Université Joseph Fourier – Grenoble 1

Architecture des ordinateurs

Exercices sur les premières notions

Représentation de l'information - Circuits - Machine de von Neumann

Janvier 2012

I. Codages ascii et autre

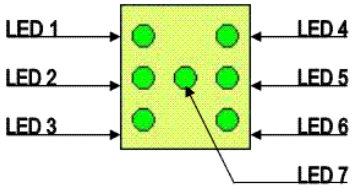
Q1. Évaluez la taille nécessaire pour la sauvegarde du texte de cette page sous forme ascii.
Q2. Évaluez la taille du fichier word de cette page.

On imagine maintenant de coder les mots d'une langue par un code unique comme ont été codées les lettres de l'alphabet en Ascii. Par exemple, le mot « maintenant » pourrait avoir le code 27, « unique » le code 135, etc.

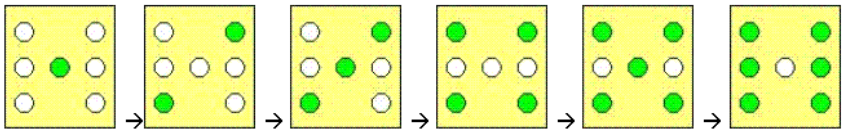
Q3. En suivant ce principe, donnez le nombre de bits nécessaires pour un mot dans un codage de l'ensemble des mots de la langue française. Note : la langue française comporte environ 900 000 mots et dérivés.
Q4. Évaluez la taille nécessaire à la sauvegarde du texte de ce devoir dans ce codage.

II. Circuit pour dé à 6 faces (barème indicatif : 8 points)

Pour représenter un dé à 6 faces, 7 diodes électroluminescentes sont utilisées :



Ce dispositif permet d'afficher les 6 configurations possibles (ci-dessous) :

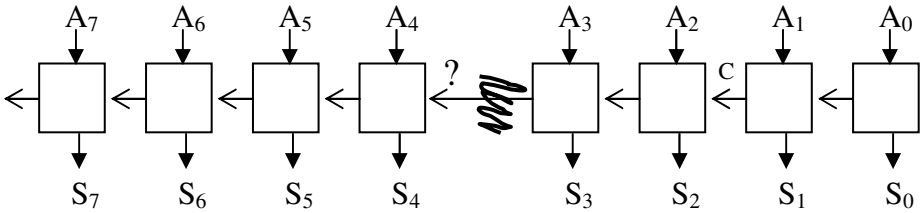


(Crédit illustrations F. Sincère)

Q1. Donner la table de vérité d'une fonction qui aurait pour entrées les 3 bits représentant un nombre à afficher et pour sorties l'état des 7 diodes corespondant.
Q2*. Dessiner le circuit correspondant (ou une partie significative). Évaluer sa complexité.
Q3*. On veut simuler l'affichage d'un lancé multiple et simultané de 5 dés. Un circuit produit le tirage aléatoire d'un nombre entre 5 et 30 codé sur 5 bits. En reprenant ce qui précède et en adjoignant les circuits qui vous sembleront adéquates (diviseur, mux, comparateur, …), définir le plan général d'un circuit qui alimente l'affichage des 5 dés à partir de l'entrée donnée par le circuit de tirage aléatoire.

III. Incrémenteur avec anticipation de retenue (barème indicatif : 6 points)

Pour réaliser un incrémenteur (additionneur simplifié pour faire « +1 » seulement), un concepteur de circuit imagine un assemblage avec propagation des retenues (cf. additionneur), ex pour 8 bits (dessin du projet) :



Une différence importante souhaite être introduite entre cet incrémenteur et l'additionneur : la retenue après les calculs pour le bit 3 -nécessaire pour le calcul du bit 4 (retenue « raturée » sur le dessin)- ne doit pas être obtenue à partir de A_3 et de la retenue précédente (comme pour les autres retenues) ; cette retenue doit être anticipées un temps plus tôt, dès la connaissance de A_2, A_3 et de la retenue « C » (retenue transitant entre les calculs pour le bit 1 et le bit 2).

