Microprocesseur:
Architecture et concept !!!

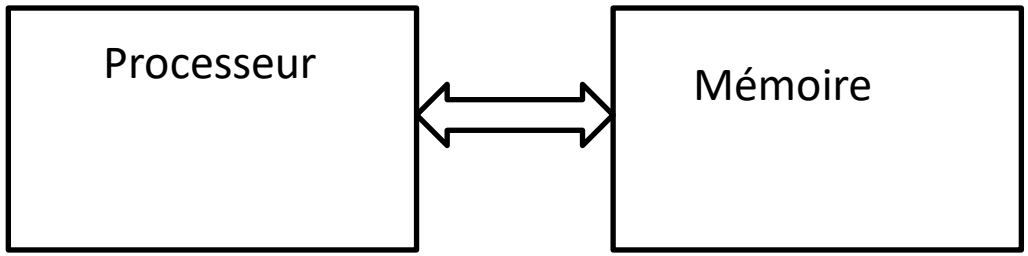
Le cœur des SE : Le processeur

- Les processeurs pour usage de bureau ne représentent qu'un petit pourcentage des processeurs vendus dans le monde.
- La plupart (98%) des processeur fabriqués sont destinés pour l'embarqué.
- Les application dans l'embarqué sont plus variées que pour les ordinateurs.

Architecture des processeurs

- Architecture Van Neuman

1. La mémoire contient les données et les instructions.
2. L'unité centrale (CPU) charges les instructions depuis la mémoire.



Architecture des processeurs

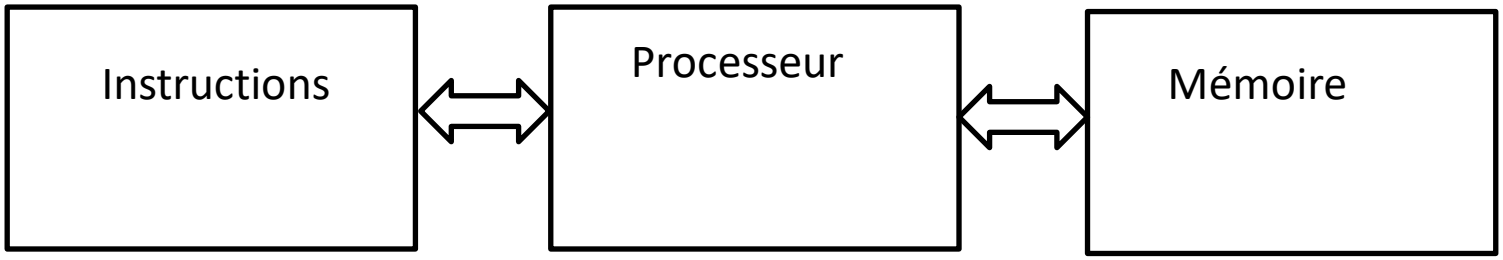
Un ensemble de registre aide le CPU.

1. Compteur d'instructions (Program Counter: PC)
2. Registre d'instruction (Instruction Register: IR)
3. Pointeur de Pile (Stack pointer: SP)
4. Registre à usage générale (Accumulateur: A)

Architecture des processeurs

- Architecture Harvard

1. Les données et les instructions dans des mémoires séparées.
2. Autorise deux accès simultanés à la mémoire.



Jeu d'instruction

- On a deux classes:

1. CISC : Complex instruction set computer
2. RISC: Reduced instruction set computer

Dans l'architecture CISC:

1. Une instruction peut designer plusieurs opérations élémentaires.
2. Grandes variation de taille et de temps d'exécution pour les instructions.
3. Un code compact mais complexe à générer.

Jeu d’instruction

Dans l’architecture RISC:

1. Instruction de même taille, ayant le même temps d’exécution.
2. Pas d’instructions complexes.
3. Accélération en utilisant des mécanismes de pipe-line.
4. Code plus simple à générer mais moins compact.

Pipe-line ?

- Le pipeline est un concept qui vient du monde de la production manufacturière.

- Son principe repose sur l'utilisation maximale de la ressource.
- Il permet d'exécuter une instruction tout en cherchant la suivante en mémoire.

Le processeur étape par étape

Qu’est ce qu’un processeur

- Un processeur est une machines de calcul programmables.

- Cette machine exécute une série d’opérations qu’on appelle programme.
- Un programme est une suite d’opération à effectuer.
- Dans le cas RISC (notre cas) les instructions sont simples.

Le programme

- Dans un programme, on distingue 2 types d'objets:

Les données:

- Les opérandes proprement dits (3, 5, ..)
- Les opérandes implicites (résultats d'un calcul, d'une étape, …)

Les instructions:

- Les opérations au sens arithmétiques
- Les tests
- Les sauts conditionnés
- …

La mémoire

- Le programme est stockés dans une mémoire de type RAM. Le processeur sera relié à cette RAM.

- L'exécution du programme est le suivant :

1. Aller lire la première ligne du programme (instruction et données associées).
2. Faire ce qui est indiqué
3. Aller lire la ligne suivante (instruction et données associées).
4. Faire ce qui est indiqué.
5. Revenir à l'étape 3.
6. …

Remarques

- Remarque 1 : le processeur doit lire les lignes une par une. Il doit donc posséder un compteur de ligne qui indique la ligne courante.

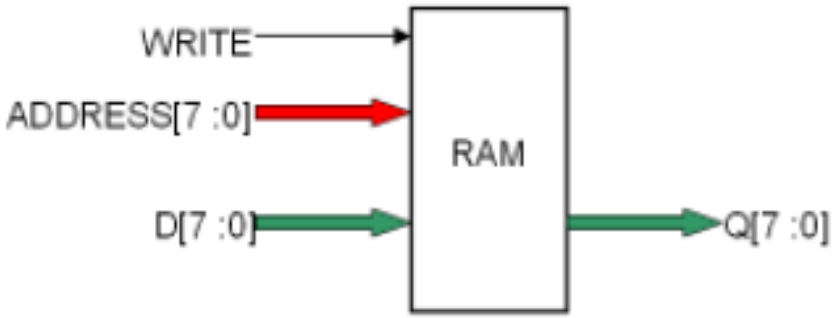
- Remarque 2 : les instructions sont codées sur un nombre de bits suffisant. Les données naturelles seront codées de façon normale (ou en complément à 2).

