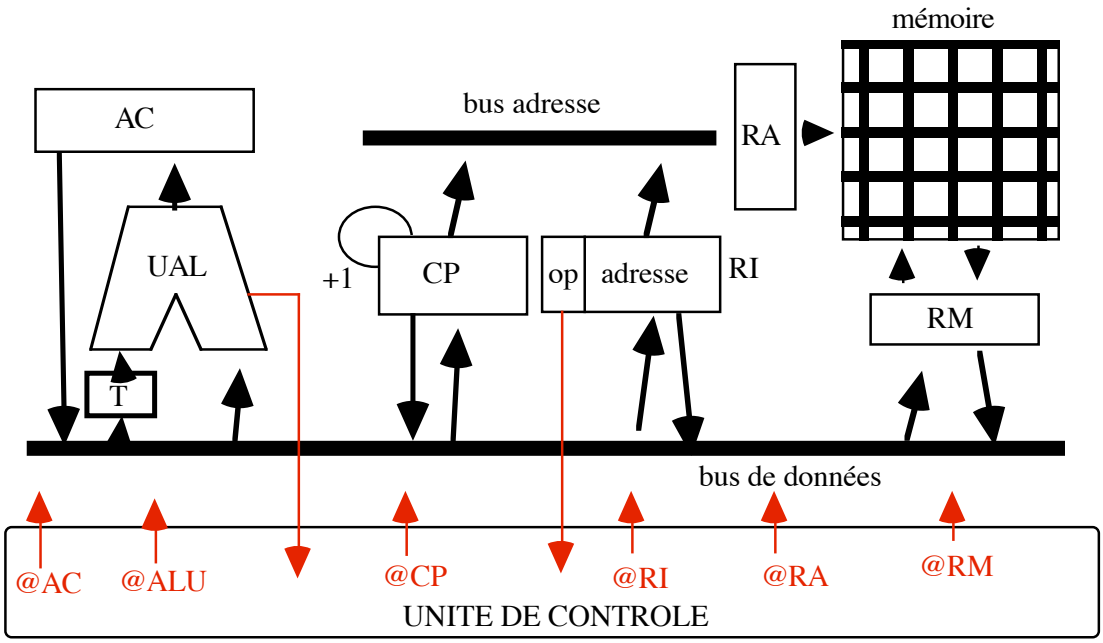
Corrigé indicatif
VARI - ED Archi.

Exercice 1 - Un processeur minimal



Rappeler le rôle des différents organes dans le schéma ci-dessus

Le principe de la machine de Von Neumann

Exécution séquentielle (automatique) d'un programme enregistré en mémoire centrale.

Les composants

L'Unité de Contrôle ou de commande

- gère l'enchaînement séquentiel des instructions
- pilote les échanges avec la mémoire
- décode l'opération et commande (suite de signaux) l'Unité Arithmétique et Logique

L'Unité Arithmétique et Logique

Exécute l'instruction

La Mémoire Centrale

Stocke le programme et les données

Fonctionnement général

La communication entre le processeur (UC + UAL) et la mémoire se fait via les bus d'adresses et de données. Le Registre Adresse **RA** contient une adresse mémoire envoyée sur le bus d'adresses par l'Unité de contrôle et le registre mot **RM** contient le contenu d'un mot mémoire destiné au processeur via le bus de données. Ces registres permettent donc les opérations de lecture et d'écriture dans la mémoire.

Le registre Compteur de Programme ou Compteur Ordinal **CP** contient l'adresse mémoire de la prochaine instruction à exécuter.

L'instruction à exécuter est contenue dans le registre instruction **RI** .

COP est le code opération de l'instruction, il est traité par le **décodeur** qui détermine l'opération à exécuter (microprogramme par exemple) ; le **séquenceur** génère les signaux de commande pour l'unité arithmétique et logique ; les signaux d'une **horloge** permettent au séquenceur d'enchaîner les commandes..La partie opérande est une adresse mémoire ou une valeur.

L'**UAL** contient tous les circuits permettant de réaliser les opérations élémentaires arithmétiques et logiques. Le registre accumulateur **AC** contient le résultat d'une opération. Le registre temporaire **T** sert à mémoriser une valeur lorsque l'opération nécessite une autre valeur, par exemple une addition.

