Travaux Dirigés d'architecture des machines

1 Planning du cours Architecture des Machines

- I. Histoire de l'informatique
- II. Représentation des informations et arithmétique
- III. Circuits logiques et algèbre de Boole
- IV. Présentation générale
- V. Le processeur
- VI. Assembleur
- VII. La mémoire
- VIII. Les entrées/sorties
- IX. Performances

2 Bibliographie

Carter Nicholas P., *Architecture de l'ordinateur*, EdiScience, 2002.
Cazes A. & Delacroix J., *Architecture des machines et des systèmes informatiques*, 4e édition, Dunod, 2011.
Hennessy J.L. & Patterson D.A., *Architecture des ordinateurs*, 3e édition, Vuibert, 2003.
Lazard E., *Collection Synthex, Architecture de l'ordinateur*, Pearson Education, 2006.
Tanenbaum A., *Architecture de l'ordinateur*, 5e édition, Pearson Education, 2005.
Zanella P., Ligier Y. & Lazard E., *Architecture et technologie des ordinateurs*, 6e édition, Dunod, 2018.

© Emmanuel Lazard 2022 - Reproduction non autorisée sans l'accord de l'auteur.

3 Exercices représentation des nombres

Exercice 1 *Exprimer le nombre décimal* 100 *dans toutes les bases de* 2 *à* 9 *et en base* 16.

Exercice 2 *Multiplier* 0111 *et* 0101 *en binaire.*

Exercice 3

1. *Exprimer les nombres décimaux* 94, 141, 163 *et* 197 *en bases* 2, 8 *et* 16.
2. *Donner sur* 8 *bits les représentations*
   (a) *signe et valeur absolue ;*
   (b) *complément à 1 ;*
   (c) *complément à 2*

   des nombres décimaux 45, 73, 84, −99, −102 *et* −118.
3. *Effectuer la soustraction de* 122 *et* 43 *dans la représentation en complément à 2 en n'utilisant que l'addition.*

Exercice 4

1. *On considère l'opération décimale* $101 + 99$. *Expliquer pourquoi un programme peut « donner » deux résultats différents. Quels sont ces résultats et de quoi dépend le fait que le programme donne l'un ou l'autre ?*
2. *Votre nouveau programme est en train de compter le nombre de caractères d'un fichier. Vous savez qu'il y en a environ 50 000. Finalement votre programme s'arrête et affiche* `-14532` *caractères. Quel bug avez-vous fait et quel est le nombre de caractères du fichier ?*
3. *On considère le programme C suivant :*

Listing 1. Calculs

```c
#include <stdio.h>

main() {
    char c;
    char str[500];
    gets(str);       /* récupère une chaîne au clavier et la met dans str */
    c = strlen(str); /* renvoie la longueur de la chaîne */
    printf("longueur : %d\n", c);
}
```

   Pensez-vous que ce programme affiche toujours la bonne valeur ?
   Quelles sont ses limites de fonctionnement ?
4. *On considère le programme C suivant :*

Listing 2. Signé/non-signé

```c
#include <stdio.h>

main() {
    short i;
    unsigned short j;

    i = -1;
    j = i;
    printf("%d\n", j);
}
```

   Quelle est la valeur affichée ?
   Quelle serait la valeur affichée si la représentation des nombres était « signe et valeur absolue » ? Et si c'était la représentation en complément à un ?

5. *Le langage C possède plusieurs types pour représenter les nombres entiers. Lesquels ?*
   Quels sont les types entiers en Java ?
   Quelle question vous posez-vous avant de choisir un type pour votre variable ? Quelles sont les valeurs critiques ?

Exercice 5 *On considère une représentation simplifiée des réels en virgule flottante.*

Un nombre réel X *est représenté par* 10 *bits* $\boxed{s\,eeee\,mmmm}$ *où* $X = (-1)^s * 1{,}m * 2^{e-7}$ *avec un exposant sur 4 bits (*$0 < e \leqslant 15$*, un exposant égal à* 0 *désigne le nombre* 0 *quelle que soit la mantisse) et une pseudo-mantisse sur 5 bits (représentant les puissances négatives de* 2*).*

1. *Quels sont le plus grand nombre et le plus petit nombre strictement positifs représentables ?*
2. *Comment se représente le nombre* 1 *? La précision* ϵ *d'une représentation est l'écart entre* 1 *et le nombre représentable suivant. Combien vaut* ϵ *pour cette représentation ?*
3. *Peut-on représenter* 7,2 *et* 57,6 *Quels sont les nombres qui encadrent* 7,2 *et* 57,6 *dans cette représentation ?*
4. *Que donne en décimal la multiplication de* 7,25 *et* 28,5 *? Écrivez* 7,25 *et* 28,5 *dans la représentation proposée et effectuez la multiplication. Quels sont les deux arrondis possibles pour le résultat et leur valeur décimale ?*