- Remarque 3 : Nous allons expliquer le cas ou la RAM contient les données et les instructions.

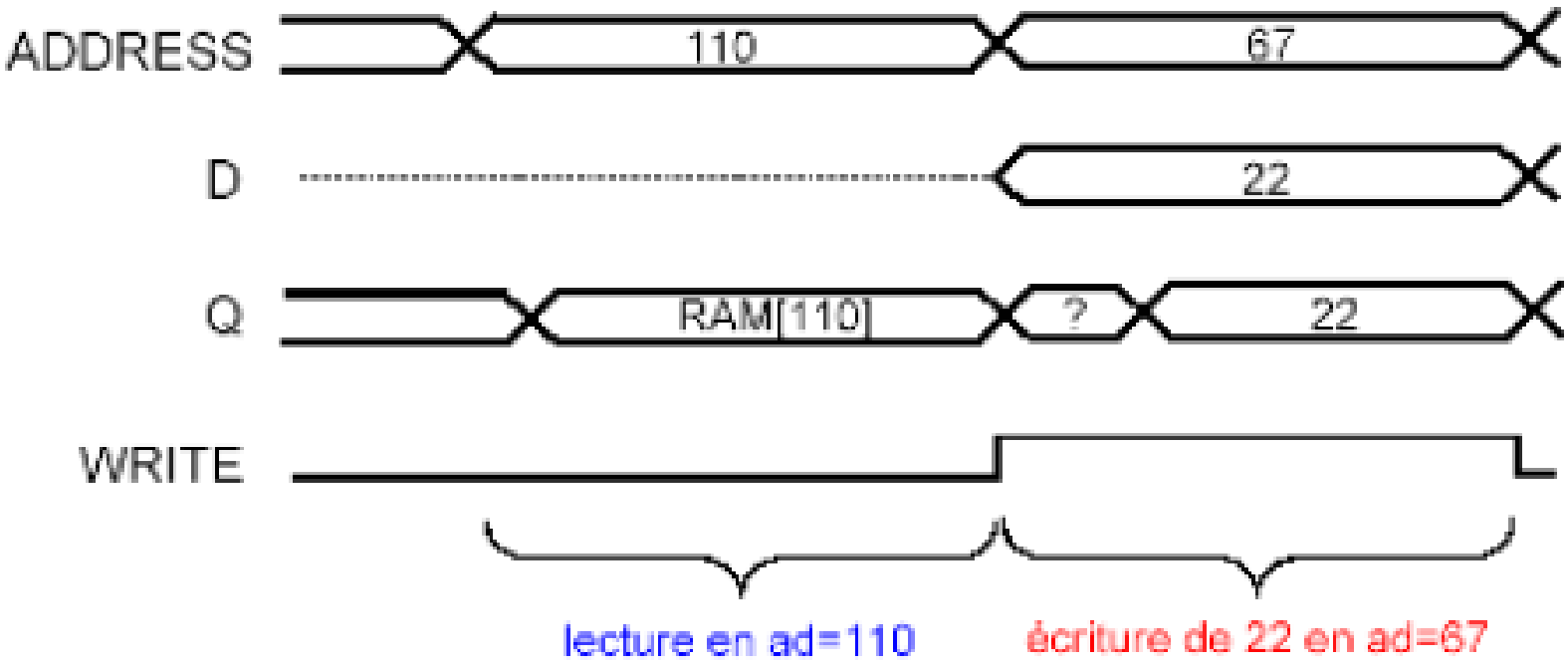
Fonctionnement de la RAM

La RAM possède 3 bus:

- Un bus d'adresse: indiquant l'emplacement mémoire de la donnée à laquelle on accède.
- Un bus de données: D[7:0] pour les données qu'on écrit sur la RAM.
- Un bus de données: Q[7:0] pour les données qu'on lit en RAM.
- Un signal de contrôle sur 1 bit, Write, indiquant si on est entrain de faire une lecture (Write = 0), ou une écriture (Write = 1).

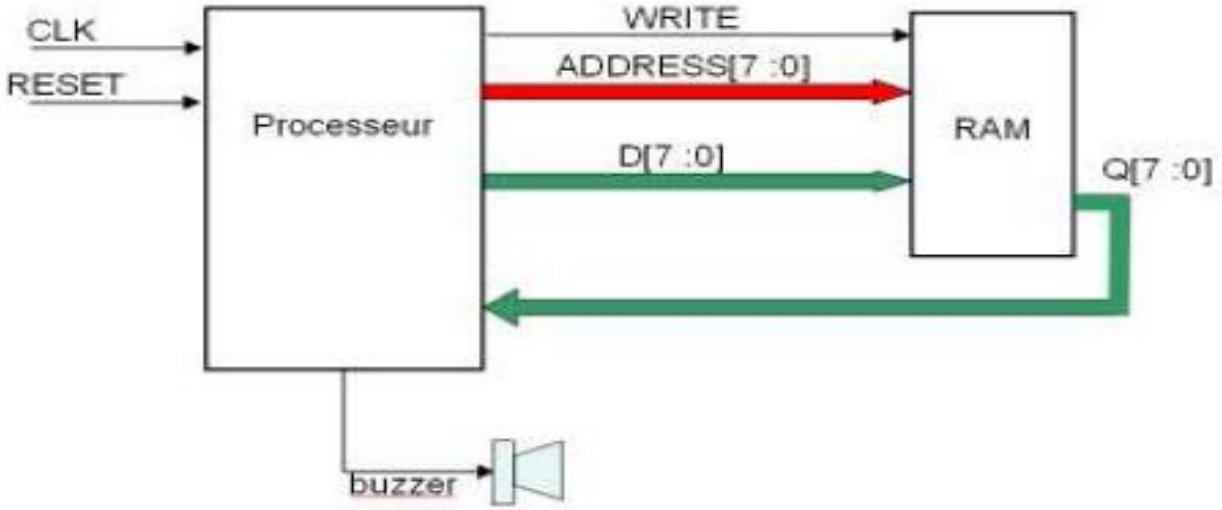


Fonctionnement de la RAM



Processeur couplé à une RAM

- Nous relions le processeur à une RAM pouvant stocker 256 mots de 8 bits. Il faut donc 8 lignes d'adresse.
- L'architecture est donc la suivante:



