Architecture et Système

Stefan Schwoon

Cours L3, 2016/17, ENS Cachan

Multiplexeur

Supposons qu'on possède k valeurs en entrée (ou $k = 2^n$), pour $n \geq 1$), ou chacune des valeurs $x_0, \ldots, x_{k-1}$ est un mot de m bits.

On souhaite sélectionner l'une de ces valeurs par son indice, p.ex. étant donné s (codé par n bits), on souhaite renvoyer x_s.

Applications:

Lire un mot dans la mémoire en spécifiant son adresse.

Choisir entre plusieurs sources de données.

Obtenir une valeur dans un tableau pré-calculé.

Multiplexeur : Exemple

Supposons $m = n = 1$, du coup on possède deux bits x_0, x_1 et un bit de sélection s_0.

Solution: on réalise la fonction $(x_0 \wedge \neg s_0) \vee (x_1 \wedge s_0)$.

Supposons $m = 1, n = 2$, du coup on posséde $x_0, \cdots, x_3$ et $s_1 s_0$:

Dans un premier temps, on évalue s_0 pour sélectionner (en parallèle) entre x_0, x_1 et x_2, x_3.

Dans un deuxième temps, on évalue s_1 pour sélectionner parmi les deux bits choisis avant.

Le problème se géneralise pour $n > 2$ avec un circuit de profondeur $\mathcal{O}(n)$.

Pour $m > 1$: On utilise m circuits en parallèle, un par bit.

Décodeur

On possède une valeur s codée par n bits et 2^n sorties $y_0, \ldots, y_{2^n-1}$.

Sortie : $y_s = 1$ et $y_{s'} = 0$ pour tout $s' \neq s$.

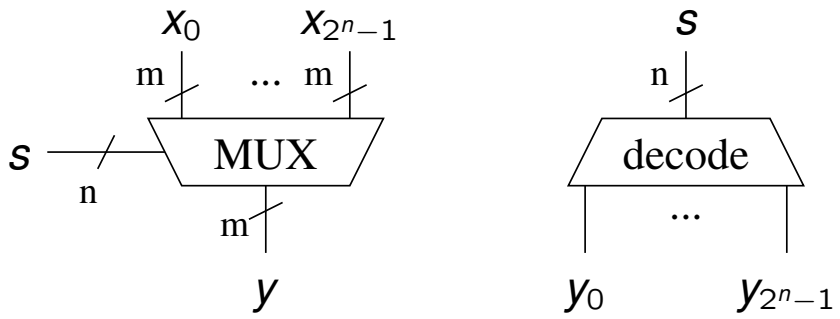
Solution (facile) : Pour tout $0 \leq s' < 2^n$, on construit la conjonction qui évalue si $s' = s$.

Applications :

- Sélectionner une cellule spécifique dans la mémoire.
- Sélectionner un périphérique parmi plusieurs pour échanger des données.

Symboles

Les symboles typiquement utilisés pour les multiplexeurs et décodeurs :



Les barres diagonales indiquent que le fil transfère un mot de la taille indiquée à côté (c'est à dire il s'agit d'une collection d'autant de bits).

Verrous et bascules

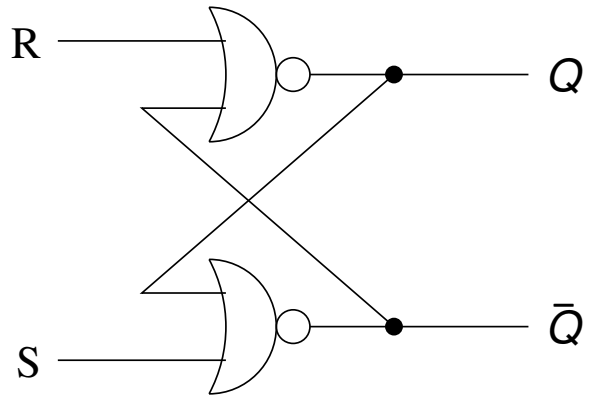
Ce sont des circuits utilisés pour stocker des bits. La littérature distingue parfois deux types de circuits :

- les circuits non-temporisés sont appelés verrous (*latch* en anglais)
- les circuits temporisés sont appelés bascules (*flip-flop* en anglais)