Q1. Calcul de la retenue anticipée. Donner la table de vérité du calcul de la retenue « ? » pour le calcul du bit 4 (après la « rature » sur le dessin) en fonction de A_2, A_3 et C. Redessiner l'incrémenteur avec le circuit correspondant au calcul de cette retenue anticipée. Ajouter le dessin du plus long chemin déterminant la complexité en temps de ce circuit. Évaluer le gain en temps possible de ce circuit par rapport au temps de calcul du circuit sans anticipation.

Q2. Anticipation multiple. Avec une seule anticipation de retenue, le gain en temps risque d'être faible. Comparer deux généralisations de l'anticipation : celle où toutes les retenues seraient anticipées et celle où

une retenue sur 2 (alternativement) serait anticipée. Choisir l'une de ces deux généralisations et dessiner le circuit correspondant, ainsi que le plus long chemin. Justifier votre choix. Évaluer le gain en temps possible.

Q3. Au delà. Au choix, proposer une adaptation aux additionneurs, ou d'autres calculs d'anticipation pour les incrémenteurs.

IV. Circuits réversibles.

Dans ses leçons sur l'informatique, Richard Feynman, prix Nobel de physique en 1965, s'intéresse à des circuits ≪ exotiques ≫ : ≪ Les opérations AND et NAND (ainsi que OR et XOR) sont des opérations irréversibles. J'entends par la qu'il est impossible de récupérer le signal d'entrée à partir du signal de sortie : l'information initiale est perdue et l'est de manière irréversible. Une sortie de porte AND à quatre entrées qui vaut 0 peut provenir de quinze combinaisons des signaux d'entrée, et rien ne nous permet de deviner laquelle (en revanche, si la sortie vaut 1, nous pouvons en déduire quelle est la combinaison à l'entrée !). Je voudrais introduire le concept d'opération réversible comme d'une opération dont le signal de sortie recèle suffisamment d'informations pour permettre d'en déduire l'entrée.≫

QII-1. Le texte de R. Feynman comporte quelques implicites. Quelles sont les quinze + 1 combinaisons dont il est question ?

QII-2. Montrez que la porte OR est irréversible.

QII-3. Qu'en est-il de la porte NON ? Est-elle réversible ? Pourquoi ?

Plus loin dans son texte, R. Feynman donne l'exemple d'une porte réversible avec la porte CCN (Controlled Controlled Not) ayant trois entrées (A , B, C) et trois sorties (A' , B' , C') :

A B C	A' B' C'
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 1
1 1 1	1 1 0

QII-4. Réalisez cette porte à l'aide d'un circuit combinatoire simple (comportant seulement NOT, AND et OR). Donnez sa complexité.

QII-5. Quelle est la table de vérité du circuit comportant deux portes CCN, les sorties de la première servant d'entrée pour la seconde.

QII-6. Rappelez la table de vérité du demi-additionneur et un circuit réalisant cette fonction. Ajouter, si nécessaire, entrée(s), sortie(s) et porte(s) logique(s) pour obtenir un circuit réversible.

QII-7. A partir du circuit obtenu à la question précédente donnez le circuit d'un additionneur 4 bits réversibles.

V. Addition récursive de Von Neumann

L'addition récursive de Von Neumann repose sur 2 principes : 1 - anticiper les retenues possibles lors des calculs d'une addition en effectuant 2 fois plus de calculs à un moment donnée (un calcul avec une hypothétique retenue entrante à 0 et un calcul avec une hypothétique retenue entrante à 1) ; 2 - décomposer les calculs de manière dichotomique.