Exercice 6 *On considère une représentation simplifiée des réels en virgule flottante.*

Un nombre réel X *est représenté par* 10 *bits* $\boxed{s\,eeee\,mmmm}$ *où* $X = (-1)^s * 1{,}m * 2^{e-7}$ *avec un exposant sur 4 bits (*$0 < e \leqslant 15$*, un exposant égal à* 0 *désigne le nombre* 0 *quelle que soit la pseudo-mantisse) et une pseudo-mantisse sur 5 bits (représentant les puissances négatives de* 2*).*

1. *Représenter* 16, 10 *et* 0,75 *en virgule flottante.*
2. *Pour additionner deux nombres en virgule flottante, il faut décaler la mantisse d'un des deux nombres pour égaliser les exposants, additionner les mantisses (sans oublier le* 1 *débutant la mantisse), puis renormaliser le nombre en arrondissant éventuellement la pseudo-mantisse.*
   Ainsi, on a $7 = 1{,}11000_2 \times 2^2$ *et* $1{,}25 = 1{,}01000_2 \times 2^0 = 0{,}0101000_2 \times 2^2$ *dont l'addition donne* $10{,}0001_2 \times 2^2$ *qui se normalise en* $1{,}00001_2 \times 2^3 = 8{,}25$.
   Expliciter l'addition flottante de 16 *et* 10*. Que donne l'addition de* 16 *et* 0,75 *si on arrondit inférieurement ?*
3. *On écrit le code suivant :*

Listing 3. Calculs

```
float f = 16.0;
for (i = 0; i < 10; i++)
    f = f + 0.75;
```

Quelle valeur aura la variable `f` *après l'exécution ? Et quelle aurait été sa valeur si on avait simplement écrit :*

```
float f = 16.0 + (10 * 0.75);
```

4 Exercices circuits logiques

Exercice 7 *Soit un entier de* 0 *à* 7 *représenté par* 3 *bits* $b_2b_1b_0$. *Soit* F *la fonction de* $[0,7] \longrightarrow \{0,1\}$ *telle que* $F(0) = F(3) = F(4) = F(7) = 1$ *et* $F(1) = F(2) = F(5) = F(6) = 0$.

1. *Donner la table de vérité de* F.
2. *Donner l'expression algébrique de* F.
3. *Simplifier* F.
4. *Représenter le circuit logique de* F *uniquement à l'aide de portes NAND.*

Exercice 8 *Soit une machine qui travaille sur des nombres binaires signés de 3 bits. La valeur signée d'un mot* $a_2a_1a_0$ *est égale à* $a_0 + 2a_1 - 4a_2$ *(c'est la représentation classique en complément à 2).*

On désire construire un circuit qui donne en sortie $b_2b_1b_0$ *l'opposé de la valeur binaire d'entrée (par exemple, si on a* 2 *en entrée, on veut la valeur binaire* -2 *à la sortie).*

1. *Toutes les valeurs sont-elles autorisées en entrée ? Autrement dit, le circuit donne-t-il toujours une valeur correcte ?*
2. *Donner la table de vérité du circuit pour toutes les valeurs autorisées.*
3. *Donner les expressions logiques des* 3 *bits de sortie en fonction des* 3 *bits d'entrée en simplifiant au maximum les expressions et en ne tenant pas compte des valeurs interdites.*
4. *Donner l'expression logique d'un bit supplémentaire en sortie,* e (débordement), *qui indiquerait qu'une valeur interdite est présente sur les entrées.*
5. *Pouvez-vous généraliser certaines de ces expressions à des nombres de* n *bits ? Plus précisément, montrer que pour un circuit donnant l'opposé d'un nombre signé sur* n *bits, le bit de poids fort et le bit de poids faible de la sortie s'expriment facilement en fonction des bits d'entrée. Même question pour le bit de débordement.*

Exercice 9 *Soit une machine qui travaille sur des nombres binaires signés de 4 bits. La valeur signée d'un mot* $A = a_3a_2a_1a_0$ *est égale à* $a_0 + 2a_1 + 4a_2 - 8a_3$ *(c'est la représentation classique en complément à 2).*

On désire construire un circuit qui donne en sortie $B = b_3b_2b_1b_0 = -2 * A$ *(par exemple, si on a* 2 *en entrée, on veut la valeur binaire* -4 *à la sortie).*

1. *Toutes les valeurs sont-elles autorisées en entrée ? Autrement dit, le circuit donne-t-il toujours une valeur correcte ?*
2. *Donner la table de vérité du circuit pour toutes les valeurs autorisées.*
3. *Donner les expressions logiques des* 4 *bits de sortie en fonction des* 4 *bits d'entrée en* **simplifiant au maximum les expressions** *et en ne tenant pas compte des valeurs interdites (c'est-à-dire que les sorties peuvent être quelconques dans les cas interdits).*
4. *Donner l'expression logique d'un bit supplémentaire en sortie,* e (débordement), *qui indiquerait qu'une valeur interdite est présente sur les entrées.*

Exercice 10

1. *On construit un circuit qui a 2 entrées (*a *et* b*) et 2 sorties (*s *et* r*) et une ligne de commande* F *tel que :*
   - *si* $F = 0$, *le circuit effectue une addition (*$a + b$ *avec une sortie* s *et une retenue* r*);*
   - *si* $F = 1$, *le circuit effectue une soustraction (*$a - b$ *avec une sortie* s *et une retenue* r*).*

   *Donner la table de vérité (*s *et* r *en fonction de* a *et* b*) pour* $F = 0$ *et celle pour* $F = 1$.