Processeur : 1ère version

- On suppose que:

- Le programme est chargé dans la mémoire.
- Les instructions sont arithmétiques ou logiques.
- La mémoire est organisée de la manière suivante:

adresse	Type du mot stockés	Exemple
0	instruction	+
1	Donnée (1er opérande)	3
2	Donnée (2ième opérande)	4
3	Donnée (résultat)	*
4	instruction	-
4	Donnée (1er opérande)	12
5	Donnée (2ième opérande)	8
6	Donnée (résultat)	*
….		

Fonctionnement de la machine

Remarques:

- Le processeur doit commencer son exécution à l'adresse 0 de la mémoire.

- On part du principe qu'on a toujours une instruction à l'adresse 0 de la RAM.

- On a toujours en mémoire une instruction puis l'opérande 1 suivit de l'opérande 2 puis un octet pour stocker le résultat.

Fonctionnement de la machine

- 1er coup d'horloge: le processeur présente d'adresse 0 à la RAM. Le processeur récupère l'instruction à faire dans le bus Q[7:0].
- 2ième coup d'horloge: le processeur incrémente l'adresse à 1. Le processeur récupère le contenu de l'opérande 1.
- 3ième coup d'horloge: le processeur incrémente l'adresse à 2. Le processeur récupère le contenu de l'opérande 2. Le processeur réalise l'opération à faire.
- 4ième coup d'horloge: le processeur incrémente l'adresse à 3 et met Write =1. Il présente sur le bus D le résultat. Le résultat est donc écrit à l'adresse 3 de la RAM.
- et le cycle se répète ….

Fonctionnement de l'automate

Remarques:

- Une séquence de 4 états modélise le fonctionnement de cette architecture.

  état 1--> état 2 --> état 3 -->état 4 -->état 1 …

- Les informations ramenées depuis la RAM doivent être stockées à l'intérieur du processeur pour pouvoir exécuter l'instruction à l'état 3 du cycle.

Fonctionnement de l'automate

- Etat 1:

- Le processeur récupère l'instruction de la RAM.

- Il faut stocker cette instruction pour plus tard. Un registre 8 bits disposant d'un Enable est ajouté. L'entrée de ce registre est reliée au bus Q[7:0].

- Le signal Enable est mis à 1 uniquement pendant l'état 1 (stockage de l'instruction).

Fonctionnement de l'automate

- Etat 2:

- Le processeur récupère l'opérande 1 de la RAM.

- Il faut stocker cette opérande pour plus tard. Un registre 8 bits disposant d'un Enable est ajouté. L'entrée de ce registre est reliée au bus Q[7:0].

- Le signal Enable de ce registre est mis à 1 uniquement pendant l'état 2 (stockage de l'opérande 1).

Fonctionnement de l'automate

- Etat 3:

- Le processeur récupère l'opérande 2 de la RAM.

- Il faut stocker cette opérande pour plus tard. Un registre 8 bits disposant d'un Enable est ajouté. L'entrée de ce registre est reliée au bus Q[7:0].

- Le signal Enable de ce registre est mis à 1 uniquement pendant l'état 3 (stockage de l'opérande 2).

Fonctionnement de l'automate

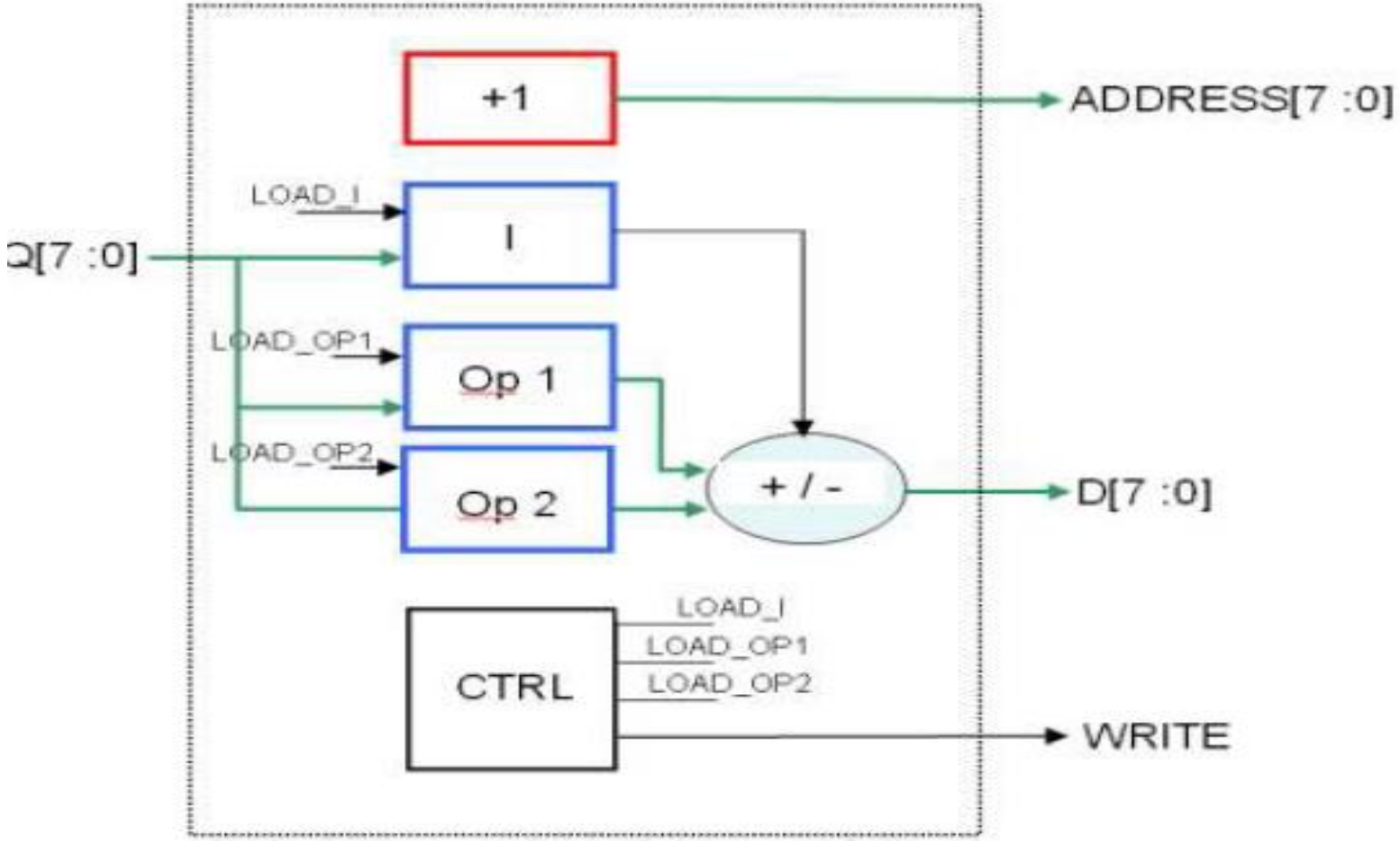
- Etat 4:

