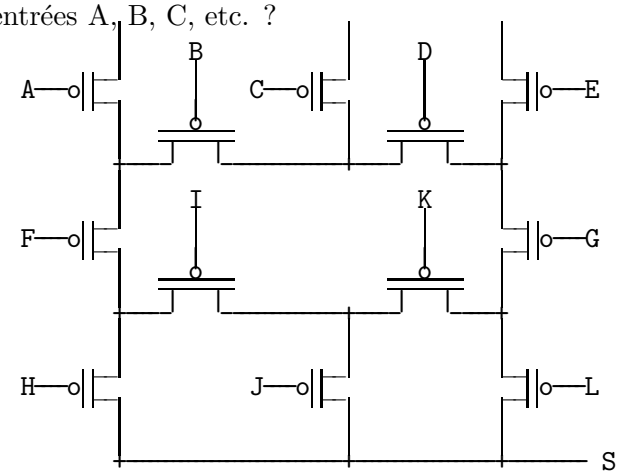
Université de Nanterre lundi 18 avril 2016
architecture des ordinateurs contrôle

Que valent CF, OF, ZF et SF après les opérations 9+7, 9+10, 9-10, 10-9, 3+4, 11+2, 11-2, 2-11, 6-6, 5+6, 5-4, 6-3, 8+7, 8-7 et 7-8 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. ?




```
p:
   loadimm16 r3,1
   sub r0,r3,r0
   jc fin
debut:
   load r1,r4
   load r2,r5
   sub  r4,r5,r6
   mov  r4,r6
   cmovg r5,r4
   cmovg r6,r5
   store r1,r4
   store r2,r5
   add r1,r3,r1
   add r2,r3,r2
   sub r0,r3,r0
   jnc debut
fin:
   ret
```

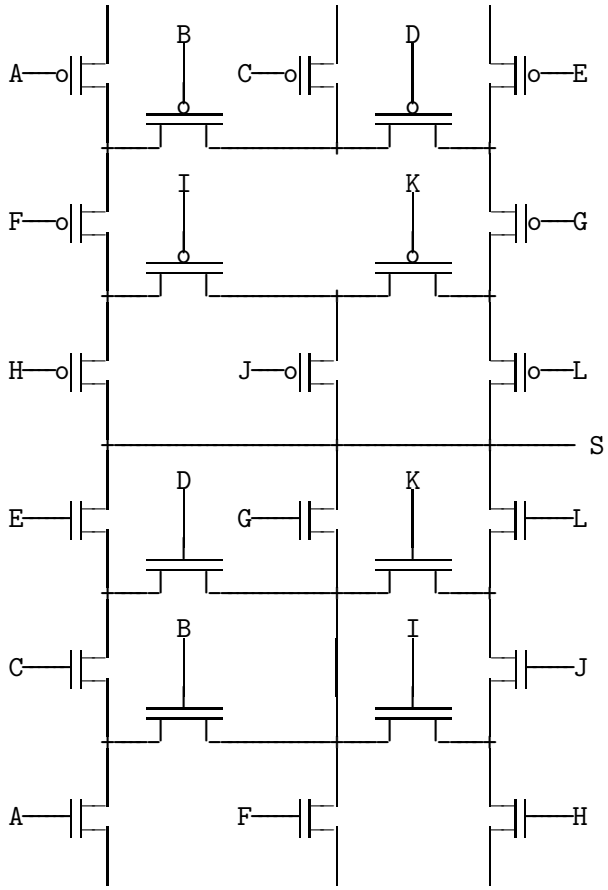
Donner un équivalent simple en C de la procédure `p`. Que fait-elle ?
Que ferait-elle si on remplaçait les 2 `cmovg` par des `cmovl` ?
Que ferait-elle si on les remplaçait par des `cmova` ?
Que ferait-elle si on les remplaçait par des `mov` ?