   Montrer que s *se calcule facilement avec une porte et* r *avec deux (on autorise NON, OU, ET et XOR). Dessiner le circuit de ce 1/2-additionneur/soustracteur.*
2. *On construit maintenant un additionneur/soustracteur complet, c'est-à-dire un circuit à trois entrées (*a, b *et* r*), deux sorties (*s *et* r'*) et une ligne de commande* F *tel que :*
   - *si* $F = 0$, *le circuit effectue une addition (*$a + b + r$ *avec une sortie* s *et une retenue* r'*);*

- *si* $F = 1$, *le circuit effectue une soustraction* ($a - (b + r)$ *avec une sortie* s *et une retenue* r'*).*

Montrer que s *s'exprime facilement en fonction de* a, b *et* r.
Donner la valeur de r' *(éventuellement en fonction de* a *et* F*) dans les cas suivants :*

- $b = r = 1$;
- $b = r = 0$;
- $b = \overline{r}$.

Donner une expression possible pour calculer r'.

3. *À partir des deux circuits précédents, proposer une construction pour un additionneur/soustracteur sur* n *bits, c'est-à-dire un circuit avec 2 fois* n *bits en entrée et qui effectue l'addition ou la soustraction de ces deux nombres suivant la valeur d'une ligne de commande.*

Exercice 11

1. *On souhaite construire un multiplicateur 2 bits par 1 bit (représentant des nombres entiers positifs), c'est-à-dire un circuit à trois entrées* a_0, a_1 *et* b, *et deux sorties* s_0 *et* s_1 *tel que :*
   - *si* $b = 0$, $s_0 = s_1 = 0$;
   - *si* $b = 1$, $s_0 = a_0$ *et* $s_1 = a_1$.

   Donner le schéma d'un tel circuit en utilisant 2 portes ET. On appelle M ce circuit.
2. *On veut maintenant construire un multiplicateur 2 bits par 2 bits. En décomposant cette multiplication en multiplications plus simples et en additions, montrer que l'on peut le faire avec deux circuits M et 2 demi-additionneurs. Donner le schéma du circuit correspondant.*
3. *Donner le schéma d'un multiplicateur 2 bits par* n *bits utilisant des circuits M, demi-additionneurs et additionneurs complets.*

Exercice 12

1. *Donner la table de vérité d'un demi-additionneur 1 bit. Donner le schéma logique d'un demi-additionneur en utilisant une porte ET et une porte XOR.*
2. *Donner la table de vérité d'un additionneur complet. Donner le schéma logique d'un additionneur complet en utilisant deux demi-additionneurs et une porte OU.*
3. *Donner le schéma d'un additionneur 4 bits.*
4. *On suppose que le passage d'un signal électrique dans une porte « coûte » 10 ns. On dit alors qu'un circuit travaille en* p *ns si tous les signaux en sortie sont disponibles en au maximum* p *ns. En combien de temps un demi-additionneur, un additionneur complet et un additionneur 4 bits travaillent-ils ?*
5. *On va construire un additionneur 4 bits à retenue anticipée, c'est-à-dire un circuit où le calcul des retenues intermédiaires se fait plus rapidement. Soit une fonction de génération de retenue* $G = ab$ *et une fonction de propagation de retenue* $P = a + b$ *; montrer que la retenue de sortie d'un additionneur complet peut se calculer par la formule* $r' = Pr + G$, *où* a *et* b *sont les entrées et* r *la retenue d'entrée.*
6. *Dans notre additionneur 4 bits, on note* $G_i = a_i b_i$ *et* $P_i = a_i + b_i$, *et* r_i *la retenue intermédiaire d'un étage, normalement calculée à partir des entrées* a_i, b_i *et de la retenue précédente* r_{i-1}. *Exprimer* r_0, r_1, r_2 *et* $r_3 = r$ *en fonction de* G_0, G_1, G_2, G_3, P_0, P_1, P_2 *et* P_3.
7. *En supposant que l'on dispose de plusieurs portes OU et ET à 2, 3 et 4 entrées, et que chacune travaille en 10 ns, en combien de temps se fait le calcul des* G_i, P_i *et* r_i *? Est-ce intéressant ? Quelle différence y a-t-il entre le temps de travail d'un additionneur 8 bits normal et d'un additionneur 8 bits à retenue anticipée (avec les bonnes portes OU et ET) ?*

