (STA, JMP, NOP) && (Etat = Ex)
WRITE	=	(I = STA) && (Etat = Ex)
Load_PC	=	(I=JMP || (I=JNZ && Z=0) || (I=JNC && C=0)) && (Etat = AF)

4e étape+ : Processeur complet

Ajout d'un port en sortie

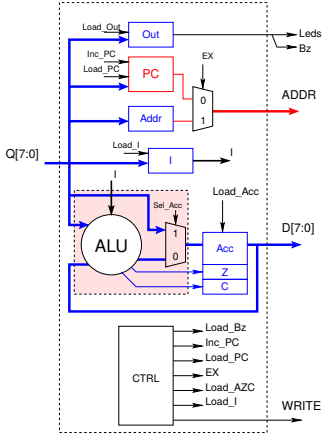
- Des signaux en sortie qu'on peut piloter à partir du processeur
  - Pour le buzzer nous avons besoin d'un seul bit en sortie
  - Les autres bits peuvent être utilisé a volonté (pilotage de leds)

Une instruction supplémentaire

code (binaire 8 bits)	instruction	effet
00001100	OUT	BZ = (AD)[0] Leds = (AD)[7:1]

4e étape+ : Processeur complet

Ajout d'un port en sortie



Load_Out = (I = OUT && (Etat = Ex))

Plan

Le langage assembleur

Spécifique à chaque microprocesseur...

- Fait pout être écrit et compris par un être humain.
- Traité par un programme pour générer le code binaire.

```
;================================================================================
;== Zone programme
;================================================================================
;label  code operande;   PC :  I  AD  -> accu   C  Z  Commentaire

:DEBUT  NOP  ZERO     ;  $00 : $0 $0B  -> $00    0  0  On ne fait rien
        LDA  COUNT    ;  $02 : $A $0A  -> $03    0  0  (A <- Memoire[$0A])
:COMPT1 SUB  UN       ;  $04 : $6 $0C  -> $02    0  0  (A <- $03 -  Memoire[$0C])
        JNC  COMPT1   ;  $06 : $E $04  -> $02    X  0  Retour à COMPT1 3 fois
        JMP  DEBUT    ;  $08 : $D $00  -> $FF    1  0  Retour à debut
;================================================================================
;== Zone donnees
;================================================================================
;Nom variable    Valeur     Ad   Décimal  Binaire     Hexadécimal
:COUNT           .db 3  ; $0A      3     00000011     $03
:ZERO            .db 0  ; $0B      0     00000000     $00
:UN              .db 1  ; $0C      1     00000001     $01
```

Le langage assembleur

Spécifique à chaque microprocesseur...

- Fait pout être écrit et compris par un être humain.
- Traité par un programme pour générer le code binaire.

```
;================================================================================
;== Contenu de la ram
;================================================================================
ADDR

0    : 00 0B
2    : 0A 0A
4    : 06 0C
6    : 0E 04
8    : 0D 00
.
.
.
.
A    : 3
B    : 0
C    : 1
```

Le langage assembleur

- Directives : étiquettes, expressions simples…

```
:debut lda adnot                ; recupere adresse de la premiere note
       sta tmpadnot             ; sauvegarde temporaire
; lecture des parametres cnote et duree
:oksui lda tmpadnot             ; recuperation du pointeur note
       sta dyn0 + 1             ;
:dyn0  lda zero                 ;
       sta dyn1 + 1             ; modifie l'instruction suivante !!!
:dyn1  lda zero                 ; en fait recupere le compteur de periode
       sta cnote                ; et sauvegarde dans cnote
; ...
:zero           .db 0           ; constantes
:un             .db 1
:deux           .db 2
:adnot          .db music
; ...
:tmpadnot       .db 0           ; variables temporaires
:cnote          .db 0
; ...
:music          .db  mi2        ; la musique (note , duree)
                .db  croche
; ...
```