Il y a plusieurs types de verrous et bascules, selon des besoins spécifiques. Quelques éléments typiques :

- contrôles : aucun effet quand tous les fils de contrôle sont 0
- signal d'horloge (bascules seulement)
- deux sorties Q et $\bar{Q}$, la valeur du bit et son complément

Verrou RS



controls R (reset) and S (set):

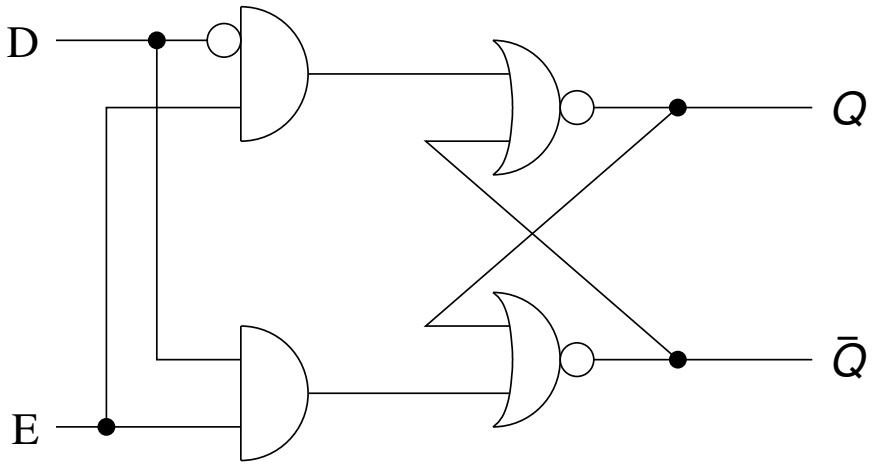
$R = 0$, $S = 1$: après peu de temps, $Q = 1$ et $\bar{Q} = 0$

$R = 1$, $S = 0$: après peu de temps, $Q = 0$ et $\bar{Q} = 1$

$R = S = 0$: les deux sorties restent inchangées

$R = S = 1$: on obtient $Q = \bar{Q} = 0$, devient instable lorsqu'on on revient à $R = S = 0$

Verrou D

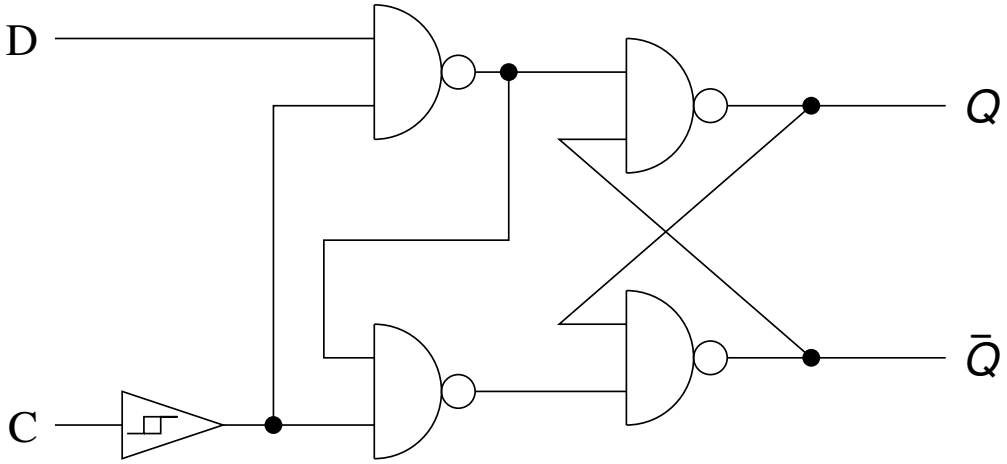


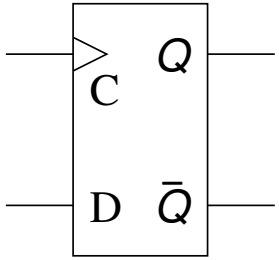
contrôles D (data) et E (enable):

- $E = 0$: stable
- $E = 1$: Q devient D

Bascule D

Reprenons le verrou D, mais en remplaçant *E* par un signal oscillant *C* (*clock* = horloge). Dans ce cas, *D* n'est pris en compte que si *C* est dans sa phase haute.