Exercice 13 *On souhaite construire une machine à vote majoritaire sur 3 entrées* a_2, a_1 *et* a_0, *qui a deux sorties* M *et* U *telles que* M *représente le vote majoritaire des entrées (*M *vaut* 0 *s'il y a plus d'entrées à* 0, *et* 1 *s'il y a plus d'entrées à* 1*) et* U *(Unanimité) vaut* 1 *si les trois entrées sont égales,* 0 *sinon.*

1. *Donner la table de vérité de la machine.*
2. *Donner les expressions logiques des* 2 *bits de sortie en fonction des* 3 *bits d'entrée.*
3. *Dessiner la machine en utilisant* exactement *:*
   - *2 portes OU et 2 portes ET pour* M *;*
   - *une porte OU, une porte NON-OU à trois entrées et une porte ET à trois entrées pour* U.
4. *On souhaite étendre la machine à 5 bits d'entrée et avoir un bit de sortie* M' *représentant le vote majoritaire des entrées. On suppose que l'on dispose de la machine majoritaire à 3 entrées ci-dessus donnant un bit* M *de sortie. Donner une expression logique pour* M' *en décomposant en trois cas : soit les trois premiers bits sont à* 1*, soit il y a une majorité de* 1 *dans les trois premiers bits, soit les* 2 *derniers bits sont à* 1.

Exercice 14 *On veut concevoir un circuit permettant de comparer deux nombres* A *et* B *de* 4 *bits (valant de* 0 *à* 15*),* $A = a_3a_2a_1a_0$ *et* $B = b_3b_2b_1b_0$*. Le circuit comporte deux sorties :* G *valant* 1 *si* $A > B$ *et* E *valant* 1 *si* $A = B$.

1. *Soit* a *et* b *deux nombres de* 1 *bit. Soit un circuit à deux sorties :* g *valant* 1 *si* $a > b$ *et* e *valant* 1 *si* $a = b$*. Donner les fonctions logiques* g *et* e*, dessiner le circuit.*
2. *En utilisant des circuits de la question précédente ainsi que des portes logiques OU, ET (à* 2*,* 3 *ou* 4 *entrées) et NON, concevoir un circuit permettant de comparer deux nombres de* 4 *bits. Dessiner le schéma en expliquant votre raisonnement.*

Exercice 15 *On veut concevoir un circuit permettant de comparer deux nombres* **signés** A *et* B *de* 4 *bits (valant de* -8 *à* 7*),* $A = a_3a_2a_1a_0$ *et* $B = b_3b_2b_1b_0$*. Le circuit comporte deux sorties :* G *valant* 1 *si* $A > B$ *et* E *valant* 1 *si* $A = B$.

1. *Soit* a *et* b *deux nombres non-signés de* 1 *bit. Soit un circuit à deux sorties :* g *valant* 1 *si* $a > b$ *et* e *valant* 1 *si* $a = b$*. Donner les fonctions logiques* g *et* e*, dessiner le circuit à l'aide de portes OU, ET, NON et XOR.*
2. *En utilisant des circuits de la question précédente ainsi que des portes logiques OU, ET (à* 2*,* 3 *ou* 4 *entrées) et NON, concevoir un circuit permettant de comparer deux nombres signés de* 4 *bits. Dessiner le schéma en expliquant votre raisonnement.*

Exercice 16 *On veut concevoir par récurrence un circuit permettant de comparer deux nombres non-signés* A *et* B *de* n *bits,* $A = a_{n-1} \dots a_0$ *et* $B = b_{n-1} \dots b_0$*. Le circuit comporte trois sorties :* G *valant* 1 *si* $A > B$*,* E *valant* 1 *si* $A = B$ *et* P *valant* 1 *si* $A < B$.

1. *Soit* a *et* b *deux nombres de* 1 *bit. Soit un circuit à trois sorties :* g *valant* 1 *si* $a > b$*,* e *valant* 1 *si* $a = b$ *et* p *valant* 1 *si* $a < b$*. Donner les fonctions logiques* g*,* e *et* p*, dessiner le circuit à l'aide de portes OU, ET, NON et XOR (celles que vous voulez).*
2. *En supposant que l'on dispose de portes logiques OU, ET et NON, d'un circuit logique comparant deux nombres non-signés de* $n - 1$ *bits (avec trois sorties G, E et P), ainsi que du circuit de la question précédente, concevoir un circuit permettant de comparer deux nombres non-signés de* n *bits en séparant le travail en deux : comparaison des deux bits de poids fort et comparaison du reste. Dessiner le schéma en expliquant votre raisonnement.*
3. *Toujours à l'aide des circuits précédents, concevoir un circuit permettant de comparer deux nombres* **signés** *de* n *bits.*

