.:: Architecture ::.

semaine n°4
Exercices

Le MCS251-Passage d'arguments

- Exercice

Fonction : void somtab(int * tab, int n); => Somme d'un tableau

1/2 – Algo
3 – Paramètres
4 – Affectation des variables locales
5 - Traduction

Le MCS251-Passage d'arguments

- Exercice

Fonction : int somtab(int * tab, int n); => Somme d'un tableau

1/2 – Algo :

```
int i = 0
int som = 0
while ( i < n ) {
    v = tab[i]
    som = som +v;
    i++
}
retourner som;
```

Le MCS251-Passage d'arguments

- Exercice

Fonction : int somtab(int * tab, int n); => Inverse le tableau

3 – Arguments : WR0 WR2

Retour de la fonction : int <=> WR10

4 – Variables locales :

```
int i = 0                    WR12
int som = 0                  WR14
while ( i < n ) {
      v = tab[i]             WR16
      som = som + v
      i++;
}
return som;
```

Le MCS251-Passage d'arguments

```
int somTab( int * tab, int n) {
                  WR0     WR2


        int i = 0                 WR12
        int som = 0               WR14
        while ( i < n ) {

              v = tab[i]




              som = som + v
              i++;
        }
        return som;
}
```

```
_somTab:
        PUSH    WR12
        PUSH    WR14
        PUSH    WR16
        MOV     WR12,#0
        MOV     WR14,#0
wh1:    CMP     WR12, WR2
        JGE ewh
            PUSH    WR12
            ADD     WR12,WR12
            ADD     WR12,WR0
            MOV     WR16,@WR12
            POP     WR12
            ADD     WR14,WR16
            INC     WR12
            JMP     wh1
ewh:    MOV     WR10,WR14
        POP     WR16
        POP     WR14
        POP     WR12
        RET
```

Inverser un tableau de long

```
void funcInv(long * tab, int nb) {
    long * debut = tab;
    long * fin = tab+nb-1;
    long t0,t1;
    while ( debut < fin ) {
        t0 = *fin;
        t1 = *debut;
        *fin = t1;
        *debut = t0;
        fin --;
        debut ++;
    }
}
```

Inverser un tableau de long

```
void funcInv(long * tab, int nb) {
    long * debut = tab;
    long * fin = tab+nb-1;
    long t0,t1;
    while ( debut < fin ) {
        t0 = *fin;
        t1 = *debut;
        *fin = t1;
        *debut = t0;
        fin --;
        debut ++;
    }
}
```

tab => WR0, nb => WR2
debut => WR12
fin => WR14
t0 => DR16 td1 => DR20

Inverser un tableau de long

```
void funcInv(long * tab, int nb) {

    long * debut = tab;
    long * fin = tab+nb-1;
```

```
funcInv :   PUSH    WR12
            PUSH    WR14
            PUSH    WR16
            PUSH    WR18
            PUSH    WR20
            PUSH    WR22
            MOV     WR12,WR0
            MOV     WR14,WR2
            DEC       WR14
            ADD     WR14, WR14
            ADD     WR14, WR14
            ADD     WR14,WR0
```

Inverser un tableau de long

```
        while ( debut < fin ) {

                t0 = *fin;

                t1 = *debut;

                *fin = t1;

                *debut = t0;

                fin --;
                debut ++;
        }
}
```

```
WH1:    CMP     WR12,WR14
        JGE     EWH1
        MOV     WR16,@WR14
        MOV     WR18,@WR14+2
        MOV     WR20,@WR12
        MOV     WR22,@WR12+2
        MOV     @WR14,WR20
        MOV     @WR14+2,WR22
        MOV     @WR12,WR16
        MOV     @WR12+2,WR18
        DEC     WR14,#4
        INC     WR12,#4
        JMP     WH1
EWH1:   POP     WR22
        POP     WR20
        POP     WR18
        POP     WR16
        POP     WR14
        POP     WR12
        RET
```

Fonction récursive : factorielle

```
int factorielle(int v) {
        int r,f;
        if ( v <= 1 ) return 1;
        f = factorielle(v-1);
        r = v * f;
        return r;
}
```

1 – affectation des paramètres et variables
2 – réécriture
3 – traduction

Fonction récursive : factorielle

Affectation des vairables :

```
int factorielle(int v) {
WR10                WR0
      int r,f;          WR10, WR12
      if ( v <= 1 ) return 1;
      f = factorielle(v-1);
      WR10          WR0
      r = v * f;
      return r;
}
```

Fonction récursive : factorielle

Réécriture :

```
int factorielle(WR0) {
 WR10                  WR0
      int r,f;         WR10, WR12
      if ( WR0 <= 1 ){
            WR10 = 1 ;
             return;
      }
      sauver WR0
      WR0 = WR0-1
      WR10 = factorielle(WR0);
      WR12 = WR10
      restituer WR0
      WR10 = (int)WR0 * WR12;
      return;
}
```

