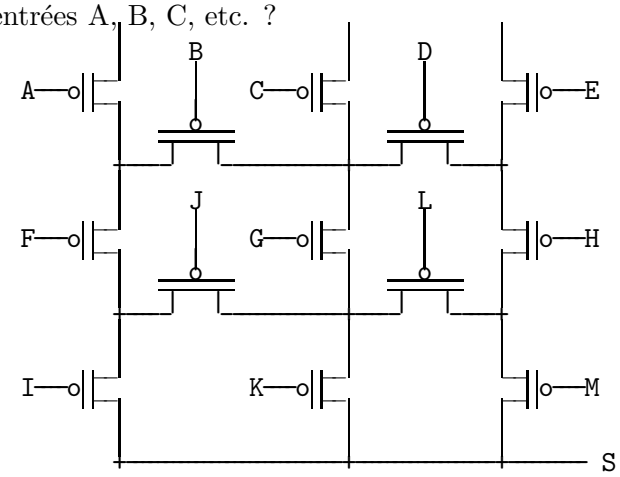
Université de Nanterre jeudi 23 juin 2016
architecture des ordinateurs contrôle

Que valent CF, OF, ZF et SF après les opérations 10+6, 5+3, 8+7, 12-3, 8-10, 5+1, 3-1, 1-3, 8+8, 14+12, 10-8, 3-8, 8-3, 3-12 et 8-8. sur des nombres codés sur 4 bits.

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




```
p:
   loadimm16 r2,1
   add  r0,r1,r0
debut:
   sub r0,r2,r0
   sub r0,r1,r3
   jb  fin
   load r0,r4
   load r1,r5
   sub  r4,r5,r3
   cmovg r5,r3
   cmovg r4,r5
   cmovg r3,r4
   store r0,r4
   store r1,r5
   add r1,r2,r1
   jmp debut
fin:
   ret
```

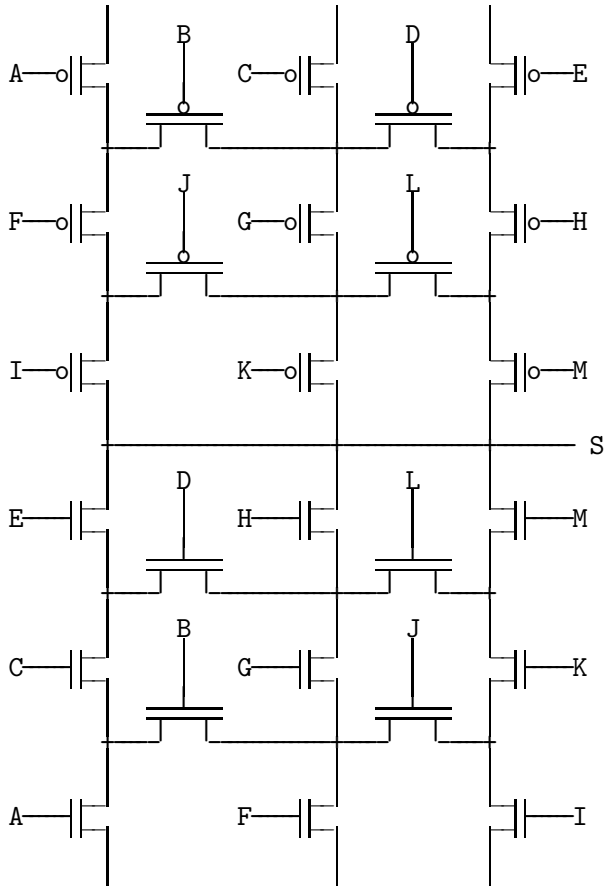
Quel est le contenu du tableau `t` après `int t[]={1,2,-3,-4,5,2,-2,2,-2}; p(9,t);` ?
Donnez un équivalent simple en C de la procédure `p`.
Que fait-elle ?
Refaites tout cet exercice en remplaçant les 3 `cmovg` par des `cmovl` ?
Idem en les remplaçant par des `cmova` ?
Idem en les remplaçant par des `cmovge` ?