- L'automate (avec les 3 registres) dispose de toutes les données pour effectuer l'opération.

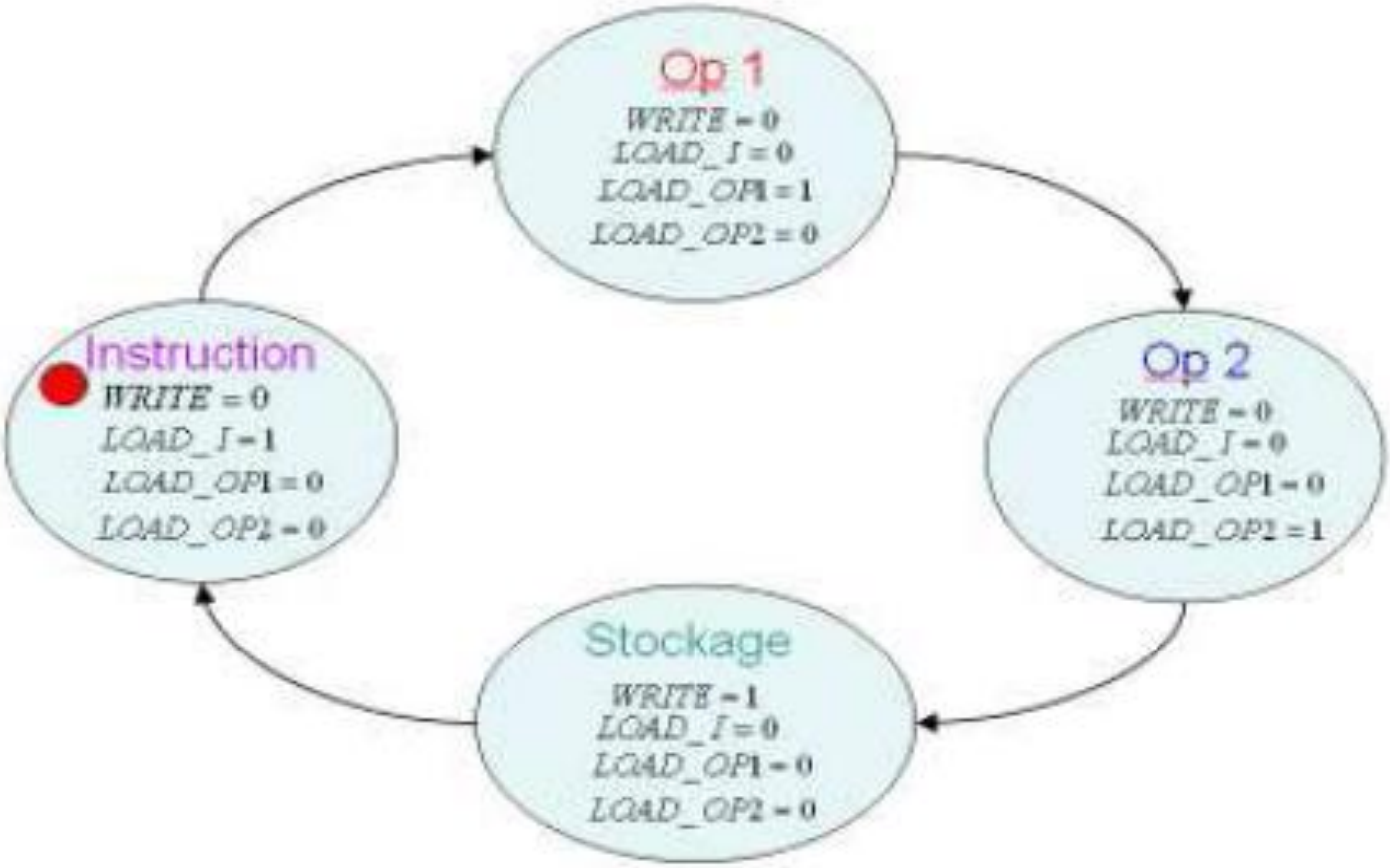
- Il suffit d'ajouter une ALU (circuit combinatoire) pour réaliser le calcul.

- L'automate met le signal Write à l'état haut pour stocker le résultat en RAM à l'adresse courante.

Architecture de l’automate



Machine à état de la 1ère version



Notion d’accumulateur

- L’architecture actuelle ne permet pas de chainer les calcul (ex: 3+4+5-7).

- Pour palier à ce problème, un registre que nous appelons accumulateur est ajouté.

- Toutes les opérations arithmétiques (ou logiques) à deux opérandes s’effectuent entre l’accumulateur et une donnée en RAM.

