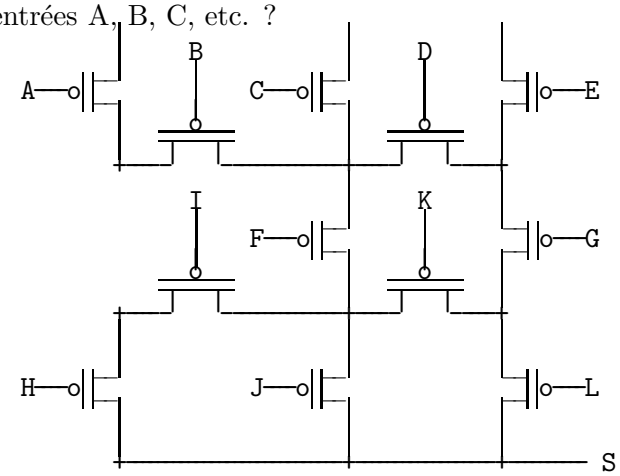
Université de Nanterre vendredi 13 mai 2016
architecture des ordinateurs contrôle

Que valent CF, OF, ZF et SF après les opérations 4+4, 3+3, 8+8, 13+13, 9+7, 8+7, 2-1, 1-2, 8-9, 9-8, 3-8, 8-3, 3-12, 11-3 et 8-8. sur des nombres codés sur 4 bits.

Compléter le bas du schéma avec autant de transistors. Que vaut la sortie S fonction des entrées A, B, C, etc. ?




```
f:
   loadimm16 r3,1
   mov  r1,r2
   mov  r1,r7
   load r1,r4
debut:
   load r2,r5
   sub  r4,r5,r6
   cmovg r5,r4
   cmovg r2,r1
   add r2,r3,r2
   sub r0,r3,r0
   jne debut
   sub r1,r7,r0
   ret
```

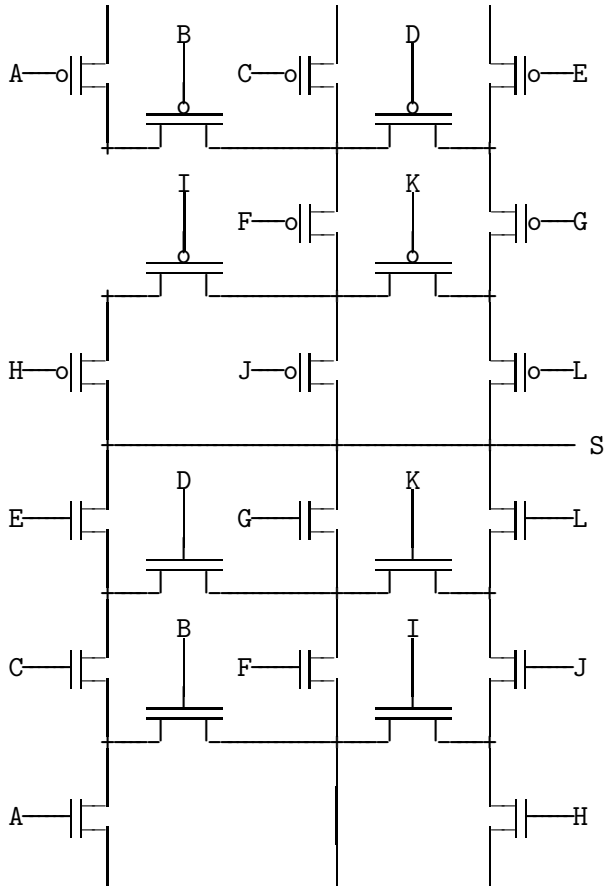
Donner un équivalent simple en C de la fonction `f`. Que fait-elle ?
Que ferait-elle si on remplaçait les 2 `cmovg` par des `cmovl` ?
Que ferait-elle si on les remplaçait par des `cmova` ?
Que ferait-elle si on les remplaçait par des `cmovge` ?

Ecrire en C puis en assembleur la fonction `int g(int n, int *t);` qui rend $\sum_{i=0}^{n-1} t[i]^4 - t[i]^8$.

Corrigé

non signé	signé	CF	OF	ZF	SF
4+4=8	4+4=-8	0	1	0	1
3+3=6	3+3=6	0	0	0	0
8+8=0	-8+-8=0	1	1	1	0
13+13=10	-3+-3=-6	1	0	0	1
9+7=0	-7+7=0	1	0	1	0
8+7=15	-8+7=-1	0	0	0	1
2-1=1	2-1=1	0	0	0	0
1-2=15	1-2=-1	1	0	0	1
8-9=15	-8–7=-1	1	0	0	1
9-8=1	-7–8=1	0	0	0	0
3-8=11	3–8=-5	1	1	0	1
8-3=5	-8-3=5	0	1	0	0
3-12=7	3–4=7	1	0	0	0
11-3=8	-5-3=-8	0	0	0	1
8-8=0	-8–8=0	0	0	1	0

