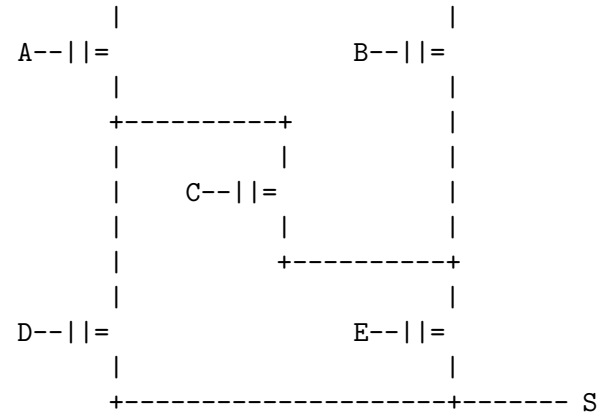
Université de Nanterre jeudi 25 juin 2015
architecture des ordinateurs contrôle

Que valent CF, OF, ZF et SF après les opérations 3+4, 5-13, 11+2, 7-14, 5+11, 13+13, 7-7, 3-6, 5+6, 14-7, 8+9, 13-4 et 11-15 sur des nombres codés sur 4 bits.

Rajouter les 5 transistors du bas du schéma. Que vaut la sortie S (en fonction des cinq entrées A, B, C, D et E) ?




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

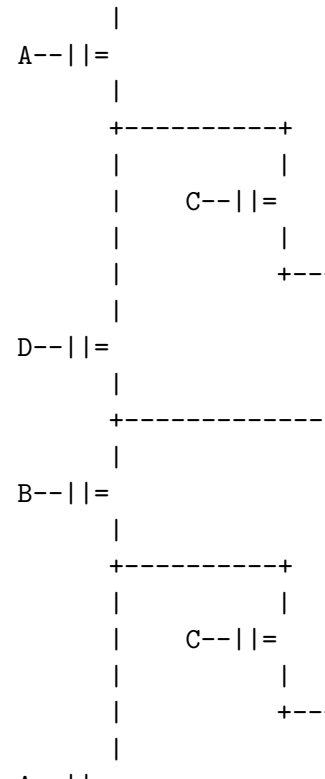
Donner un équivalent simple en C de la procédure `p`. Que fait-elle ? Que ferait-elle si on remplaçait `shrimm` et `sarimm` par `shlimm` ?

Compiler la procédure

```
void g(int n, int *t, int *u, int *v) { while(n--) v[n]=*t++ * *u++; }
```

Corrigé

$S = (\bar{A} \wedge \bar{D}) \vee (\bar{B} \wedge \bar{E}) \vee (\bar{C} \wedge (\bar{A} \vee \bar{B}) \wedge (\bar{D} \vee \bar{E})) = (\bar{A} \wedge \bar{D}) \vee (\bar{B} \wedge \bar{E}) \vee (\bar{A} \wedge \bar{C} \wedge \bar{E}) \vee (\bar{B} \wedge \bar{C} \wedge \bar{D})$




```
//  r0   r1   r2   r3   r4   r5   r6
//  n    t    u    1   u+n   x    y
p:
   loadimm16 r3,1  // 1
   sub r0,r3,r0    // n--
   jc fin          // if(n==-1) goto fin
debut:
   add r0,r2,r4    // u+n
   load r1,r5      // x=*t
   load r4,r6      // y=u[n]
   shrimm r5,1,r5  // (unsigned)x>>=1
   sarimm r6,1,r6  // y>>=1
   store r1,r6     // *t=y
   store r4,r5     // u[n]=x
   add r1,r3,r1    // t++
   sub r0,r3,r0    // n--
   jnc debut       // if(n!=-1) goto debut
fin:
   ret
void p(int n, int *t, int *u)
{ while(n--)
  { unsigned x=*t;
    int y=u[n];
    *t++=y>>1;
    u[n]=x>>1;
  }
}
```

```
//  r0   r1   r2   r3   r4   r5   r6   r7
//  n    t    u    v    1   v+n   *t   *u
g:
   loadimm16 r4,1   // 1
   sub r0,r4,r0     // n--
   jc fing          // if(n==-1) goto fing
debutg:
   add r0,r3,r5     // v+n
   load r1,r6       // *t
   load r2,r7       // *u
   mul r6,r7,r6     // *u * *t
   store r5,r6      // v[n]=*t * *u
   add r1,r4,r1     // t++
   add r2,r4,r2     // u++
   sub r0,r4,r0     // n--
   jnc debutg       // if(n!=-1) goto debutg
fing:
   ret
```

Si par exemple `int t[]={a,b,c}, u[]={d,e,f};` alors après `p(3,t,u)` les contenus de `t` et `u` sont échangés, retournés et divisés par deux. `t[]={f>>1,e>>1,d>>1}, u[]={c>>1,b>>1,a>>1}` mais `a`, `b` et `c` sont signés alors que `d`, `e` et `f` sont non signés.

Si on remplace les `sar` (Shift Arithmetic Right, décalage à droite signé) les `shr` (SHift Right, décalage à droite non signé) par des `shl` (SHift Left, décalage à gauche) alors au lieu de diviser par 2 on multiplie par 2.

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

Barème

1) 6.5pt=13x0.5pt
Chaque opération: $\pm 1/6 \pm 1/6 \pm 1/12 \pm 1/12$ tronqué dans $[0, 0.5]$.

2) 5pt
dessin de la porte logique : 3pt
-0.5pt si 1 ou plusieurs !
-1pt si il manque le transistor commandé par `C` ou s'il y en a plusieurs
-1.5pt si `A`, `B`, `D` et `E` sont permutés ou -2pt s'il n'apparaissent pas une fois chacun
formule de `S` : 2pt
1pt si cela marche pour `C=1` ou `C=0`
-0.5pt s'il manque un ou plusieurs non.

3) 5.5pt
`p` en C 3pt-0.5pt par erreur comme argument manquant, test de boucle faux, incrément oublié
que fait `p` ? 1.5pt=3x0.5 échange de 2 tableaux, retornés, divisés par 2
`shl` 1pt

4) 3pt On ne tient pas compte des commentaires. -0.5pt pour chaque instruction assembleur fausse ou manquante.