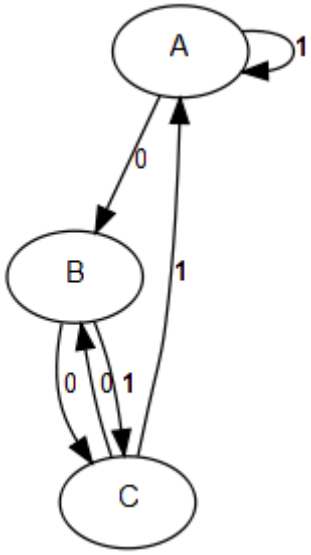
Q1. Cas de base. Pour la somme de 2 nombres (A, B) codés sur 1 bit, donner la table de vérité du circuit de base qui effectue les calculs en doubles : avec une hypothétique retenue entrante à 0 et avec une hypothétique retenue entrante à 1. En entrée, il doit y avoir 2 nombres (A, B) codés sur 1 bit (A_0, B_0), en sortie, il doit y avoir 2 résultats (R, S), codés sur 2 bits (R_1 R_0, S_1 S_0), et l'on doit avoir R=A+B, S=A+B+1. À partir ce cette table de vérité, on suppose la définition d'un circuit $\mathbf{C}_1$: (A_0, B_0) => (R_1 R_0, S_1 S_0). Donner le circuit correspondant.

Q2. Premier niveau. A l'aide de multiplexeurs, assembler 2 circuits $\mathbf{C}_1$ obtenus à la question précédente pour obtenir un circuit $\mathbf{C}_2$ effectuant la somme de 2 nombres (A, B) codés sur 2 bits, en effectuant les calculs en doubles : avec une hypothétique retenue entrante à 0 et avec une hypothétique retenue entrante à

1. En entrée il doit y avoir 2 nombres (A, B) codés sur 2 bits ($A_1\,A_0$, $B_1\,B_0$), en sortie, il doit y avoir 2 résultats (R, S), chacun sur 3 bits ($R_2\,R_1\,R_0$, $S_2\,S_1\,S_0$), et l'on doit toujours avoir R=A+B, S=A+B+1.
Indications : Pour obtenir cet assemblage, utiliser un premier circuit $\mathbf{C_1}$ avec (A_0, B_0), et un second circuit $\mathbf{C_1}$ avec (A_1, B_1). Combiner les résultats de ces circuits avec des multiplexeurs pour obtenir ($R_2\,R_1\,R_0$, $S_2\,S_1\,S_0$). $R_0\,S_0$ est obtenu à partir du premier circuit $\mathbf{C}_1$. Les deux autres sorties de ce premier circuit $\mathbf{C}_1$ servent à commander les multiplexeurs (ce sont les retenues entrantes effectives du second circuit, elles peuvent valoir toutes les deux 0, ou toutes les deux 1, ou être différentes). En entrées de ces multiplexeurs on trouve également les sorties du second circuit $\mathbf{C}_1$. (Pas de table de vérité à cette question, juste le circuit)
Q3. Généralisation. Comme dans la question précédente, à l'aide de multiplexeurs et de 2 circuits $\mathbf{C}_n$ effectuant la somme de 2 nombres (A, B) codés sur n bits et donnant 2 résultats (R,S), chacun sur n+1 bits tels que R=A+B, S=A+B+1, obtenir un circuit $\mathbf{C}_{2n}$ effectuant la somme de 2 nombres (A, B) codés sur 2n bits et donnant 2 résultats (R,S), chacun sur 2n+1 bits tels que R=A+B, S=A+B+1.
Q4. Complexité. D'après votre table de vérité (Q1), évaluer la complexité du circuit $\mathbf{C}_1$. À partir de Q2, évaluer la complexité du circuit $\mathbf{C}_2$. Enfin, évaluer la complexité du circuit $\mathbf{C}_{32}$ et la comparer avec la complexité du circuit d'addition classique.

VI. Réalisation d'un petit automate (barème indicatif : 7 points)

Pour l'automate dessiné ci-contre :



Q1. Modélisation. Définir les états, les entrées. Associer un codage binaire. Avec ce codage, expliciter la fonction de transition (sous la forme d'une table de vérité).
Q2. Réalisation de la transition. Dessiner un circuit réalisant la fonction de transition.
Q3*. Réalisation de l'automate. Donner le dessin du circuit global qui réalise l'automate. Donner un chronogramme du fonctionnement de ce circuit partant de l'état A et lisant les entrées 0, 1, 0.