Automate avec accumulateur

Pour effectuer 3+4 et stocker le résultat en RAM, le processeur effectue les opérations suivantes:

- Charger 3 dans l'accumulateur.
- Addition du contenu de l'accumulateur avec une opérande en RAM.
- Stockage du contenu de l'accumulateur en RAM.

Processeur 2ième version

On doit donc disposer de deux instructions :

- Load: chargement de l'accumulateur à partir de la RAM.
- Store: stockage du contenu de l'accumulateur dans la RAM.

Les opérations arithmétiques et logiques n'ont besoin que d'un seul opérande (le 2ième opérande est dans l'accumulateur).

Organisation de la mémoire

Remarque :

- Une adresse sur 2 contient une instruction, 1 sur 2, une donnée (soit opérande, soit stockage du contenu de l'accumulateur).
- La mémoire est parcourue d'une manière linéaire (pas de saut).

Adr.	Type	Exemple	Effet
0	Instruction	Load	
1	Donnée	3	Accu =3
2	Instruction	+	
3	Donnée	4	Accu= 7
4	Instruction	-	
5	Donnée	1	Accu=6
6	Instruction	Store	
7	Donnée	*	Cet emplacement contient 6
…			

- Pour une instruction arithmétique ou logique

1. Chercher l'instruction en RAM et la stocke dans le registre I.
2. Chercher l'opérande en RAM, effectuer le calcul et stocke le résultat dans l'accumulateur.

Pour un Load:

1. Chercher l'instruction en RAM, la stocke dans I.
2. Chercher l'opérande en RAM, le stocke dans l'accumulateur.

Pour un Store:

1. Chercher l'instruction en RAM, la stocke dans I.
2. Ecrire le contenu de l'accumulateur à l'adresse courante.

Lors du deuxième cycle, l'accumulateur est modifié selon le type de l'opération.

- Pour une opération arithmétique ou logique l'accumulateur est modifié par le résultat de l'opération.
- Pour un load, l'accumulateur est chargé par la donnée sortant de la RAM.
- Pour un store, l'accumulateur n'est pas modifié.

En entrée de l'accumulateur on doit mettre un multiplexeur qui présentera à l'accumulateur soit le résultat de l'opération en cours soit le contenu de la RAM.

Dans les deux cas, le signal enable de l'accu est mis à l'état haut pour autoriser sa modification. Dans le cas d'un store, l'enable de l'accu est à l'état bas pour ne pas le modifier.

La machine à état correspondante est la suivante :

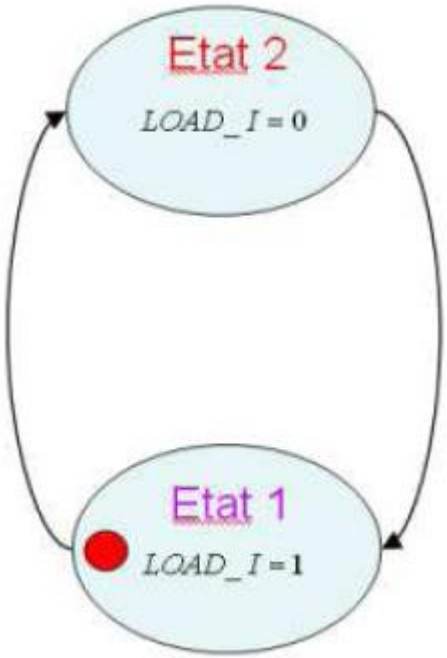
SEL_ACC = (I[7:0] == LOAD)

LOAD_ACC = (I[7:0] <> STORE) ET (Etat = état 2)

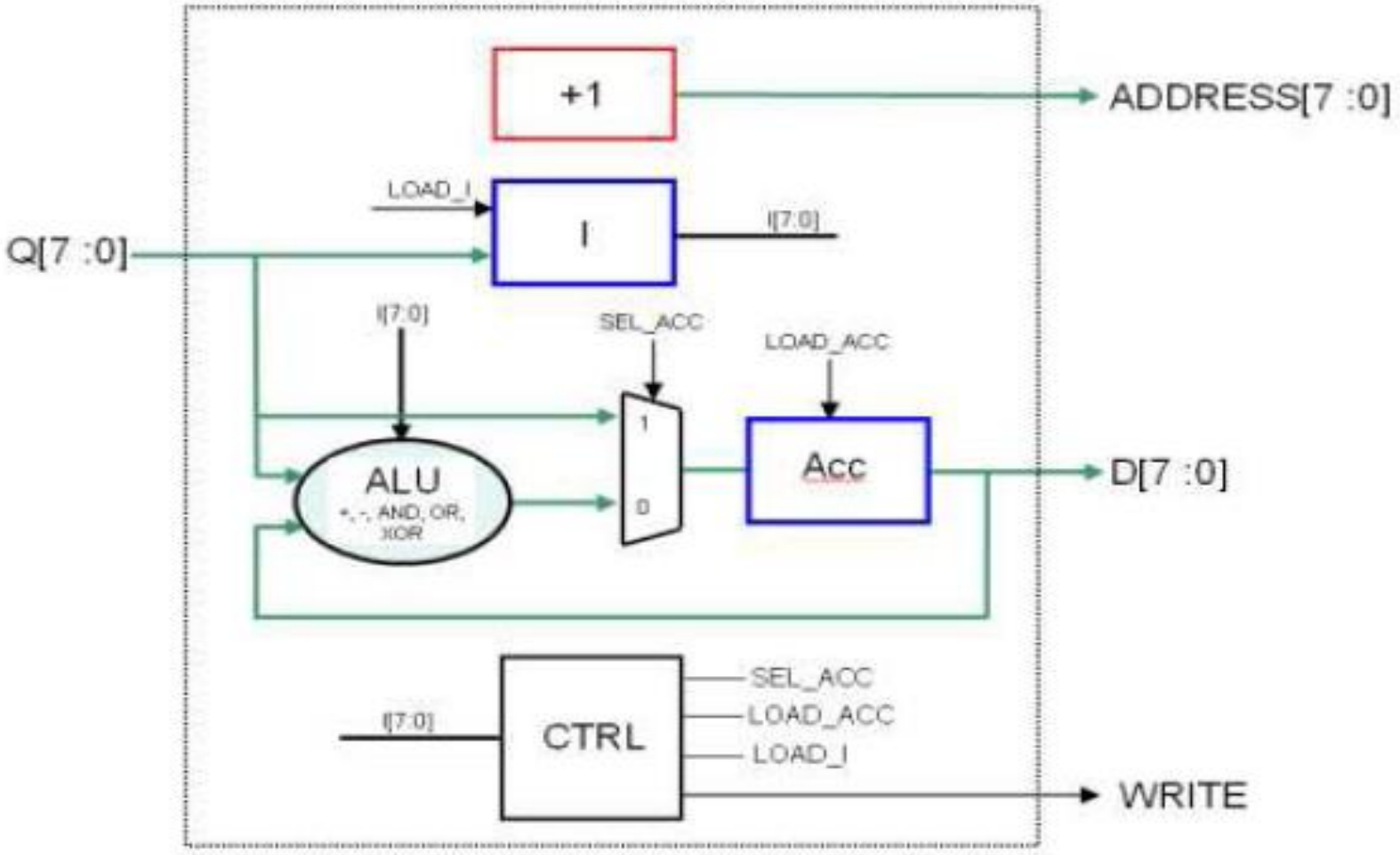
WRITE = (I[7:0] == STORE) ET (Etat = état 2)