Dans les diagrammes, le signal d'horloge est typiquement indiqué par un triangle.

Quand le signal C est partagé parmi tous les bascules du système, ceci permet de synchroniser les calculs dans le système entier.

Le délai entre les “1” de C doit être suffisamment long pour que tous les calculs en parallèle puissent réussir.

En plus, les “1” doivent être suffisamment courts pour bien séparer les phases d’un calcul.

Mémoire vive

Les bascules/verrous sont un exemple d'une *mémoire vive*, avec les propritétés suivantes :

mode lecture ou écriture

volatile (contenu perdu sans alimentation électrique)

utilisé pour faire des calculs, stocker des données

en anglais: RAM (random-access memory, on accède facilement à toute adresse)

Mémoire morte

Par contre, une *mémoire morte* est dite d'avoir les propriétés suivantes :

mode lecture seulement (pas de moyen pour l'ordinateur ou le programmeur d'en modifier)

durable (même sans alimentation électrique)

utilisé pour assurer le fonctionnement de l'ordinateur (microprogrammation, tables mathématiques pré-calculées)

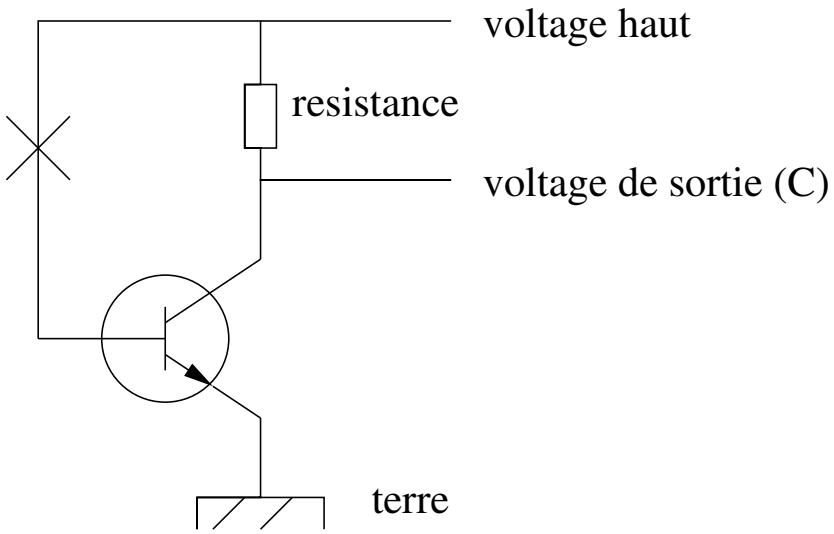
en anglais: ROM (read-only memory)

Types de mémoire morte

Façon la plus simple de produire un ROM :

prendre une série de transistors, chacun représentera un bit

physiquement interrompre le fil à la position X pour donner 1, sinon 0.



→ “programmable” une seul fois

→ appelé PROM quand la programmation se fait après la production initiale

EPROM

EPROM = Erasable programmable ROM
et EEPROM = electrically erasable PROM

Consiste des transistors avec une porte supplémentaire (“floating gate”) qui accède à une couche entre la base et les autres élements du transistor.

Le “floating gate” est programmable avec des charges électriques supérieur à ceux utilisé pendant l’opération normale de l’ordinateur.

$\rightarrow$ reprogrammable par l’extérieur, mode lecteur seulement pour l’ordinateur

Programmable logic array (PLA)

Circuit logique qui sert aussi à réaliser une mémoire morte

n bits d'entrée, m bits de sortie

tous les bits d'entrée disponible aussi en négation

Partie ET: $k \leq 2^n$ lignes, chacune choisit une conjonction d'entrées

Partie OU: m colonnes, tout bit de sortie est formé par la disjonction d'un sous-ensemble des k lignes

programmation consiste en choisissant les conjonctions et disjonctions

Architecture d'un ordinateur simple

Les éléments:

processeur (CPU), consiste d'une unité de contrôle (CU), unité arithmétique-logique (ALU), unité de microprogrammation et horloge