VII. Circuits d'un petit automate

Un automate de Moore est donné par la modélisation de ses états, de ses entrées, de ses sorties et par les tables de vérité de ses fonctions de sortie et de transition :

Etat	Représentation binaire
Init	00
tmp	01
Final	10

Entrées	Rep. bin.
« 0 »	0
« 1 »	1

Sortie	Rep. bin.
Pas Ok	0
Ok	1

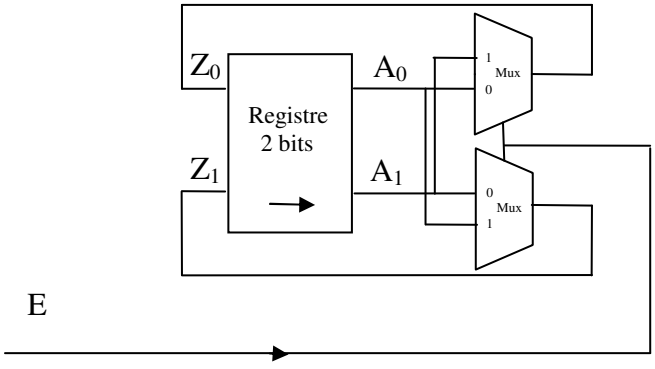
Fonction de sortie : Etat	
00	0
01	0
10	1

Transition : Etat	Entré	
00	0	00
00	1	01
01	0	01
01	1	10
10	0	10
10	1	10

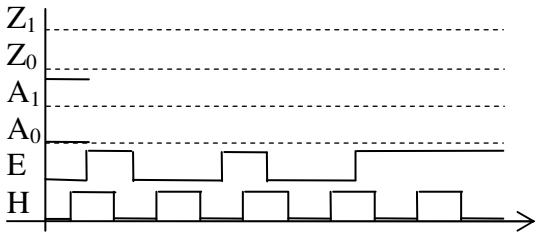
Q1. Dessiner l'automate.
Q2. Dessiner un circuit qui réalise cet automate.
Q3. Dessiner le chronogramme qui correspond à la lecture de la suite d'entrées « 0 1 0 » (chaque valeur en entrée étant disponible sur une période d'horloge, l'automate étant dans l'état Init au départ).

VIII. Circuit séquentiel avec multiplexeur (barème indicatif : 6 points)

Pour le circuit séquentiel avec 1 registre (2 bits) et 2 multiplexeurs$_{2\to1}$ suivant :



Q1*. Modélisation Inverse. Définir la table de transition. Dessiner l'automate correspondant.
Q2*. Compléter le chronogramme. Indication : bascule du registre sur front descendant d'horloge.



IX. Calcul de la puissance

Deux algorithmes sont proposés pour calculer R la puissance n d'un nombre X ($R \leftarrow X^n$).

Algorithme 1 :
$R \leftarrow 1$
Pour i allant de 1 à n faire :
  $R \leftarrow R*X$

Algorithme 2 :
$R \leftarrow 1$
Tant que n != 0 faire :
  Si n est impair :
    $R \leftarrow R*X; N \leftarrow N-1$
  $X \leftarrow X^2$
  $N \leftarrow N/2$

Q1. Choix de l'algorithme à implémenter. Choisir entre les deux algorithmes celui que vous comptez implémenter, donnez vos raisons.
Q2. Implémentation sous la forme d'un circuit à flot de donnée. Donner un circuit type « flot de données » réalisant l'exécution du calcul.
Q3. Implémentation sous la forme d'un circuit PC/PO. Donner le dessin de l'automate de la partie PC d'un circuit « PC/PO » réalisant le calcul.