Exercice 17 *On désire construire un comparateur arithmétique binaire de nombres de n bits. C'est-à-dire un circuit à 2 fois* n *entrées* $X = x_{n-1} \dots x_0$ *et* $Y = y_{n-1} \dots y_0$ *représentant deux nombres binaires non-signés sur* n *bits et 2 sorties,* Sup *et* Inf*. On veut que* $\text{Sup} = (X > Y)$ *et* $\text{Inf} = (X < Y)$ *(autrement dit,* Sup *(resp.* Inf*) vaut* 1 *si* $X > Y$ *(resp.* $X < Y$*) et* 0 *sinon).*

On effectue une comparaison bit par bit en commençant par ceux de poids fort.

On pose

$$\text{Sup}_i = (x_{n-1}x_{n-2} \dots x_i > y_{n-1}y_{n-2} \dots y_i)$$

$$\text{Inf}_i = (x_{n-1}x_{n-2}\dots x_i < y_{n-1}y_{n-2}\dots y_i)$$

C'est-a-dire est-ce que la valeur binaire formée par les premiers bits de poids fort de X est plus grande (resp. plus petite) que la valeur binaire formée par les premiers bits de poids fort de Y.

1. *Trouver les formules de récurrence*

$$\text{Sup}_i = f(\text{Sup}_{i+1}, \text{Inf}_{i+1}, x_i, y_i)$$

$$\text{Inf}_i = g(\text{Sup}_{i+1}, \text{Inf}_{i+1}, x_i, y_i)$$

   C'est-à-dire exprimer Sup_i *et* Inf_i *en fonction de* Sup_{i+1}, Inf_{i+1}, x_i *et* y_i.
2. *Donner le schéma d'un comparateur 1 bit. Donner le schéma d'un comparateur* n *bits à partir de comparateur(s)* $n-1$ *bits, de comparateur(s) 1 bit, de porte(s) NON, ET et OU à 2 entrées.*
3. *On suppose que le passage d'un signal électrique dans une porte « coûte » 10 ns. On dit alors qu'un circuit travaille en* p *ns si tous les signaux en sortie sont disponibles en au maximum* p *ns. En combien de temps un comparateur 1 bit et un comparateur* n *bits travaillent-ils ?*

5 Exercices Assembleur

Exercice 18 *Dans* `r0` *se trouve l'adresse d'une chaîne de caractères terminée par l'octet nul. Écrire une procédure assembleur qui, lorsqu'elle se termine, permet de récupérer le nombre de caractères de la chaîne dans* `r1`.

Exercice 19

1. *Un tableau d'entiers est stocké en mémoire. Chaque élément a une valeur de 1 à 127 et est donc stocké sur un octet. L'octet qui suit le dernier élément du tableau est nul. L'adresse du premier élément du tableau se trouve dans le registre* `r0`. *Écrire une procédure assembleur qui, lorsqu'elle se termine, permet de récupérer le plus grand élément du tableau dans le registre* `r1`.
2. *Rajouter au programme précédent le calcul de la somme de tous les éléments du tableau dans le registre* `r2`.

Exercice 20 *Un tableau d'entiers est stocké en mémoire, chacun étant stocké sur un octet (et représente un nombre en complément à 2) et aucun n'est nul. L'octet qui suit le dernier élément du tableau est nul. L'adresse du premier élément du tableau se trouve dans le registre* `r0`.

Écrire une procédure assembleur qui, lorsqu'elle se termine, permet de récupérer le nombre d'entiers positifs du tableau dans le registre `r1` *et le nombre d'entiers du tableau égaux à 40 (décimal) dans le registre* `r2`.

Exercice 21 *Une chaîne de caractères est stockée en mémoire. Chaque élément est stocké sur un octet et aucun n'est nul. L'octet qui suit le dernier élément de la chaîne est nul. L'adresse du premier élément de la chaîne se trouve dans le registre* `r0`. *On souhaite effectuer une opération de remplacement sur certains caractères de la chaîne. Plus précisément, le registre* `r1` *contient le caractère dont on souhaite remplacer chaque occurence dans la chaîne par le caractère se trouvant dans le registre* `r2`.

Écrire une procédure assembleur qui, lorsqu'elle se termine, assure que toutes les occurences (dans la chaîne) du caractère contenu dans `r1` *sont remplacées par le caractère contenu dans* `r2`. *On souhaite également avoir à la fin dans le registre* `r3` *le nombre de remplacements effectués.*

Exercice 22 *Dans les 16 bits de poids faible de* `r1` *se trouve un nombre entier non-signé ; les 16 autres bits de* `r1` *sont nuls. Dans les 16 bits de poids faible de* `r2` *se trouve un nombre entier non-signé ; les 16 autres bits de* `r2` *sont nuls. Écrire de deux manières différentes une procédure assembleur qui, lorsqu'elle se termine, permet de récupérer dans les 32 bits de* `r0` *le produit des deux nombres (sans bien sûr utiliser directement une instruction de multiplication).*

1. *La suite d'instructions est une simple boucle effectuant autant d'additions que nécessaire.*
2. *La suite d'instructions implémente la procédure classique de multiplication comme suite d'additions des produits partiels.*

Exercice 23 *Une chaîne de caractères est stockée en mémoire. Chaque caractère est stocké sur un octet et l'octet qui suit le dernier caractère de la chaîne est nul. L'adresse du premier élément de la chaîne se trouve dans le registre* `r0`.