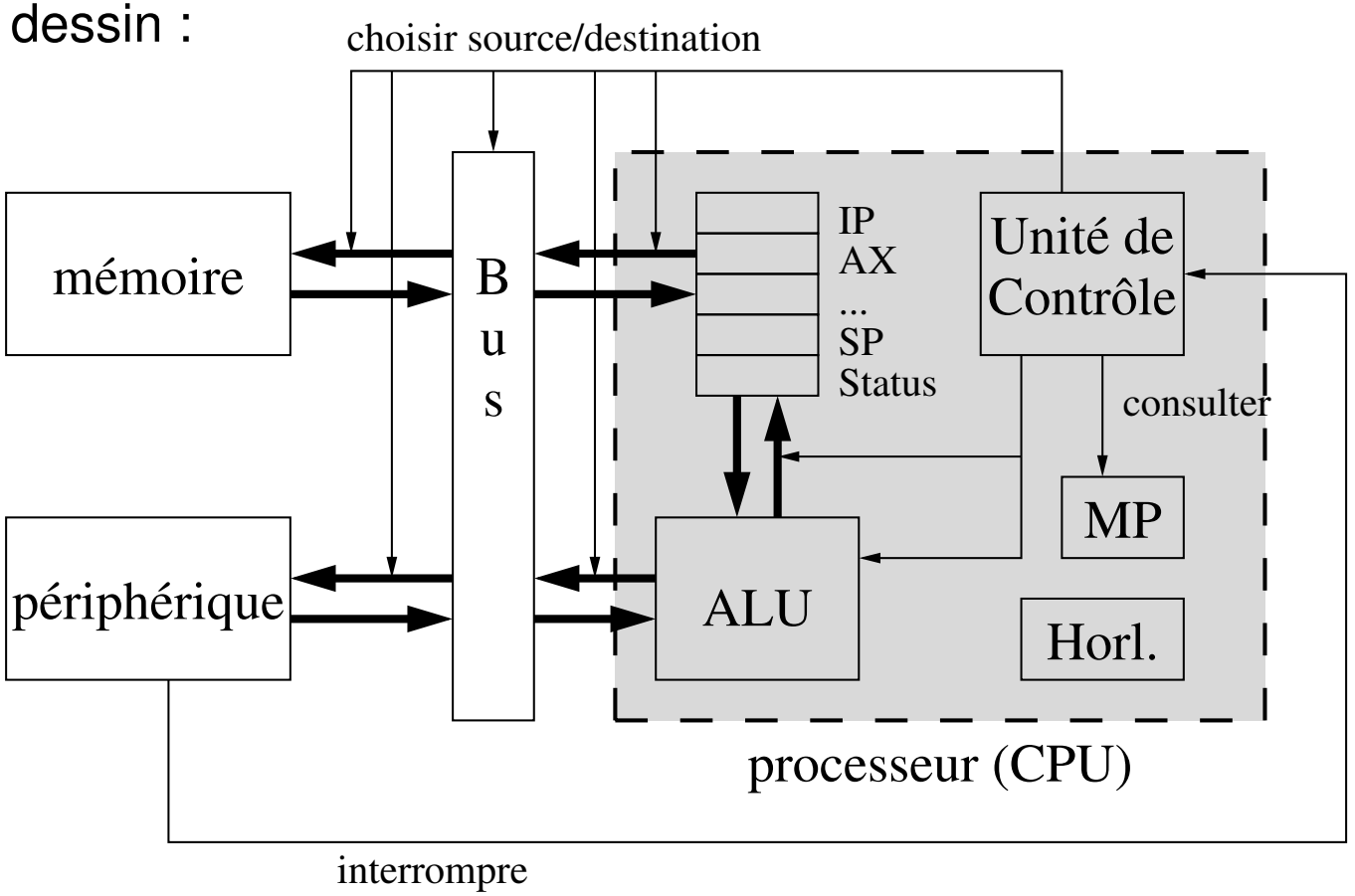
mémoire principale

les périphériques (fournissent des entrées/sorties pour les données)

le bus qui transfert des données entre les composants

Architecture

Un premier dessin :



Mémoire principale

Pour stocker données et instructions (architecture dite “von Neumann”)

