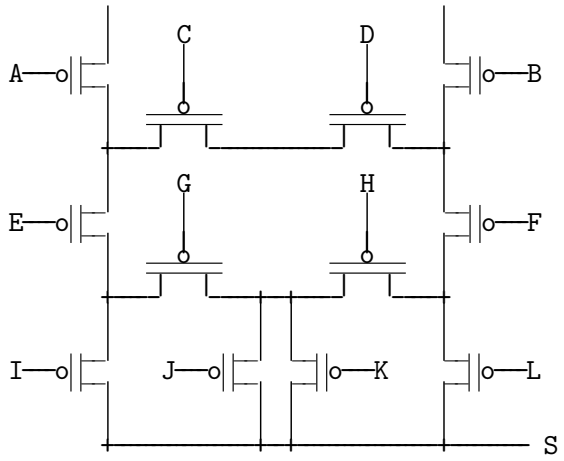
Université de Nanterre vendredi 28 juin 2019
architecture des ordinateurs contrôle

Que valent CF, OF, ZF et SF après les opérations 4+12, 13-14, 11+12, 15-8, 1+9, 4+7, 1+6, 7-9, 6-15, 1-2, 14-3, 7-0, 12+14, 13-7, sur des nombres codés sur 4 bits.

Complétez le bas du schéma avec autant de transistors.




```
f:  loadimm16 r3,1
l1: sub  r0,r3,r0
    jc   l2
    load r1,r4
    load r2,r5
    sub  r4,r5,r6
    movcl r4,r6
    movcl r5,r4
    movcl r6,r5
    store r1,r4
    store r2,r5
    add  r1,r3,r1
    add  r2,r3,r2
    jmp  l1
l2: ret
```

Donnez un équivalent simple en C de la fonction `f`.

Que contient `x` après `int t[]={1,5,-2,3,2,4,5,5}; f(3,t,t+4);`

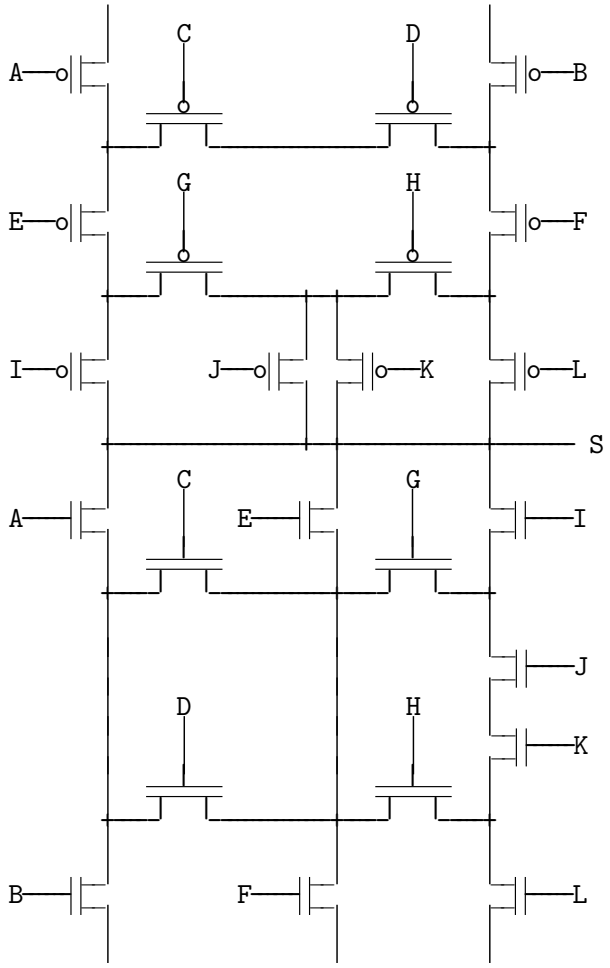
Dites en une phrase ce que fait `f`.

Redonnez le nouveau contenu de `t[]`, si on remplace les trois `movcl` par trois `movcg` ou bien si on les remplace par des `movcb`

Ecrivez en C puis en assembleur la fonction `void g(int n, int *t, int *u);` qui remplace le vecteur `t` à `n` composantes par la somme des deux vecteurs `t` et `u`, en faisant par exemple `t[i]+=u[i]` pour tout `i` entre 0 et `n-1`.