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

Corrigé

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

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



```
p:              //          r0     r1     r2    r3    r4     r5   r6
   loadimm16 r3,1// void p(int n, int*t, int*u) // 1   x=*t   y=*u   z
   sub r0,r3,r0  // { if(n--)
   jc fin
debut:          //   do
   load r1,r4    //   { int x=*t,
   load r2,r5    //         y=*u,
   sub  r4,r5,r6 //         z=x-y;      //*t-*u;
   mov  r4,r6    //     z=x;            //*t
   cmovg r5,r4   //     if(*t>*u) x=y;  //x=min(*t,*u)
   cmovg r6,r5   //     if(*t>*u) y=z;  //y=max(*t,*u)
   store r1,r4   //     *t=x;
   store r2,r5   //     *u=y;              void p(int n, int*t, int*u)
   add r1,r3,r1  //     t++;               { while(n--)
   add r2,r3,r2  //     u++;                 { int x=*t, y=*u, z=x;
   sub r0,r3,r0  //   } while(n--)             if(x>y) x=y, y=z;
   jnc debut                                  *t++=x, *u++=y;
fin:ret         // }                        } }
```

La procédure `p` compare chacun des `n` éléments du tableau `t` avec l'élément correspondand du tableau `u`, et les échange éventuellement, pour qu'àprès cela le plus petit et le plus grand des deux soient respectivement dans `t` et dans `u`. Par exemple `t[]={-3,-4,6,9}, u[]={1,-5,-8,2}` deviendra `t[]={-3,-5,-8,2}, u[]={1,-4,6,9}`.

Si on remplace les 2 `cmovg` par des `cmovl`, le plus grand est mis dans `t` et le plus petit dans `u`.
On obtient `t[]={1,-4,6,9}, u[]={-3,-5,-8,2}`.

Si on remplace les 2 `cmovg` par des `cmova`, on considère que les deux tableaux contiennent des entiers non signés. Dans le code C, il faut remplacer tous les `int` par des `unsigned`. Autrement dit, les entiers négatifs sont considérés comme plus grands que les positifs.
On obtient `t[]={1,-5,6,2}, u[]={-3,-4,-8,9}`.

Si on remplace les 2 `cmovg` par des `mov`, les contenus des deux tableaux sont échangés.
On obtient `t[]={1,-5,-8,2}, u[]={-3,-4,6,9}`.

```
f:              //          r0     r1     r2     r3   r4    r5   r6  r7
   loadimm16 r3,1 // int f(int n, int*t, int*u) //  1  x=*t  y=*u   z   s
   xor r7,r7,r7  // { int s=0;
debut:           //
   sub r0,r3,r0  //   while(n--)
   jc fin
   load r1,r4    //   { int x=*t,
   load r2,r5    //         y=*u,
   sub  r4,r5,r6 //         z=x-y;     // *t-*u;
   cmovg r5,r4   //     if(*t>*u) x=y; // x=min(*t,*u)
   mul r4,r4,r4  //     x*=x;          // x=min(*t,*u)^2
   mul r4,r4,r4  //     x*=x;          // x=min(*t,*u)^4
   add r4,r7,r7  //     s+=x;          // s+=min(*t,*u)^4
   add r1,r3,r1  //     t++;
   add r2,r3,r2  //     u++;
   jmp debut     //   }
fin:
   mov r7,r0     //   return s;
   ret           // }
```

```
int f(int n, int*t, int*u)
{ int s=0;
  while(n--)
  { int x=*t++, y=*u++;
    if(x>y) x=y;
    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

-1pt si le dessin du bas est le même qu'en haut : Il manque la liaison verticale.

-0.5pt pour toute autre 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 $\mathtt{F} = \mathtt{G} = 0$ ou $\mathtt{F} = \bar{\mathtt{G}} = 0$ ou $\bar{\mathtt{F}} = \mathtt{G} = 0$.

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

-0.5pt pour toute autre erreur.

3) 6pt

`p` en C	2pt-0.5pt par erreur comme argument manquant, test de boucle faux, incrément oublié
Que fait `p` ?	1pt
`cmovl`	1pt
`cmova`	1pt
`mov`	1pt

4) 4pt

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