Organisée en 2^n mots de m bits, les mots stockés à des *adresses* $0..2^n - 1$

Valeur typique pour m: 8; pour n: 30..34 (= 1..16 GB)

Réalisable, par exemple, avec des bascules D

Échanges avec le processeur :

lecture : multiplexeur pour choisir un mot à une adresse précise, transfert au bus

obtenir un mot et une adresse du bus, utiliseur décodeur pour l’écrire à la bonne destination

Le bus

Pour connecter CPU, mémoire, périphériques: tous composants peuvent lire et écrire

Utiliseur des multiplexeurs et décodeurs pour sélectionner source et destination du transfert

Un transfert à la fois (on peut pas obtenir deux mots de la mémoire au même temps, par exemple).

Le processeur central (CPU)

Contrôle toutes les opérations, avec les composants suivants:

registres: mèmoire de court terme pour les opérations actuelles

ALU: pour manipuler des données

control unit: pour coordonner les transferts, avec l'aide de l'horloge et l'unité de microprogrammation

Registres

Un registre stocke un mot (souvent plus qu'un mot de mémoire).

Il existe plusieurs registres pour des objectifs différents.

registres géneraux (accumulateur, AX, BX, . . .) : utilisés pour les calculs

compteur d'instructions : pointe vers la prochaine instruction à exécuter dans la mémoire

registre de statut : pour stocker quelques bits indiquant le succés ou non d'une opération (comparaison, division par zéro, . . .)

pointeur de pile (SP) : indique le sommet d'une pile qui sauvegarde des valeurs de certains registres, adresse de retour pour les sous-routines, etc

Unité de contrôle (CU)

Travaille dans des phases indiqués par l'horloge :

Chaque phase réalise une opération nécessaire pour exécuter une instruction:

Dans chaque phase, le CU doit organiser les transferts de données:

- donner les bons bits de sélection au bus
- transferer les donées entre ALU et registres
- réagir aux périquériques

Quelques phases typiques: obtenir une instruction, décoder une opération, exécuter une opération (en plusieurs phases)

Un ROM (unité de microprogrammation) peut servir pour fournir des signaux à donner aux autres élements

Phases

Phase 1 (Instruction fetch):

Laisser le bus utiliseur la valeur d'IP comme sélecteur d'adresse, transferer au CPU

Phase 2 (Decode):