Corrigé

non signé	signé	CF	OF	ZF	SF
4+ 12= 0	4+ -4= 0	1	0	1	0
13- 14=15	-3- -2=-1	1	0	0	1
11+ 12= 7	-5+ -4= 7	1	1	0	0
15- 8= 7	-1- -8= 7	0	0	0	0
1+ 9=10	1+ -7=-6	0	0	0	1
4+ 7=11	4+ 7=-5	0	1	0	1
1+ 6= 7	1+ 6= 7	0	0	0	0
7- 9=14	7- -7=-2	1	1	0	1
6- 15= 7	6- -1= 7	1	0	0	0
1- 2=15	1- 2=-1	1	0	0	1
14- 3=11	-2- 3=-5	0	0	0	1
7- 0= 7	7- 0= 7	0	0	0	0
12+ 14=10	-4+ -2=-6	1	0	0	1
13- 7= 6	-3- 7= 6	0	1	0	0



```
                //           r0     r1     r2     r3  r4 r5 r6
f:  loadimm16 r3,1// void f(int n, int*t, int*u)//   1   x  y  z
l1: sub  r0,r3,r0 // { while(n--)
    jc   l2
    load r1,r4    //   { int x=*t,
    load r2,r5    //         y=*u,
    sub  r4,r5,r6 //         z=x-y;
    movcl r4,r6   //     if(x<y) z=x,
    movcl r5,r4   //             x=y,
    movcl r6,r5   //             y=z;
    store r1,r4   //     *t=x;
    store r2,r5   //     *u=y,
    add  r1,r3,r1 //     t++;
    add  r2,r3,r2 //     u++;
    jmp  l1       //   }
l2: ret           // }
```

```
void f(int n, int*t, int*u)
{ while(n--) { int x=*t++, y=*u++, z; if(x<y) z=x,x=y,y=z; *t++=x, *u++=y; } }
```

```
int t[]={max(1,2),max(5,4),max(−2,5),3,min(1,2),min(5,4),min(−2,5),5}
       ={2,5,5,3,1,4,-2,5}
```

`f(n,t,u)` reçoit deux vecteurs `t` et `u` de taille `n` dont il compare les éléments correspondants, et les échange si l'élément de `t` est le plus petit des deux. Donc on met le plus grand des deux dans `t` et le plus petit dans `u`.

Si on remplace les trois `movcl` par trois `movcg`, la condition est inversée. On prendra le minimum au lieu du maximum et réciproquement.

```
int t[]={min(1,2),min(5,4),min(−2,5),3,max(1,2),max(5,4),max(−2,5),5}
       ={1,4,-2,3,2,5,5,5}
```

Si on remplace les trois `movcl` par trois `movcb`, les entiers rangés dans les deux tableaux sont considérés comme non signés. Un nombre négatif est augmenté de 2^{32}, il est donc plus grand que tout nombre positif.

```
int t[]={max(1,2),max(5,4),max(−2,5),3,min(1,2),min(5,4),min(−2,5),5}
       ={2,5,-2,3,1,4,5,5}
```

```
                //           r0     r1     r2     r3  r4 r5
g:  loadimm16 r3,1// void g(int n, int*t, int*u)//   1   x  y
l1: sub  r0,r3,r0 // { while(n--)
    jc   l2
    load r1,r4    //   { int x=*t,
    load r2,r5    //         y=*u,
    add  r4,r5,r4 //     x+=y;
    store r1,r4   //     *t=x;
    add  r1,r3,r1 //     t++;
    add  r2,r3,r2 //     u++;
    jmp  l1       //   }
l2: ret           // }
void g(int n, int*t, int*u) { while(n--) *t++ +=*u++; }
```

Barème

1) 4pt

Chaque opération partiellement fausse ou manquante : -0.3pt.

2) 6pt

-1pt pour un transistor dont une patte est mal raccordée.
-1.5pt pour un transistor dont les deux pattes sont mal raccordées.

3) 6pt

`f` en C	1.5pt	-0.3pt par erreur
2,5,5,3,1,4,-2,5	1pt	
f en français	1.5pt	
1,4,-2,3,2,5,5,5	1pt	
2,5,-2,3,1,4,5,5	1pt	

4) 4pt

`g` en C	1pt	
`g` en assembleur	3pt	-0.5pt pour chaque instruction fausse ou manquante.