Examen
Programmation des Systèmes

Exercice 1: (1 pt) Un exercice du cours consiste à coder un itérateur en assembleur en implémentant les méthodes définies dans le fichier interface *moncontext.h* ci-dessous,

```
#ifndef MONCONTEXT_H
#define MONCONTEXT_H

struct moncontext_t;

void mkctx_1(void (*)(void),int);
void mkctx_2(void (*)(void),int);
void swap21();
void swap12();

#endif
```

1. Décrivez comment le fichier *moncontext.h* est utilisé par le programmeur C qui utilise les fonctions ainsi que la fonction des directives du préprocesseur # ifndef, # define.
2. Donnez le pseudo-code des fonctions.
3. Utilisez les fonctions pour transformer le code ci-dessous en un itérateur. Et donnez le code pour afficher les trois premiers éléments (dans la routine *main*()).

```
void affiche(struct noeud *noeud)
{
        if (noeud != NULL)
        {
                affiche(noeud->droit);
                printf(" \% d ",noeud->val);
                affiche(noeud->gauche);
        }
}
```

Exercice 2: (0.5 pt) Répondez aux questions

1. Quelle est la différence entre allocation *statique* et *dynamique* des variables?

2. Décrivez comment déclarer des variables statiques et dynamiques en assembleur, donnez des extraits de code assembleur.

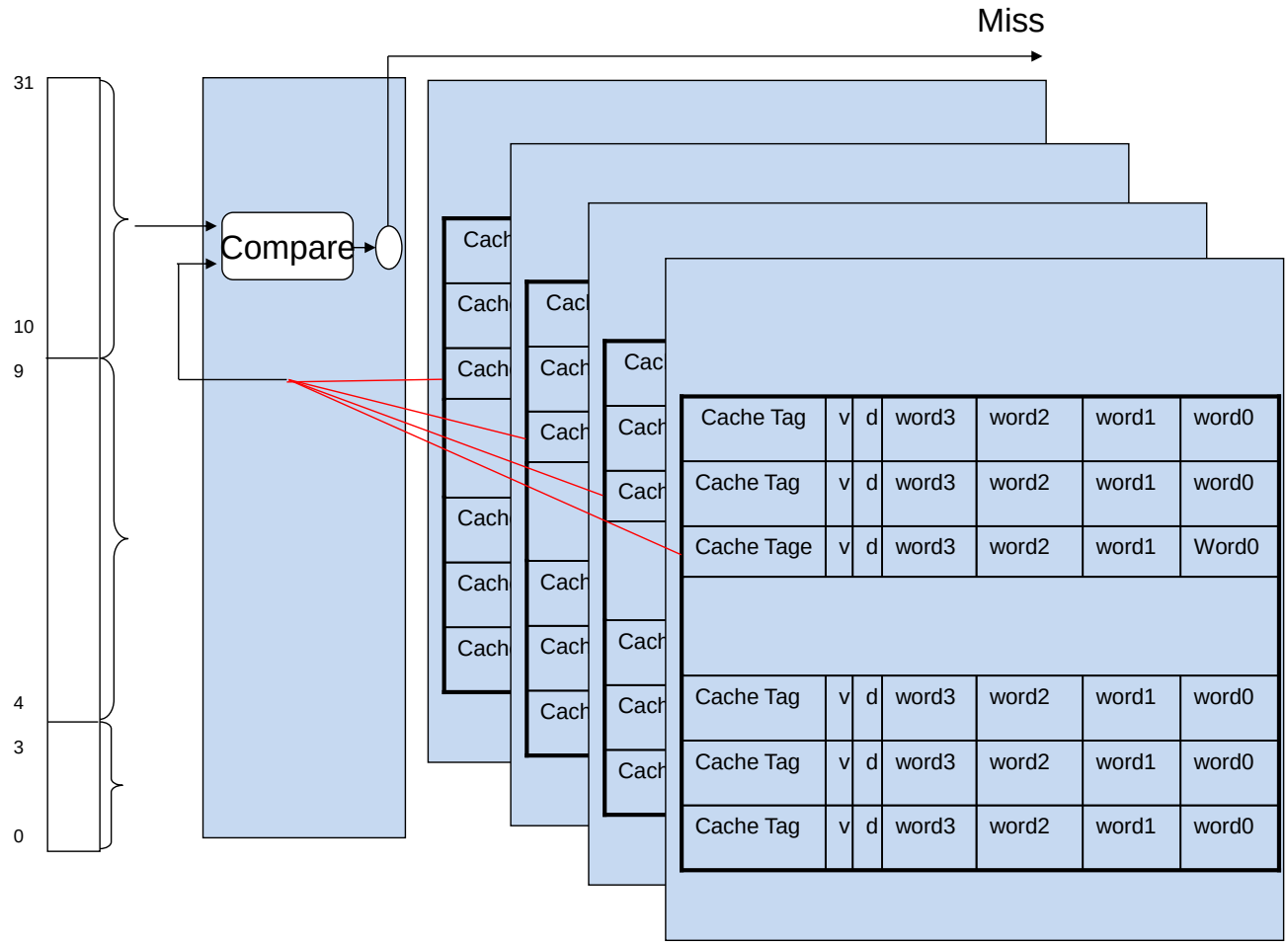
3. Quelle est la différence d'utilisation entre le tas et la pile?

Exercice 3: (1 pt) On considère la mémoire cache représentée sur la figure ci-dessous et constituée de 4 structures identiques.

1. Décrivez les entrés *Cache Tag*, *v*, *d*.

2. Quelle est la capacité de la mémoire cache? (nombre de bytes maximum stockables).

3. Donnez les pseudo-code d'accès en lecture et écriture qui utilisent cette mémoire cache (stratégies read-allocate et writeback).



Exercice 4: (0.5 pt) Décrivez l'exécution du programme ci-dessous

```
mov     r0 , pc
mov     r0 , r0
mov     pc , r0
```

Exercice 5:(1 pt) Ecrivez un programme en assembleur pour gérer une file d'attente FIFO implémentée avec une liste chaînée. Les éléments à mémoriser sont des *int*, utilisez une structure du type *elements*.

```
struct elements
{
   struct elements *suivant;
   int valeur;
}
```

Il faut écrire l'équivalent des deux fonctions *push* et *pop* dont les en-têtes en C sont

```
void push(int valeur)
int pop()
```

Les fonctions doivent respecter les conventions ARM (AAPCS, Arm Architecture Procedure Call Standard).

Exercice 6:(1 pt) Ecrivez en assembleur l'équivalent de la procédure *mult()* ci-dessous

```
unsigned long long mult(unsigned long a, unsigned long b){
    unsigned long long c;
    c = a*b;
    return c;
}
```

La fonction doit respecter les conventions ARM (AAPCS, Arm Architecture Procedure Call Standard).

Exercice 7:(1 pt) Aprés l'exécution de l'instruction *cmp r0,r1*,

1. Indiquez comment sont positionnés les bits du *cpsr*.
2. Montrez que la condition $N{==}V$ est équvalente à $r0 \geq r1$.
3. Quelle est la différence entre les conditions *HS* Higher or Same (C==1) et *GE* Greater than or Equal (N==V)?