X. Valeurs 0, 1, 11, 111, …, 111, 11, 1, 0 (barème indicatif : 6 points)

Dans le petit programme ci-dessous, la variable X (codée sur un octet) possède, successivement, les valeurs binaires suivantes : 0, 1, 11, 111, …, 111, 11, 1, 0.

```
Début
  X ← 0
  Pour i=0..16
    | Si i<8 Alors X ← 2*X+1 Sinon X ←X/2 FinSi
  FinPour
Fin.
```

Q1*. Implémentation sous la forme d'un circuit à flot de donnée. Donner un circuit type « flot de données » réalisant l'exécution de la boucle du programme (les initialisations ne sont pas demandées).
Q2*. Implémentation sous la forme d'un circuit PC/PO (Partie Contrôle/Partie Opérative). Donner le dessin de l'automate de la partie PC (partie contrôle) d'un circuit PC/PO réalisant ce programme (vous pouvez utiliser sans la définir une PO (partie opérative) semblable à celle donnée en cours et en TD).

XI. Instruction Add à 3 adresses (barème indicatif : 8 points)

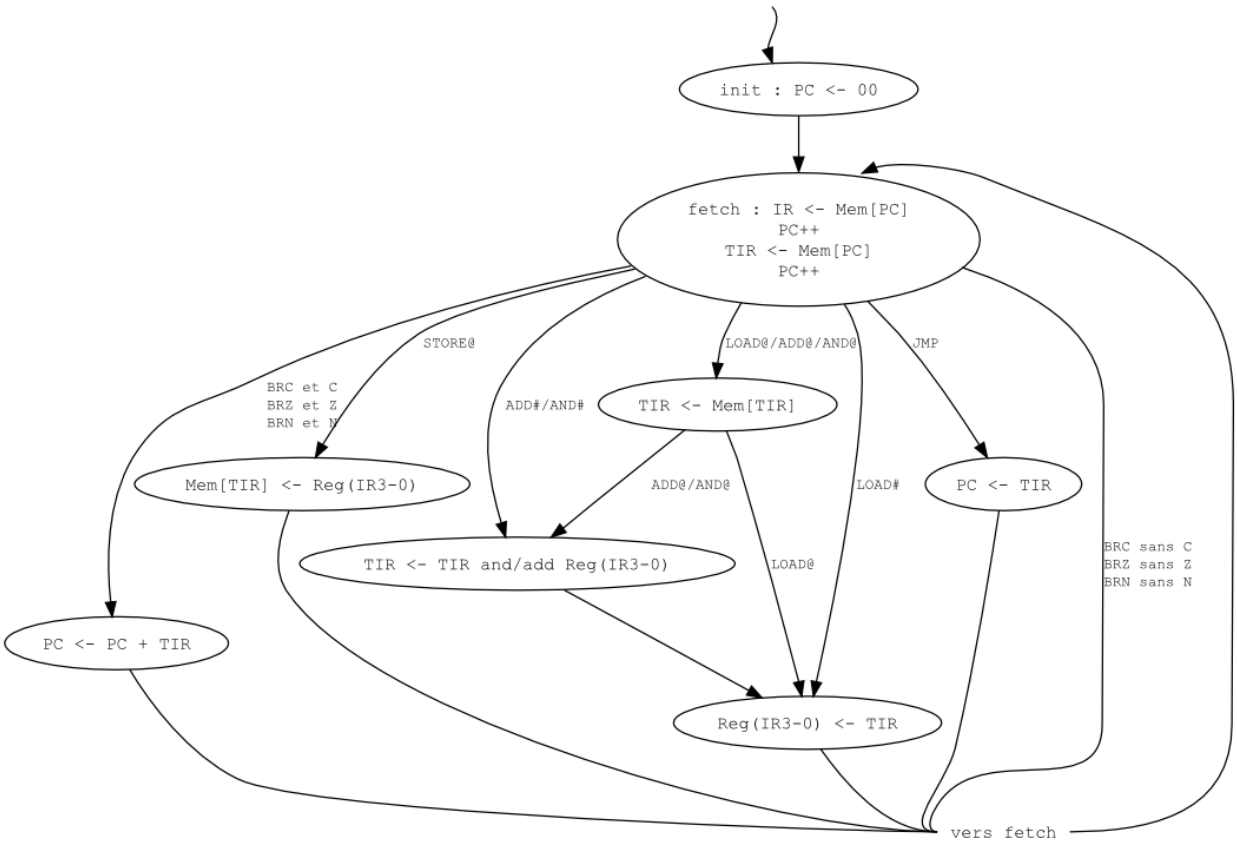
Prenons la machine vue en cours. Cette machine possède le langage machine suivant :

