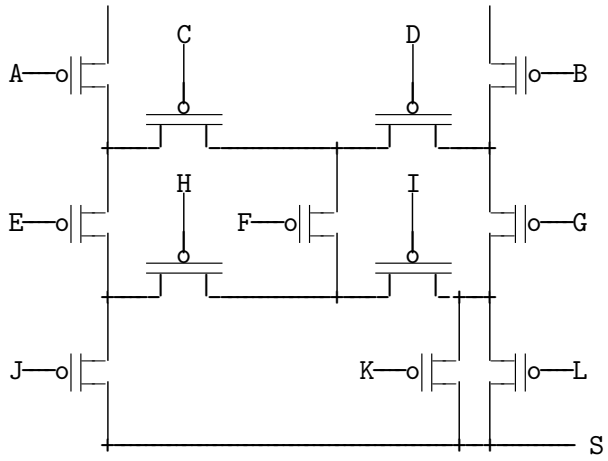
Université de Nanterre lundi 8 avril 2019
architecture des ordinateurs contrôle

Que valent CF, OF, ZF et SF après les opérations 0+15, 10+15, 7-3, 2-10, 11-4, 3+13, 14-15, 9+13, 14-8, 4+4, 0+1, 4-14, 6-7, 13-0, sur des nombres codés sur 4 bits.

Complétez le bas du schéma avec autant de transistors.




```
f:  loadimm16 r2,1
    xor  r1,r1,r1
l1: load r0,r3
    add  r0,r2,r0
    load r0,r4
    add  r0,r2,r0
    sub  r3,r4,r3
    je   l2
    mul  r3,r3,r4
    mul  r3,r4,r3
    mul  r3,r4,r4
    add  r1,r4,r1
    jmp  l1
l2: mov  r1,r0
    ret
```

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

Que vaut `x` dans `int t[]={6,5,3,1,2,2,5,5}, x=f(t);`

Redonnez `x` si dans `t` on remplace 1 par 3.

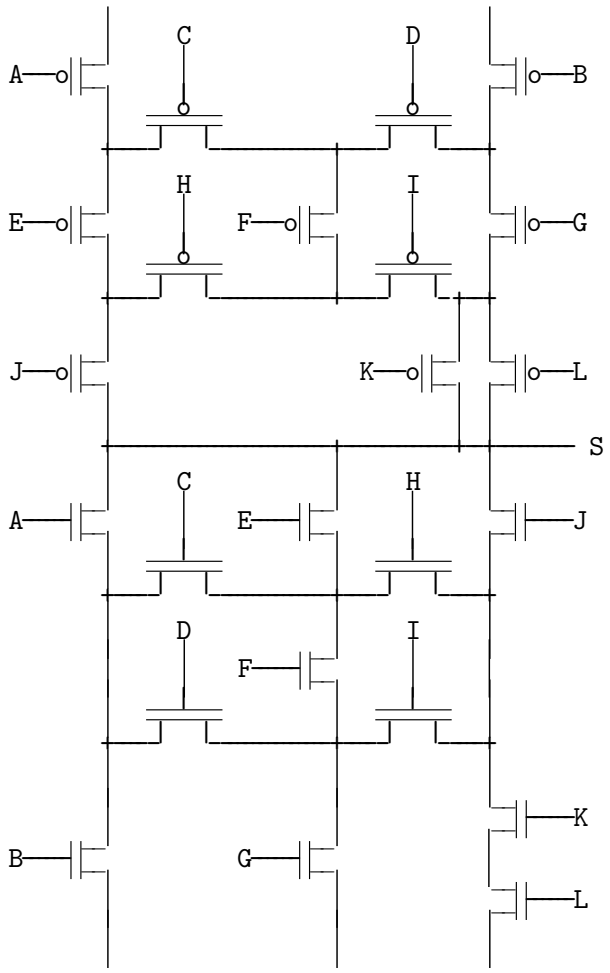
Dites en une phrase ce que calcule `f`.