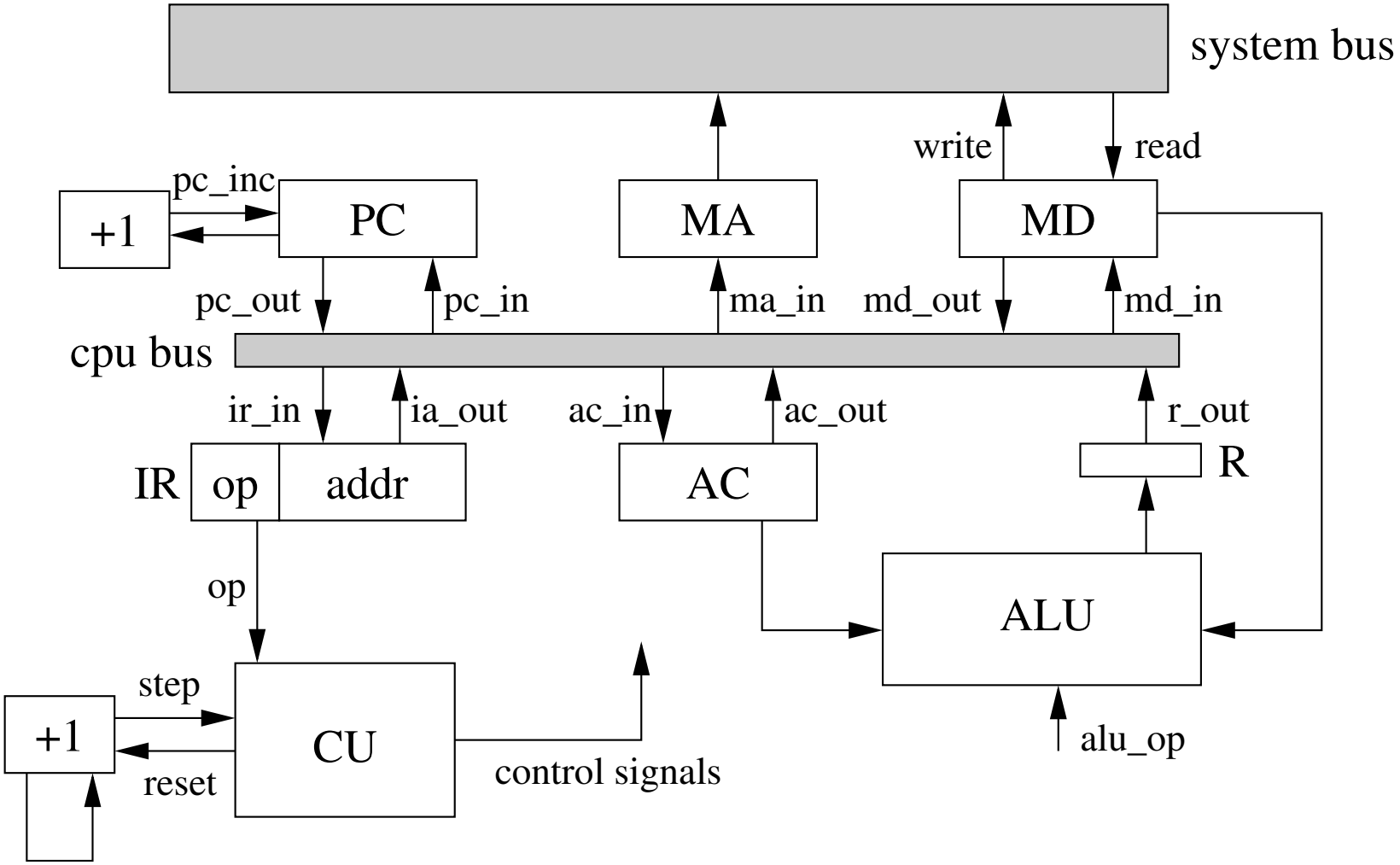
Consulter unité de microprogrammation pour organiser les transferts néseccaires pour l'opération

Phases 3, 4, . . . (Execute):

En utilisant le MU, organiser les bons transferts et manipulations dans chaque phase.

En fonction de leur complexité, les différentes instructions nécessitent un nombre différent de phases.

Aperçu d'un processeur



Ce dessin montre un bus à l'intérieur du CPU et un bus principal.

MA (memory address) et MD (memory data) communiquent avec la mémoire par le bus principal.

IR (instruction register) : on suppose ici qu'une instruction contient un code d'opération et une adresse (comme dans l'IAS).

L'unité de contrôle (CU) prend en compte l'instruction actuelle et un compteur de phase. Ces deux permettent d'obtenir les bons signaux de contrôle a fournir aux bus.

Exemples

LOAD *addr*: charger le contenur d'une adresse dans AC :

- phase 3: charger *addr* dans MA: (`ia_out`, `ma_in`)
- phase 4: obtenir données de la mémoire (`read`)
- phase 5: transfert vers l'ALU, puis prochaine opération (`md_out`, `alu_in`, `reset`)

ADD *addr*: rajouter contenu de *addr* à l'accumulateur

- phase 3: charger *addr* dans MA: (`ia_out`, `ma_in`)
- phase 4: obtenir des données (`read`)
- phase 5: faire l'addition (`alu_op=add`)
- phase 6: transfert vers ALU, prochaine opération (`r_out`, `alu_in`, `reset`)

Interruptions

Des périphériques (clavier, réseau) n'ont pas toujours besoin de communiquer avec le processeur, mais si besoin, il faut réagir vite.

Quand un périphérique souhaite échanger des données avec le processeur, il met son signal d'interruption à 1.

Quand le processeur est prêt pour la prochaine instruction, il vérifie s'il y a reçu un tel signal d'interruption et choisit celui avec la priorité la plus élevée.

Dans ce cas le processeur continue son travail à une certaine adresse en fonction du signal choisi. Cette adresse devrait contenir du code pour gérer la communication. Ensuite on continue l'exécution normale.