- LOAD#/ADD#/AND# en mode d'adressage immédiat, codage sur 2 octets, avec resp. les codes binaires 1, 0, 2 dans le quartet de poids fort du premier octet de l'instruction, le numéro du registre paramètre de l'instruction dans le quartet de poids faible de cet octet et la valeur immédiate dans le second octet de l'instruction
- LOAD@/STORE@/ADD@/AND@ en mode d'adressage direct, codage sur 2 octets, avec resp. les codes binaires 5, 7, 4, 6 dans le quartet de poids fort du premier octet de l'instruction, le numéro du registre paramètre de l'instruction dans le quartet de poids faible de cet octet et l'adresse de l'opérande dans le second octet de l'instruction
- et divers sauts JMP/BRC/BRZ/BRN en adressage absolu ou relatif, codage sur 2 octets, avec resp. les codes binaires C0, C1, C9, CD dans le premier octet de l'instruction et l'adresse du saut ou la valeur de décalage dans le second octet de l'instruction

Rappel de quelques instructions :

Mode d'adressage	Mnémo	Codage binaire	Interprétation
Immédiat	LOAD Reg, #i	1(N°Reg) puis i	Reg ←i, PC ← PC + 2
Absolu	JMP @ad	C0 puis ad	Pc ←ad

Rappel de l'automate de contrôle pour l'interprétation de ce langage machine :



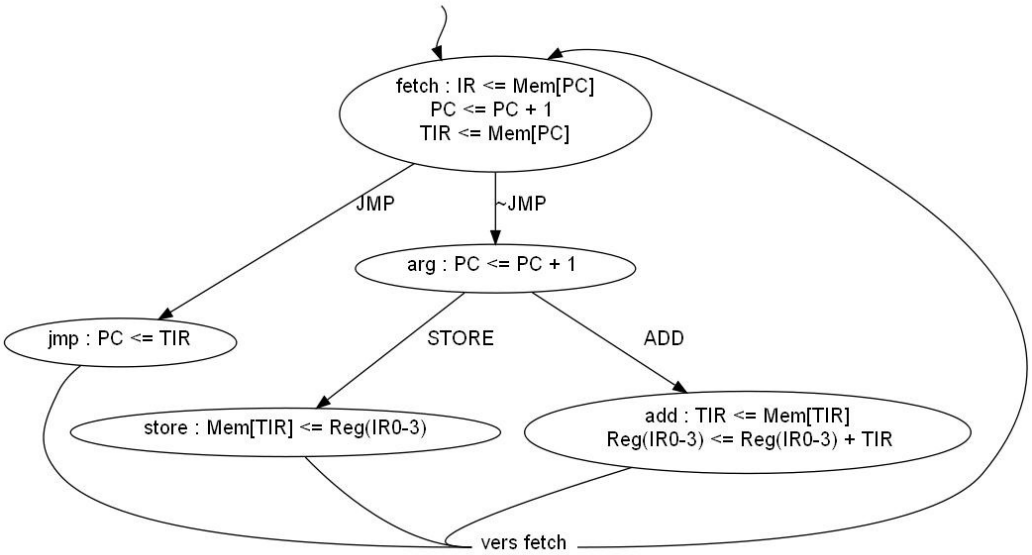
Cet exercice porte sur l'introduction de nouvelles instructions assembleurs dans le langage interprété par l'automate de contrôle de cette machine.
Les instructions en question sont des instructions permettant de faire la somme de deux registres, le résultat étant stocké dans un troisième registre. Le mnémonique choisi pour cette instruction est Ad3. Dans une première version, les trois registres sont implicitement A, B, C et l'interprétation de l'instruction « $Ad3_1$ » en pseudo langage est le suivant :
A←B+C ; Pc←Pc+1 (ou 2 selon la taille de l'instruction)
Q1. Proposer un codage pour cette instruction $Ad3_1$, code binaire de l'instruction, éventuellement nombre d'opérandes, place des opérandes
Q2. Modifier le graphe de contrôle pour prendre en compte cette instruction.
Seconde version, les trois registres doivent pouvoir être définis dans l'instruction « $Ad3_2$ »
Q3. Proposer un codage pour cette instruction $Ad3_2$, code binaire de l'instruction, éventuellement nombre d'opérandes, place des opérandes
Q4. Modifier le graphe de contrôle pour prendre en compte cette instruction.
Q5. Donner le temps d'exécution des instructions $Ad3_1$ et $Ad3_2$ (discuter d'éventuelles hypothèses à prendre sur la mise en œuvre du graphe de contrôle)
Q6. Ces instructions assembleurs pouvaient-elles être simulées avec l'ensemble d'instructions disponibles pour la machine initiale ? Si oui, comparer les mises en œuvre.