On souhaite renverser la chaîne, c'est-à-dire l'écrire en commençant par le dernier caractère de la chaîne et stocker cette nouvelle chaîne en mémoire à une adresse contenue dans le registre `r1`. *(Ainsi, si la première chaîne est* `truc`, *on souhaite écrire la chaîne* `curt` *en mémoire avec le premier élément à l'adresse qui se trouve dans* `r1` *et avec l'octet nul final.)*

Écrire une procédure assembleur qui renverse la chaîne pointée par `r0` *et la stocke à l'adresse pointée par* `r1`.

Exercice 24 *Une chaîne de caractères est stockée en mémoire. Chacun est stocké sur un octet et aucun n'est nul ; l'octet qui suit le dernier élément de la chaîne est nul. L'adresse du premier élément de la chaîne se trouve dans le registre* `r0`. *On souhaite générer des acronymes dans cette chaîne, c'est-à-dire mettre un* `'.'` *après chaque lettre, par exemple en transformant* `IBM` *en* `I.B.M.` *Cette chaîne modifiée se mettra à l'adresse contenue dans* `r1`.

Pour indiquer que la transformation est souhaitée, on place un caractère `#`, *pour indiquer qu'elle n'est plus nécessaire une autre* `#` *doit apparaître. Les* `#` *disparaissent de la transformation. On considérera qu'il n'y a pas de*

blancs ni de caractères bizarroïdes entre les `#`*. Par exemple* `"je suis chez #IBM# pour toujours"` *deviendra* `"je suis chez I.B.M. pour toujours"`*; la chaîne peut commencer par le caractère* `#`*:* `"#RATP#, #SNCF# : même combat"` *deviendra* `"R.A.T.P., S.N.C.F. : même combat"`*. Il peut y avoir aucun ou plusieurs* `#` *et si c'est le dernier caractère de la chaîne, il pourra ne pas être présent.*

Écrire une procédure assembleur qui, lorsqu'elle se termine, laisse à l'adresse qui était pointée par `r1` *(qui peut avoir changé) la nouvelle chaîne modifiée.*

Exercice 25 *Une chaîne de caractères est stockée en mémoire. Elle a moins de 128 éléments, chacun est stocké sur un octet et aucun n'est nul ; elle est composée de caractères* `a`*, de caractères* `b` *et d'autres caractères. L'octet qui suit le dernier élément de la chaîne est nul. L'adresse du premier élément de la chaîne se trouve dans le registre* `r0`*. On souhaite comparer le nombre de caractères* `a` *et le nombre de caractères* `b` *de la chaîne. Plus précisément, on souhaite avoir dans* `r1`*, à la fin de la procédure,* `0` *s'il y a plus de* `a` *que de* `b` *et* `1` *s'il y a autant ou plus de* `b` *que de* `a`*.*

Écrire une procédure assembleur qui, lorsqu'elle se termine, assure que le registre `r1` *contient* `0` *s'il y a plus de* `a` *que de* `b` *et* `1` *sinon.*

Exercice 26 *On désire faire l'addition de deux nombres entiers positifs stockés sous la forme de deux chaînes de caractères ASCII. Chaque caractère d'une chaîne représente un chiffre de* `0` *à* `9` *stocké sur un octet par son code ASCII de valeur décimale* 48 *(pour* `0`*) à* 57 *(pour* `9`*). Les deux chaînes comportent le même nombre de caractères et se terminent chacune par un octet nul. L'adresse du premier caractère de la première chaîne se trouve dans le registre* `r0`*. L'adresse du premier caractère de la deuxième chaîne se trouve dans le registre* `r1`*.*

On désire stocker l'addition des deux nombres sous la forme d'une chaîne de caractères ASCII en mémoire dont le premier caractère sera mis à l'adresse contenue dans le registre `r2` *et qui sera terminée par un octet nul. Les deux chaînes sont stockées à l'envers de leur affichage : le premier caractère représente le chiffre le moins significatif et le dernier le plus significatif. On peut donc effectuer l'addition en parcourant les chaînes du premier au dernier caractère.*

Écrire une procédure assembleur qui, lorsqu'elle se termine, laisse en mémoire à l'adresse initialement pointée par `r2` *la chaîne de caractères ASCII correspondant à la somme des deux nombres initiaux.*