Avec:

<> : différent.



Architecture de la 2ième version



2ième version modifiée

On peut séparer le code des données, pour :

- Faire tourner le même code sur des données différentes.
- Faire tourner différents code sur des données identiques.
- Faire tourner un code sur des données qui ne sont pas connues avant l'exécution du programme.

On séparant le code des données, on aurait en RAM

- Une zone avec les instruction
- Une zone avec le code

Organisation de la RAM

Remarque:

D'habitude, on sépare même la zone de données en deux :

- Celle qui sont connues à l'écriture du programme.
- Celles qui sont modifiées par le programme.

Adr.	Type	Exemple	zone
0	instruction	Load	1
1	Adresse opérande	100	1
2	instruction	+	1
3	Adresse opérande	101	1
4	instruction	store	1
5	Adresse opérande	103	1
…			1
100	Donnée	3	2
101	Donnée	4	2
102	Donnée	1	2
103	Donnée	*	2

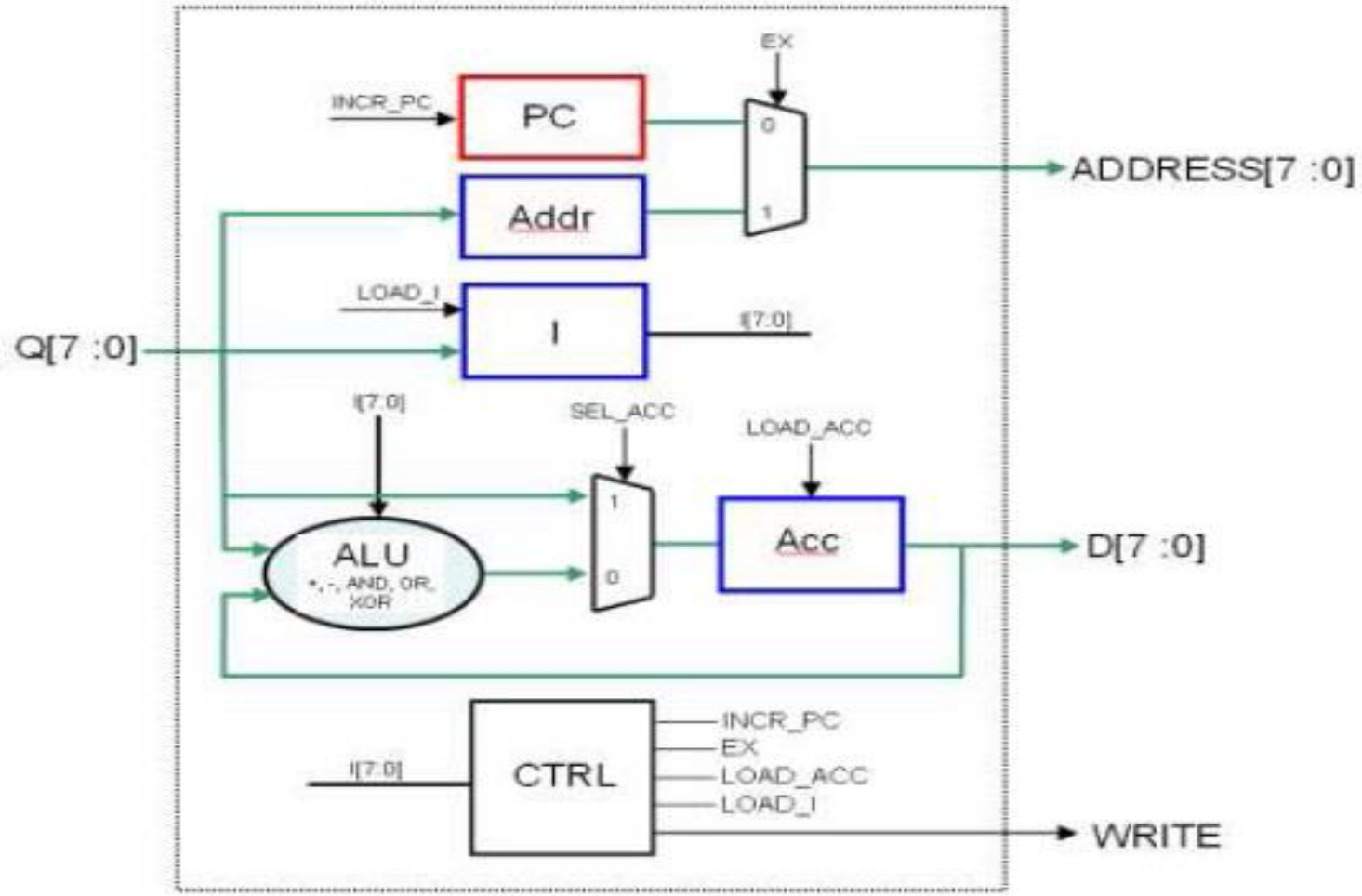
Architecture de la version modifiée

L'automate doit maintenant pour chaque instruction:

- Chercher l'instruction et la stocke dans le registre d'instruction.
- Chercher l'adresse de l'opérande et le stocke dans un registre dit d'adresse.
- Chercher l'opérande proprement dit.