Que manque-t-il surtout dans cette architecture minimale ?

- des registres :

  des registres banalisés : registre pouvant servir à diverses opérations telles que le stockage des résultats intermédiaires. Ils ont le même rôle que l'accumulateur

  des registres spécialisés : Stack pointer SP, pointeur de pile pour les procédures, registres pour les différents types d'adressage segmentés/basé/indexé.

  le registre MEP mot d'état programme (PSW) qui indique l'état du système, notamment caractérise le résultat des opérations : dépassement de capacité(overflow), retenue(carry), résultat posistif, résultat nul.

- Éventuellement plusieurs UAL (flottant...) et des caches pour faire du pipeline par exemple.

- La communication avec les unités d'Entrées /Sorties.

Quelle est la taille de l'espace mémoire adressable ?

2^64 = 1.8 * 10^19 : c'est énorme - et un peu inutile (sauf pour faire une mémoire virtuelle)

Quels sont les nombres manipulables par l'UAL ?

Pour des entiers signés,on peut coder 2^64 –1 nombres (c'est le cas des long integer de l'ADA 95)

C'est aussi la taille utilisée pour les flottants dans la norme IEEE 754 en double précision.

Comment peuvent être codées les instructions de ce processeur ?Quel est l'inconvénient de cette approche et comment le pallier ?

Tout dépend des instructions... elles peuvent avoir 0, 1, 2 voire 3 paramètres (opérandes). Ici avec un seul registre accumulateur, on peut imaginer 1 seul opérande au

pire. Celui-ci peut être une valeur ou une adresse. Il est impossible d'avoir dans le même mot de 64 bits, un champ de code opération et un nombre. Deux solutions :

- mettre le code opérateur et la valeur sur 2 mots consécutifs : on gaspille

- réduire la taille des nombres manipulables par les instructions. Pour les adresses ce n'est pas un problème (en pratique, une vingtaine de bits suffisent de nos jours), mais pour les entiers c'est dommage. Si beaucoup d'opérations sans paramères sont utilisées, on gaspille.

On peut résoudre ce problème en réduisant la taille du mot mémoire, et/ou en permettant que les bits d'un nombre soient à cheval sur 2 mots consécutifs.

Exercice 2 - Chronogramme d'exécution

L'Unité de Contrôle commande deux phases ou cycles **:** le cycle de recherche (fetch) d'une instruction en mémoire centrale (MC) et le cycle d'exécution de l 'instruction.

Étapes du cycle de recherche :

1) transfert de l'adresse de l'instruction du CP (Compteur de Programme) vers le RA (Registre Adresse de la mémoire)
2) une impulsion de lecture provoque le transfert de l'instruction vers le RM (Registre Mot)
3) transfert de l'instruction dans le RI(Registre Instruction)
4) parallèlement envoi de l'adresse de l'opérande vers le RA et transmission au décodeur du code opération, détermination du type d'opération et envoi au séquenceur du signal correspondant à l'opération
5) le CP est incrémenté pour le cycle de recherche suivant.

Étapes du cycle d'exécution :

1) le séquenceur envoie les signaux de commande vers la mémoire pour lire l'opérande dont l'adresse est stockée dans RA et le stocker dans RM.
2) transfert du contenu de RM vers l'accumulateur de l'UAL, s'il s'agit d'une opération de lecture de l'opérande, si c'est une mémorisation d'un résultat, c'et un transfert de AC vers RM.
3) L'opération est effectuée par l'UAL sous contrôle du séquenceur

Décrire le séquencement de l'instruction ADD 155

Rappelons que le schéma est très simplifiant. Il manque toute la circuiterie (portes logiques, adapatation électrique) pour faire en sorte de gouverner les flux entre les organes. Cette circuiterie est commandée par les signaux produits par l'unité de contrôle.

1) Phase fetch : recherche de l'instruction courante

CP -> bus adresse // bus adresse -> RA

lecture mémoire

RM -> bus données // bus données -> RI

2) Phase de décodage. L'opcode contenu dans RI est lu par l'unité de contrôle qui démarre le micro-programme correspondant : ici celui pour l'instruction ADD qui est une instruction avec 1 opérande

RI -> bus adresse // bus adresse-> RA

lecture mémoire

3) Exécution. A ce stade le contenu de la case 155 est dans RM

AC -> bus données // bus données -> T

RM -> bus données // bus données -> entrée UAL

addition

sortie UAL -> AC

Décrire le séquencement de l'instruction ADDi 155

Seule la phase de décodage change : il faut aller chercher ce que contient la case dont l'adresse est le contenu de la case 155.