$$S = (\bar{\mathtt{A}} \wedge \bar{\mathtt{B}} \vee \bar{\mathtt{C}} \vee \bar{\mathtt{D}} \wedge \bar{\mathtt{E}}) \wedge \bar{\mathtt{F}} \wedge (\bar{\mathtt{H}} \wedge \bar{\mathtt{I}} \vee \bar{\mathtt{J}} \vee \bar{\mathtt{K}} \wedge \bar{\mathtt{L}}) \vee (\bar{\mathtt{E}} \vee \bar{\mathtt{D}} \wedge (\bar{\mathtt{C}} \vee \bar{\mathtt{B}} \wedge \bar{\mathtt{A}})) \wedge \bar{\mathtt{G}} \wedge (\bar{\mathtt{L}} \vee \bar{\mathtt{K}} \wedge (\bar{\mathtt{J}} \vee \bar{\mathtt{I}} \wedge \bar{\mathtt{H}}))$$



```
f:              //          r0     r1     r2    r3    r4     r5   r6  r7
   loadimm16 r3,1// int f(int n, int*t)//  u      1  x=*t   y=*u  x-y   T
   mov  r1,r2    // { int *u=t,
   mov  r1,r7    //       *T=t,
   load r1,r4    //       x=*t;
debut:           //   do
   load r2,r5    //   { int y=*u;
   sub  r4,r5,r6 //     if(x>y)
   cmovg r5,r4   //       x=y,
   cmovg r2,r1   //       t=u;
   add r2,r3,r2  //     u++;
   sub r0,r3,r0  //   } while(--n);
   jne debut
   sub r1,r7,r0  //   return t-T;
   ret           // }
```

`f(n,t)` rend l'indice de la première occurrence du plus petit élément du tableau parmi les `n` premiers éléments du tableau `t` : `T` pointe toujours sur le début du tableau. `u` pointe successivement sur chacun des éléments du tableau en allant du début jusqu'à la fin. `t` pointe sur le plus petit élément trouvé jusque là et `x` est sa valeur.

Avec le tableau `int tab[]={4,5,2,-7,-5,-7,1,5,3}`, `f(9,tab)` vaut 3.

En remplaçant les 2 `cmovg` par des `cmovl`, on calcule l'indice de la première occurrence du plus grand élément du tableau. Alors `f(9,tab)` vaut 1.

En les remplaçant par des `cmova`, on considère que le tableau contient des entiers non signés. On calcule toujours l'indice de la première occurrence du plus petit élément du tableau, mais en considérant que les nombres négatifs sont plus grands que les positifs. Alors `f(9,tab)` vaut 6.

En les remplaçant par des `cmovge`, en cas d'égalité on prend la deuxième occurrence plutôt que la première. Donc on cherche l'indice de la dernière occurrence du plus petit élément du tableau. Alors `f(9,tab)` vaut 5.

```
g:              //          r0     r1     r2     r3   r4
   loadimm16 r3,1 // int g(int n, int*t)//   s      1  x=(*t)^i
   xor r2,r2,r2   // { int s=0;
debut:            //
   sub r0,r3,r0   //   while(n--)
   jc fin
   load r1,r4     //   { int x=*t;
   mul r4,r4,r4   //     x*=x;          // x=(*t)^2
   mul r4,r4,r4   //     x*=x;          // x=(*t)^4
   add r2,r4,r2   //     s+=x;          // s+=(*t)^4
   mul r4,r4,r4   //     x*=x;          // x=(*t)^8
   sub r2,r4,r2   //     s-=x;          // s-=(*t)^8
   add r1,r3,r1   //     t++;
   jmp debut      //   }
fin:
   mov r2,r0      //   return s;
   ret            // }
int g(int n, int*t)
{ int s=0, x;
  while(n--) x=*t++, x*=x, s+=x*=x, s-=x*=x;
  return s; }
```

Barème

1) 5pt=15x0.33pt

Chaque opération: 0 ou 0.33pt.

2) 5pt

Dessin de la porte logique : 3pt

0.5pt `A` et `B` en parallèle.

0.5pt `A`$\vee$`B` et `C` en série.

0.5pt `H` et `I` en parallèle.

0.5pt `H`$\vee$`I` et `J` en série.

1pt reste du schéma.

-0.5pt pour toute erreur : Il manque 1 ou plusieurs !. Chaque lettre (commande) manquante ou en trop ou mal placée.

Formule de `S` : 2pt

1pt si cela marche pour $\texttt{F} = \texttt{G} = 0$ ou $\texttt{F} = \bar{\texttt{G}} = 0$ ou $\bar{\texttt{F}} = \texttt{G} = 0$.

-0.5pt s'il manque un ou plusieurs non.

-0.5pt pour toute autre erreur.

3) 6pt

`f` en C	2 pt	-0.5pt par erreur comme argument manquant, test de boucle faux, incrément oublié
Que fait `f`?	0.5pt	"première" occurrence ou "plus petit" indice
	0.5pt	"position" ou "indice"
	0.5pt	"minimum" ou "plus petit" élément
`cmovl`	0.5pt	
`cmova`	1pt	
`cmovge`	1pt	

4) 4pt

On ne tient pas compte des commentaires. -0.5pt pour chaque instruction assembleur fausse ou manquante.