Redonnez les deux valeurs de `x`, si on enlève le deuxième `add r0,r2,r0`
ou bien si on remplace `mul r3,r4,r3` par `mul r4,r4,r3`
ou bien si on remplace `l2: mov r1,r0` par `l2: mov r3,r0`

Ecrivez en C puis en assembleur la fonction `int g(int *t);` qui rend $t[0]^3 + t[1]^5 + t[2]^8$.

Corrigé

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



```
                //          r0    r1   r2   r3                r4
f:  loadimm16 r2,1// int f(int*t) // s    1    x                 y
    xor  r1,r1,r1 // { int s=0;   //          t[0],t[2],...     t[1],t[3],...
l1:               //   for(;;)               // d=x-y,d^3        d^2,d^5
    load r0,r3    //   { int x=*t++,         // t[2*i]
    add  r0,r2,r0
    load r0,r4    //         y=*t++;         // t[2*i+1]
    add  r0,r2,r0
    sub  r3,r4,r3 //     x-=y;               // d=t[2*i]-t[2*i+1]
    je   l2       //     if(!x) return s;
    mul  r3,r3,r4 //     y=x*x;              // d^2
    mul  r3,r4,r3 //     x*=y;               // d^3
    mul  r3,r4,r4 //     y*=x;               // d^5
    add  r1,r4,r1 //     s+=y;               // s+=d^5
    jmp  l1       //   }
l2: mov  r1,r0    // }
    ret
```

$x = (6-5)^5 + (3-1)^5 = 1^5 + 2^5 = 1 + 32 = 33$ puis $x = (6-5)^5 = 1^5 = 1$

`f(t)` prend les nombres deux par deux dans le tableau `t`, et calcule la somme des puissances cinquièmes de leurs différences. La somme s'arrête au premier terme nul à ajouter.

Si on enlève le deuxième `add r0,r2,r0` à chaque itération de la boucle on n'avance que d'une case dans le tableau au lieu de deux.

$x = (6-5)^5 + (5-3)^5 + (3-1)^5 + (1-2)^5 = 1^5 + 2^5 + 2^5 - 1^5 = 1 + 32 + 32 - 1 = 64$ puis $x = (6-5)^5 + (5-3)^5 = 1^5 + 2^5 = 1 + 32 = 33$

Si on remplace `mul r3,r4,r3` par `mul r4,r4,r3` on élève à la puissance 6 au lieu de 5 : $x = (6-5)^6 + (3-1)^6 = 1^6 + 2^6 = 1 + 64 = 65$ puis $x = (6-5)^6 = 1^6 = 1$

Si on remplace `l2: mov r1,r0` par `l2: mov r3,r0` la fonction rend toujours 0.

```
g:  loadimm16 r2,1
    add  r0,r2,r1   // t+1
    add  r1,r2,r2   // t+2
    load r0,r0      // t[0]
    load r1,r1      // t[1]
    load r2,r2      // t[2]
    mul  r0,r0,r3   // t[0]^2
    mul  r0,r3,r0   // t[0]^3
    mul  r1,r1,r3   // t[1]^2
    mul  r3,r3,r3   // t[1]^4
    mul  r1,r3,r1   // t[1]^5
    mul  r2,r2,r2   // t[2]^2
    mul  r2,r2,r2   // t[2]^4
    mul  r2,r2,r2   // t[2]^8
    add  r0,r1,r0
    add  r0,r2,r0
    ret
```

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	1pt-0.3pt par erreur comme argument manquant, test de boucle faux, incrément oublié
x=33	0.5pt
x=1	0.5pt
f en français	1pt
x=64	0.5pt
x=33	0.5pt
x=65	0.5pt
x=1	0.5pt
x=0	1pt

4) 4pt

`g` en C	1pt
`g` en assembleur	3pt

-0.3pt pour chaque instruction fausse ou manquante.