Fonction récursive : factorielle

Traduire :

```
int factorielle(WR0) {
        int r,f;            WR10, WR12
        if ( WR0 <= 1 ){

                WR10 = 1 ;

                return;
        }
        sauver WR0
        WR0 = WR0-1
        WR10 = factorielle(WR0);
        WR12 = WR10
        restituer WR0
        WR10 = (int)WR0 * WR12;


        return;
}
```

```
factorielle:
                PUSH    WR12
if0:            CMP     WR0,#1
                JG      eif0
then0:            MOV   WR10,#1
                  POP   WR12
                  RET
eif0:
                PUSH    WR0
                DEC           WR0
                CALL    factorielle
                MOV     WR12,WR10
                POP     WR0
                MOV     WR10,WR12
                MUL     WR10,WR0
                POP     WR12
                RET
```

TP – Fonction Trie

Fonction trie

Traduire :

```
void trie(int * tab, int len) {
   int iv;
   int i=0;
   if ( len > 1 ) {
        while ( i <  len -1) {
              iv = min(tab,i,len);
              permute(tab,i,iv);
              i++;
        }
   }
}

void permute(int *tab, int is, int id) {
        int v0 = tab[is];
        int v1=tab[id];
        tab[is]=v1;
        tab[id]=v0;
}
```

```
int min(int * tab, int i, int len) {
        int imin=i;
        int min=tab[i];
        int k = i+1;
        while ( k < len ) {
              int v = tab[k];
              if ( v < min ) {
                    imin = k;
                    min = v;
              }
              k++;
        }
        return imin;
        }
```

TP – Fonction Trie

```
void trie(int * tab, int len) {          _trie:      PUSH    WR14
            WR0     WR2
  int iv;           WR10
 int i=0;                 WR14                       MOV     WR14,#0
  if ( len > 1 ) {                        if1:       CMP     WR2,#1
                                                     JGE     eif1
      while ( i <  len -1) {              wh1:       MOV     WR4,WR2
                                                     DEC     WR4
                                                     CMP     WR14,WR4
                                                     JGE     ewh1
            iv = min(tab,i,len);                     PUSH    WR2
                                                     MOV     WR4,WR2
                                                     MOV     WR2,WR14
                                                     CALL    min
                                                     POP     WR2
            permute(tab,i,iv);                       PUSH    WR2
                                                     MOV     WR2,WR14
                                                     MOV     WR4,WR10
                                                     CALL    permute
                                                     POP     WR2
            i++;                                     INC     WR14
      }                                              JMP     wh1
  }                                       ewh1:
}                                         eif1:      POP     WR14
                                                     RET
```

TP – Fonction Trie

```
int min(int * tab, int i, int len) {
WR10      WR0   WR2  WR4
      int imin=i;  WR10
      int min=tab[i];  WR12




      int k = i+1;  WR14

      while ( k < len ) {

            int v = tab[k];  WR16




            if ( v < min ) {

                  imin = k;
                  min = v;
            }
            k++;
      }
      return imin;

}
```

```
min:    PUSH    DR12
        PUSH    WR16
        MOV     WR10,WR2
        PUSH    WR2
        ADD     WR2,WR2
        ADD     WR2,WR0
        MOV     WR12,@WR2
        POP     WR2
        MOV     WR14,WR2
        INC     WR14
wh2:    CMP     WR14,WR4
        JGE     ewh2
        PUSH    WR14
        ADD     WR14,WR14
        ADD     WR14,WR0
        MOV     WR16,@WR14
        POP     WR14
if2:    CMP     WR16,WR12
        JSGE    eif2
th2:    MOV     WR10,WR14
        MOV     WR12,WR16
eif2:
        INC     WR14
        JMP     wh2
ewh2:   POP     WR16
        POP     DR12
        RET
```

TP – Fonction Trie

```
void permute(int *tab, int is, int id) {
                        WR0, WR2, WR4
    int v = tab[is];              WR12

    int v1=tab[id];               WR14

    tab[is]=v1;

    tab[id]=v0;

}
```

```
permute:    PUSH    DR12

            PUSH    WR2
            ADD     WR2,WR2
            ADD     WR2,WR0
            MOV     WR12,@WR2
            POP     WR2
            PUSH    WR4
            ADD     WR4,WR4
            ADD     WR4,WR0
            MOV     WR14,@WR4
            POP     WR4
            PUSH    WR2
            ADD     WR2,WR2
            ADD     WR2,WR0
            MOV     @WR2,WR14
            POP     WR2
            PUSH    WR4
            ADD     WR4,WR4
            ADD     WR4,WR0
            MOV     @WR4,WR12
            POP     WR4

            POP     DR12
            RET
```