Exercice 27 *Une chaîne de caractères est stockée en mémoire. Chaque caractère est stocké sur un octet et est soit un caractère '*`x`*' soit un caractère '*`y`*'. L'octet qui suit le dernier caractère de la chaîne est nul. L'adresse du premier élément de la chaîne se trouve dans le registre* `r0`*.*

On souhaite comparer le nombre de groupes de caractères `x` *et le nombre de groupes de caractères* `y` *de la chaîne (un groupe de caractères* `x` *[resp.* `y`*] est simplement un ou plusieurs caractères* `x` *[resp.* `y`*] qui se suivent). Plus précisément, on souhaite avoir dans* `r1`*, à la fin de la procédure,* `0` *s'il y a plus de groupes de* `x` *que de groupes de* `y` *et* `1` *s'il y a autant ou plus de groupes de* `y` *que de groupes de* `x`*.*

Écrire une procédure assembleur qui, lorsqu'elle se termine, assure que le registre `r1` *contient* `0` *s'il y a strictement plus de groupes de* `x` *que de groupes de* `y` *et* `1` *sinon.*

6 Exercices mémoire cache

Exercice 28 *Un programme se compose d'une boucle de 36 instructions à exécuter 3 fois ; les instructions se trouvant, dans l'ordre, aux adresses mémoire 1 à 8, 77 à 84, 9 à 16, 77 à 84, 17 à 20. Ce programme doit tourner sur une machine possédant un cache d'une taille de 16 instructions. Le temps de cycle de la mémoire principale est M et le temps de cycle du cache est C. Le cache est associatif (un bloc mémoire peut venir dans n'importe quel bloc du cache) et la stratégie de remplacement utilisée est LRU (on remplace le bloc le moins récemment utilisé).*

1. *Le cache possède 4 blocs de 4 instructions : les blocs que l'on peut transférer sont 1-4, 5-8, ..., 73-76, 77-80, 81-84... Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ? Le cache est vide au départ.*
2. *Le cache possède 2 blocs de 8 instructions : les blocs que l'on peut transférer sont 1-8, 9-16, ..., 73-80, 81-88... Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ? Le cache est vide au départ.*
3. *Refaire les deux questions précédentes en supposant que les instructions sont aux adresses mémoire 1 à 8, 75 à 82, 9 à 16, 75 à 82, 17 à 20. Le cache est vide au départ.*

Rappel : Lorsque l'on va chercher un mot en mémoire, pendant M on ramène le mot pour le processeur et tout le bloc correspondant dans le cache.

Exercice 29 *Un programme se compose d'une boucle de 20 instructions à exécuter 4 fois ; cette boucle se trouve aux adresses mémoire 1 à 20. Ce programme doit tourner sur une machine possédant un cache d'une taille de 16 instructions. Le temps de cycle de la mémoire principale est M et le temps de cycle du cache est C.*

1. *Le cache est à correspondance directe, formé de 8 blocs de 2 instructions. On rappelle que cela veut dire que les mots mémoire 1 et 2, 17 et 18... vont dans le premier bloc, que les mots 3 et 4, 19 et 20... vont dans le deuxième, etc. Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ?*
2. *Le cache est maintenant associatif (un bloc mémoire peut venir dans n'importe quel bloc du cache) et la stratégie de remplacement utilisée est LRU (on remplace le bloc le moins récemment utilisé). Quel est le temps d'exécution du programme ?*
3. *Refaire les deux premières questions en prenant un cache composé de 4 blocs de 4 mots.*

Rappel : Lorsque l'on va chercher un mot en mémoire, pendant M on ramène le mot pour le processeur et tout le bloc correspondant dans le cache.

Exercice 30 *Un programme se compose d'une boucle de 16 instructions consécutives à exécuter 5 fois. Ce programme doit tourner sur une machine possédant un cache d'une taille de 16 instructions. Le temps de cycle de la mémoire principale est M et le temps de cycle du cache est C. Le cache est associatif (un bloc mémoire peut venir dans n'importe quel bloc du cache) et la stratégie de remplacement utilisée est LRU (on remplace le bloc le moins récemment utilisé).*

1. *Le cache possède 4 blocs de 4 instructions : les blocs mémoire que l'on peut transférer dans le cache sont 1-4, 5-8, 9-12, 13-16, 17-20...*
   (a) *Le programme se place aux adresses 1 à 16 en mémoire. Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ?*
   (b) *Le programme se place aux adresses 3 à 18 en mémoire. Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ? A-t-on le même résultat si le programme se place aux adresses 2 à 17 ou 4 à 19 ?*
2. *Le cache possède 1 bloc de 16 instructions : les blocs mémoire que l'on peut transférer dans le cache sont 1-16, 17-32... Reprendre les questions ci-dessus.*

3. *Sachant que l'adresse de placement du programme est tirée aléatoirement, pouvez-vous calculer l'efficacité du cache ?*

Rappel : Lorsque l'on va chercher un mot en mémoire, pendant M on ramène le mot pour le processeur et tout le bloc correspondant dans le cache.