XII. Ré-ordonnancement dynamique (barème indicatif : 8 points)

Q1*. Simuler l'exécution de 6 instructions assembleur pour le programme suivant en prenant l'automate de contrôle décrit ci-après et calculer le temps pris pour l'exécution de ces 6 instructions.

```
Prog :  Add A, 21
        Store A, 20
        Jmp 0                   /* => JMP Prog */
```

Indications : la mémoire comporte au départ le programme à l'adresse 0 (et suivantes) ;
aux adresse 20, 21 se trouvent les valeurs 5 et 10 ;
les registres PC et A sont initialement à 0 (initialisations non représentées dans le graphe de contrôle) ;
le temps d'exécution d'une étape $Registre_{fixe}$ <= $Registre_{fixe'}$ + $Registre_{fixe''}$ ou $Registre_{fixe}$ <= $Registre_{fixe'}$ est le temps de base (1),
accéder à un Registre via son numéro, stocké dans les bits de poids faible du registre instruction (Reg_{IR0-3}) nécessite un temps de plus (+1),
accéder à la mémoire en lecture nécessite 3 temps de plus (+3),
accéder à la mémoire en écriture nécessite 4 temps de plus (+4).



Pour effectuer des ré-ordonnancements dynamiques (exécutions des instructions assembleur dans un autre ordre que celui prévu par le programmeur) 2 registres sont ajoutés : X-IR, et X-TIR pour pouvoir lire et traiter les instructions assembleur en connaissant 2 instructions à exécuter à un moment donné (la première dans IR/TIR, la suivante dans X-IR/X-TIR).

Q2*. Ajouter au graphe de contrôle, la lecture d'une seconde instruction dans X-IR/X-TIR :

- pensez à l'initialisation (où deux lectures complètes d'instruction sont nécessaires),
- au cas courant (où X-IR/X-TIR étant connu, ils prennent la place de IR/TIR ; la lecture complète d'une seule nouvelle instruction dans X-IR/X-TIR étant nécessaire),
- et au cas où une rupture de séquence rend la valeur X-IR/X-TIR caduque (et où, par conséquent, deux lectures complètes d'instruction sont nécessaires).

Calculer le temps d'exécution du programme de la **Q1** sur ce nouveau graphe.

Q3. Deux instructions étant connues, cf. **Q2**, où peut se situer, dans le graphe, l'endroit où on effectuerait le ré-ordonnancement (exécution des deux instructions dans un autre ordre que celui prévu par le programmeur : soit l'avancement de l'exécution de la seconde instruction, soit le retardement de l'exécution de la première instruction) ? Dans quels cas ce ré-ordonnancement pourrait avoir lieu ? Proposez un exemple.