On a donc une machine qui possède un état de plus.

Architecture de la 3ième version

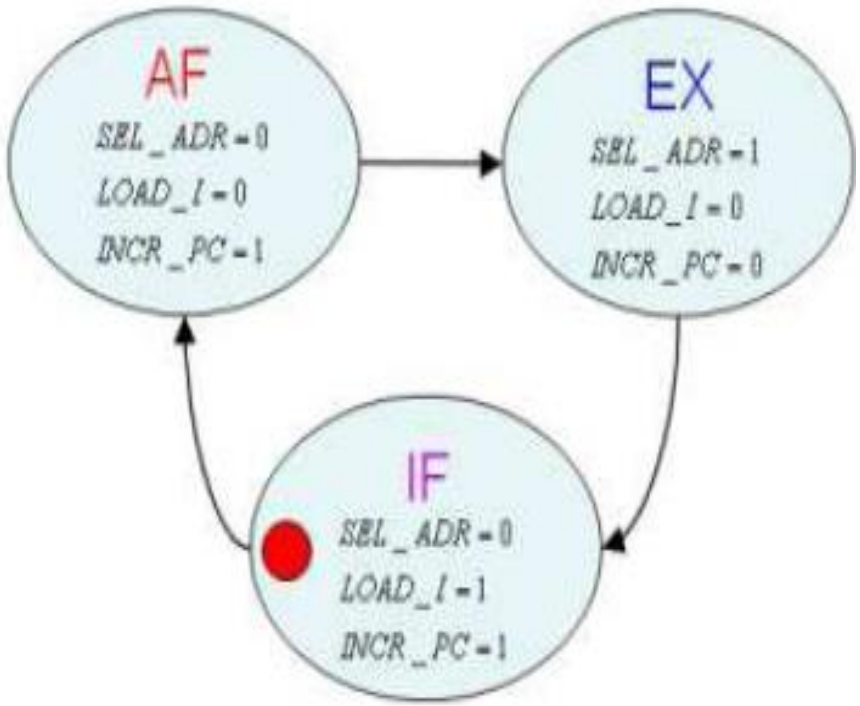


Machine à état de la 3ième version

SEL_ACC = (I[7:0] == LOAD)

LOAD_ACC = (I[7:0] <> STORE) ET (Etat = Ex)

WRITE = (I[7:0] == STORE) ET (Etat = Ex)



Avec:

<> : différent.

Remarque : L’architecture actuelle ne sait faire que des calculs linéaires (suite fixe d’instruction) sur des données potentiellement inconnues (mais dont l’adresse de stockage est connue).

Processeur 4^{ième} version

- Pour compléter notre processeur, on doit ajouter des instructions de saut (conditionnés ou non).
- Notion de Falgs (drapeaux):
- Chaque opération (logique ou arithmétique) va positionner deux signaux devant être mémorisés pour l'instruction suivante.
- Ces signaux (bits) ne doivent être modifiés que si on modifie l'accumulateur.

Les FLAGS !!

- C (carry):

- Mis à 1 si l'opération courante est arith. et donne lieu à une retenue.
- Mis à 0 si l'opération courante est arith. et ne donne lieu à une retenue.
- Mis a 0 si on fait un load.

- Z (zero):

- Mis à 1 si on charge 0 dans l'accumulateur.
- Mis à 0 dans tous les autre cas.

On peut ajouter maintenant deux opérations, ADDC et SUBC, qui prennent en compte la retenue C de l'opération précédente.

Pour implémenter les sauts on définit trois instructions supplémentaires.

JMP: saut inconditionnel

L'exécution de cette instruction fait sauter l'exécution du programme directement à une adresse donnée.

JNC: saut si C est nul

L'exécution de JNC fait sauter l'exécution du programme à une adresse donnée si C est nul. Sinon, équivalent à NOP (on continue à l'adresse suivante).

JNZ: saut si Z est nul

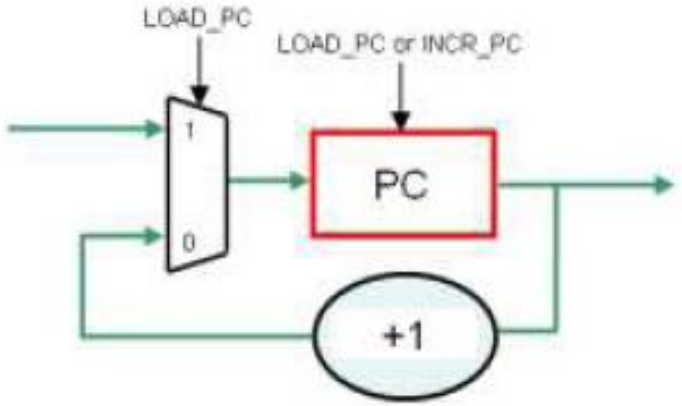
L'exécution de JNZ fait sauter l'exécution du programme à une adresse donnée si Z est nul. Sinon, équivalent à NOP