RI -> bus adresse // bus adresse-> RA

lecture mémoire

RM -> bus données // bus données-> RI

RI -> bus adresse // bus adresse-> RA

lecture mémoire

Combien de cycles d'horloge sont maintenant nécessaires ?

Trois de plus

Quelle valeur doit-on choisir pour le temps de cycle élémentaire?

C'est la micro-opération la plus lente qui doit définir le temps de cycle. On a :

- des transferts registre / registre

- des opérations UAL

- des lecture/écriture en mémoire

C'est cette dernière opération qui est la plus lente: 100 ns pour de la vieille DRAM. Ce qui implique une vitesse d'horloge pour notre processeur de 10 MHz

Exercice 3 - Exemple de programme

Ecrire ce programme en supposant que ses données ne sont pas dupliquées.

En pseudo Ada :

```
for k := addeb to adfin loop
    Mem(k+1000) := Mem(k)
end loop
addeb := addeb + 1000
adfin := adfin + 1000
if adfin < Mem'size then goto addeb
```

On va d'abord transcrire en une sorte d'assembleur, c'est à dire sans se préoccuper de l'emplacement mémoire des données et des adresses de saut pour les branchements :

```
LOAD addeb
STORE k
au_debut : LOAD adfin
INF k
BRA plus_bas  // si k>adfin on arrete
SET +1000
ADD k
STORE j          // j = k+1000
LOADi k
STOREi j        // M(j) := M(k)
SET +1
ADD k
STORE k      // k := k+1
RAZ
BRA au_debut  // end loop
plus_bas : SET +1000
ADD addeb
STORE addeb // addeb := addeb + 1000
SET +1000
ADD adfin
STORE adfin // adfin := adfin + 1000
INF -1 //astuce pour avoir le plus grand entier positif admissible
BRA au_debut_prochaine_copie // pour propager l'execution vers la copie (pourquoi pas ?)
HALT
```

A ce stade, on connaît l'occupation mémoire du programme. Comme 1 instruction = 1 case, il y a ici 24 cases pour le code. On peut placer à la suite les 4 variables addeb, adfin, k , j. On peut maintenant construire l'exécutable :

```
#case          valeur (en décimal)     valeur (en binaire)
00   LOAD 24
01   STORE 26
02   LOAD 25
03   INF 26
04   BRA +11
05   SET +1000
06   ADD 26
07   STORE 27
08   LOADi 26
```

```
09  STOREi 27
10  SET +1
11  ADD 26
12  STORE 26
13  RAZ
14  BRA -12
15  SET +1000
16  ADD 24
17  STORE 24
18  SET +1000
19  ADD 25
20  STORE 25
21  INF -1
22  BRA +978
23  HALT
24  0
25  23
26  XXX
27  XXX
```

Que manque-t'il au jeu d'instruction pour que les données soient elles aussi dupliquées ?

Dans cette version, les 4 variables occupent l'emplacement initial. On pourrait vouloir que les variables de la n-ième copie se trouvent à la suite du code de cette copie. Pour cela, il faut un mecanisme d'adressage relatif des données, comme c'est déjà le cas pour le flot d'instruction (l'instruction BRA utilisée est relative).

Une première solution est d'ajouter au jeu d'instruction :

"load relatif" LOADr valeur : (AC) := (CP + valeur)

"store relatif" STORErvaleur : (CP+valeur) := (AC)

En pratique, pour éviter le cas de l'addition, on utilise plutôt un registre de "segment" ou de "base" qui donne la partie haute de l'adresse, la partie basse étant fournie par l'instruction.

Ces mécanismes rendent le code "relogeable". L'intérêt de la chose apparaîtra dans le cours consacré au système d'exploitation.

Nommer un type de programme utilisant ce mécanisme.

Les virus. C'est plutôt d'actualité... Le gros inconvénient de pouvoir manipuler une instruction comme étant une donnée est qu'il faut prévoir ce type d'usage malfaisant.