Exercice 31 *Notre machine possède un cache d'une taille de 16 instructions regroupées en 4 blocs de 4 instructions. Le cache est à correspondance directe. On rappelle que cela veut dire que les mots mémoire 1-4, 17-20... vont dans le premier bloc du cache, que les mots 5-8, 21-24... vont dans le deuxième, etc. Le temps de cycle de la mémoire principale est M et le temps de cycle du cache est C.*

1. *Un programme se compose d'une boucle de 40 instructions à exécuter 3 fois ; les instructions se trouvant, dans l'ordre, aux adresses mémoire 1 à 20, 21 à 24, 1 à 4, 21 à 24, 1 à 4, 21 à 24. Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ? Le cache est vide au départ.*
2. *Le compilateur a réussi à optimiser votre programme et en a réduit la taille en gagnant quelques intructions. Il se compose d'une boucle de 36 instructions à exécuter 3 fois ; les instructions se trouvant, dans l'ordre, aux adresses mémoire 1 à 16, 17 à 20, 1 à 4, 17 à 20, 1 à 4, 17 à 20. Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ? Le cache est vide au départ.*
3. *Que concluez-vous des deux précédents résultats ?*

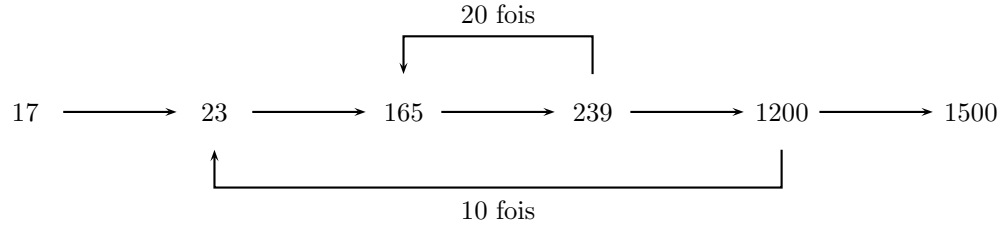
Rappel : Lorsque l'on va chercher un mot en mémoire, pendant M on ramène le mot pour le processeur et tout le bloc correspondant dans le cache.

Exercice 32 *Notre machine possède un cache d'une taille de 24 instructions regroupées en 6 blocs de 4 instructions. Le cache est associatif (un bloc mémoire peut venir dans n'importe quel bloc du cache) et la stratégie de remplacement utilisée est LRU (on remplace le bloc le moins récemment utilisé). Le temps de cycle de la mémoire principale est M et le temps de cycle du cache est C.*

1. *Un programme se compose d'une boucle de 28 instructions à exécuter 5 fois ; les instructions se trouvant, dans l'ordre, aux adresses mémoire 1 à 28. Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ? Le cache est vide au départ.*
2. *Vous avez réorganisé votre code et votre boucle est maintenant découpée en deux : on exécute d'abord 5 fois les instructions 1 à 14* puis *5 fois les instructions 15 à 28. Quel est le temps total d'exécution du programme en ne tenant pas compte des temps de calcul ? Le cache est vide au départ.*
3. *Que concluez-vous des deux précédents résultats ?*
4. *Refaire uniquement la première question en supposant que le cache est à correspondance directe. On rappelle que cela veut dire que les mots mémoire 1-4, 25-28... vont dans le premier bloc du cache, que les mots 5-8, 29-32... vont dans le deuxième, etc.*

Rappel : Lorsque l'on va chercher un mot en mémoire, pendant M on ramène le mot pour le processeur et tout le bloc correspondant dans le cache.

Exercice 33 *Un programme se compose de deux boucles* `for` *imbriquées : une petite boucle interne et une plus grande boucle externe. La structure générale du programme est donnée dans la figure suivante.*



Les adresses décimales données déterminent l'emplacement des deux boucles ainsi que le début et la fin du programme. Tous les emplacements mémoire des différents blocs, 17-22, 23-164, 165-239, etc., contiennent des instructions devant être exécutées séquentiellement. Le programme doit s'exécuter sur une machine possédant un cache. Le cache est organisé sous la forme de correspondance directe et les paramètres en sont :

- *taille de la mémoire principale : 64K mots ;*
- *taille du cache : 1K (soit 1 024) mots ;*
- *taille d'un bloc : 128 mots.*

Le temps de cycle de la mémoire principale est 100 ns et le temps de cycle du cache est 10 ns. Une instruction occupe un mot en mémoire.

En ne tenant pas compte des temps de recherche et de rangement d'un opérande ou d'un résultat associés aux opérations de lecture et d'écriture, calculer le temps total nécessaire à la recherche des instructions dans le programme. Estimer le facteur d'amélioration résultant de l'utilisation du cache.