Ecrivez en C puis en assembleur la fonction `int g(int n, int *t);` qui rend $\sum_{i=0}^{n-1} t[i]^9$. (On rappelle que le résultat explicite d'une fonction est rendu dans `r0`.)

Corrigé

non signé	signé	CF	OF	ZF	SF
10+6=0	-6+6=0	1	0	1	0
5+3=8	5+3=-8	0	1	0	1
8+7=15	-8+7=-1	0	0	0	1
12-3=9	-4-3=-7	0	0	0	1
8-10=14	-8- -6=-2	1	0	0	1
5+1=6	5+1=6	0	0	0	0
3-1=2	3-1=2	0	0	0	0
1-3=14	1-3=-2	1	0	0	1
8+8=0	-8+-8=0	1	1	1	0
14+12=10	-2+-4=-6	1	0	0	1
10-8=2	-6- -8=2	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
8-8=0	-8- -8=0	0	0	1	0

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



```
p:              //          r0     r1     r2    r3    r4     r5
   loadimm16 r2,1// int p(int n, int*t)//  1      z  y=*u   x=*t
   add  r0,r1,r0 // { int *u=t+n;
debut:          //   do
   sub r0,r2,r0  //   { u--;
   sub r0,r1,r3  //     if(u<t)
   jb  fin       //             return;
   load r0,r4    //     int y=*u,
   load r1,r5    //         x=*t,
   sub  r4,r5,r3 //         z=y-x;
   cmovg r5,r3   //     if(y>x) z=x,
   cmovg r4,r5   //             x=y,
   cmovg r3,r4   //             y=z;
   store r0,r4   //     *u=y;
   store r1,r5   //     *t=x;
   add r1,r2,r1  //     t++;
   jmp debut     //   }
fin:
   ret           // }
```

`p(n,t)` compare et échange éventuellement le premier et le dernier élément du tableau `t` de taille `n`, puis le second et l'avant dernier, puis le troisième et l'antépénultième, etc. de telle sorte que la première moitié du tableau contienne des nombres plus petits que la deuxième moitié. `t` pointe successivement sur chacun des éléments du bas du tableau en allant du début jusqu'au milieu. `u` pointe successivement sur chacun des éléments du haut du tableau en allant de la fin jusqu'au milieu.

Après `int t[]={1,2,-3,-4,5,2,-2,2,-2}; p(9,t);` le tableau `t` contient `{-2,2,-3,-4,5,2,-2,2,1}`

En remplaçant les 2 `cmovg` par des `cmovl`, on met les plus grands éléments au début. `t[]={1,2,-2,2,5,-4,-3,2,-2};`

En les remplaçant par des `cmova`, on considère que le tableau contient des entiers non signés. Les nombres négatifs sont plus grands que les positifs donc `t[]={1,2,-3,2,5,-4,-2,2,-2};`

En les remplaçant par des `cmovge`, cela ne change rien (d'échanger ou non deux nombres égaux). `t[]={-2,2,-3,-4,5,2,-2,2,1}`

```
g:                //          r0     r1     r2     r3   r4    r5
   loadimm16 r3,1 // int g(int n, int*t)//   s      1  x=*t  y=x^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,r5   //         y=x*x;     // y=(*t)^2
   mul r5,r5,r5   //     y*=y;          // y=(*t)^4
   mul r5,r5,r5   //     y*=y;          // y=(*t)^8
   mul r5,r4,r5   //     y*=x;          // y=(*t)^9
   add r2,r5,r2   //     s+=y;          // 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, y;
  while(n--) x=*t++, y=x*x, y*=y, y*=y, s+=y*=x;
  return s; }
```

Barème

1) 5pt=15x0.33pt

Chaque opération: 0 ou 0.33pt.

2) 5pt

Dessin de la porte logique : 3pt

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

Formule de `S` : 2pt

note proportionnelle au nombre de conjonctions correctes (sur les 29)

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

3) 6pt

`f` en C 1pt

Que fait `f`? 1pt

`t` 4pt

`cmovg` 1 pt -0.5pt par nombre faux

`cmovl` 1pt

`cmova` 1pt

`cmovge` 1pt

4) 4pt

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