instruction	effet	explication
NOP		ne fait rien !
XOR	Acc = Acc XOR (AD)	effectue un XOR bit à bit entre le contenu de l'accumulateur et une donnée en RAM, le résultat est stocké dans l'accumulateur
AND	Acc = Acc AND (AD)	effectue un ET bit à bit entre le contenu de l'accumulateur et une donnée en RAM, le résultat est stocké dans l'accumulateur
OR	Acc = Acc OR (AD)	effectue un OU bit à bit entre le contenu de l'accumulateur et une donnée en RAM, le résultat est stocké dans l'accumulateur
ADD	Acc = Acc + (AD)	additionne le contenu de l'accumulateur à une donnée en RAM, le résultat est stocké dans l'accumulateur
ADC	Acc = Acc + (AD) + C	additionne le contenu de l'accumulateur à une donnée en RAM et à la carry C, le résultat est stocké dans l'accumulateur
SUB	Acc = Acc - (AD)	soustrait du contenu de l'accumulateur une donnée en RAM, le résultat est stocké dans l'accumulateur
SBC	Acc = Acc - (AD) - C	soustrait du contenu de l'accumulateur une donnée en RAM et la carry C, le résultat est stocké dans l'accumulateur
ROL	Acc = {Acc[6 :0], Acc[7]}	effectue une rotation vers la gauche des bits de l'accumulateur
ROR	Acc = {Acc[0], Acc[7 :1] }	effectue une rotation vers la droite des bits de l'accumulateur
LDA	Acc = (AD)	charge dans l'accumulateur une donnée en RAM
STA	(AD) = Acc	stocke le contenu de l'accumulateur en RAM
OUT	BZ = (AD)[0]	Sort sur la broche BZ le bit de poids faible de la donnée en RAM, stockée à l'adresse opérande
JMP	PC = AD	saute à l'adresse opérande
JNC	PC = AD si C=0	saute à l'adresse opérande si C est nul, ne fait rien sinon
JNZ	PC = AD si Z=0	saute à l'adresse opérande si Z est nul, ne Afait rien sinon

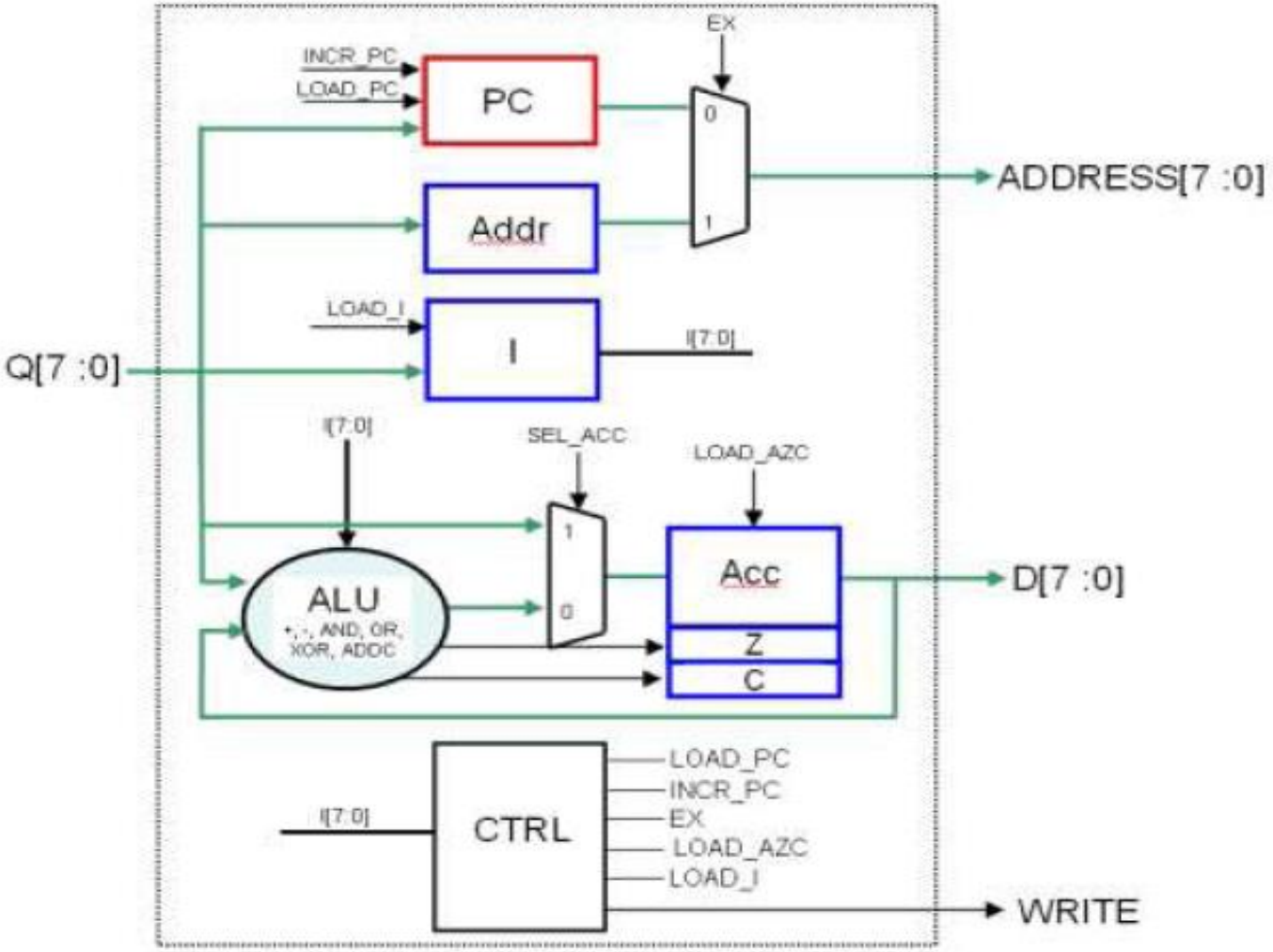
PC !!

Pour implémenter les sauts, il suffit de se donner la possibilité de remplacer le contenu de PC par la valeur lue en RAM.

Le compteur de programme devient un peu plus complexe.



Architecture de la 4ième version



Machine à état de la 4ième version

SEL_ACC = (I[7:0] == LOAD)

LOAD_ACC = (I[7:0] <> (STORE ou saut)) ET (Etat = Ex)

WRITE = (I[7:0] == STORE) ET (Etat = Ex)

LOAD_PC = si (I[7:0] == JMP ou I[7:0] == JNC et C == 0 ou I[7:0] == JNZ et Z == 0) et (état = Ad), alors 1, sinon 0

Avec:

<> : différent.