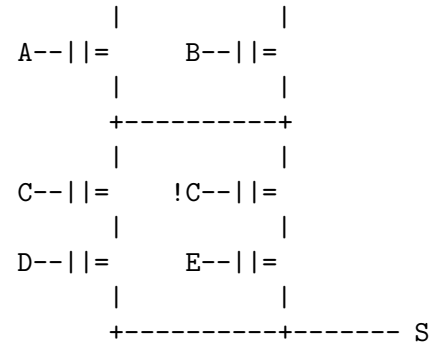
Université de Nanterre lundi 18 mai 2015
architecture des ordinateurs contrôle

Que valent CF, OF, ZF et SF après les opérations 2+3, 4+5, 12+2, 4+12, 9+13, 12+14, 5-5, 4-7, 13-6, 14-5, 6-13, 5-14, 13-13 et 12-14 sur des nombres codés sur 4 bits.

Rajouter les 6 transistors du bas du schéma. Que vaut la sortie S (en fonction des cinq entrées A, B, C, D et E ?




```
f:
   loadimm16 r5,1
   load r1,r2      // r2=*r1
   sub r0,r5,r0
   jc fin
debut:
   load r1,r3
   sub r3,r2,r10
   cmovg r3,r2
   add r1,r5,r1
   sub r0,r5,r0
   jnc debut
fin:
   mov r2,r0
   ret
```

Donner un équivalent simple en C de la fonction `f`. Que calcule-t-elle ? Que calculerait-elle si on remplaçait `cmovg` par `cmovge` ? ou par `cmovl` ? ou par `cmova` ? Ecrire une version de `f` qui n'utilise pas de déplacement conditionnel. Cette version est-elle meilleure ou pire ? Pourquoi ?

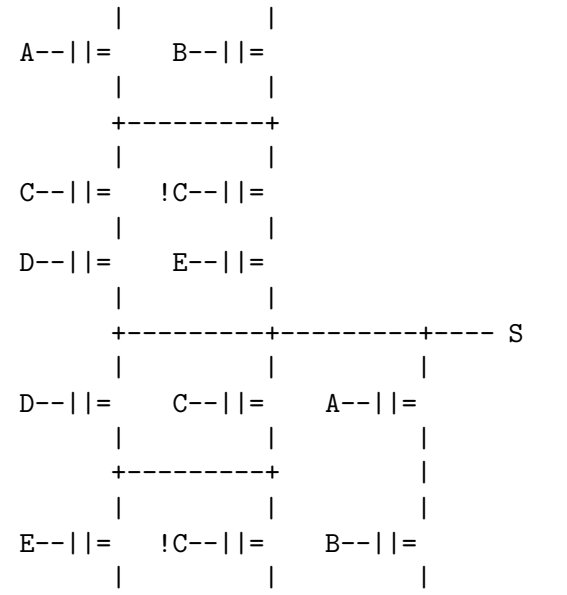
Compiler la fonction

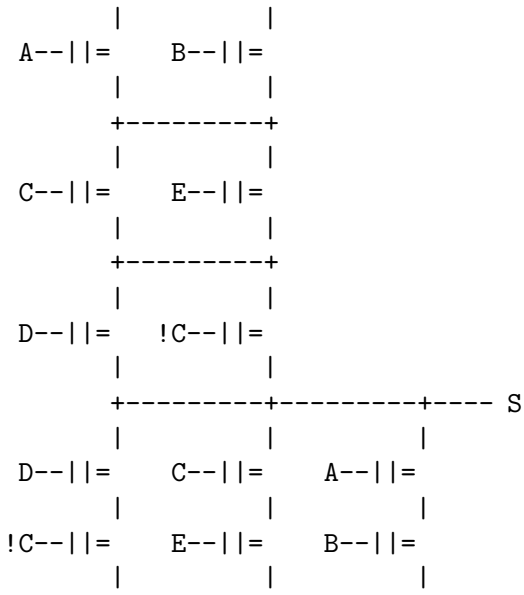
```
void g(int n, int *t, int *u, int *v)
{ while(n--) *v++=*t<*u? *t :*u, t++, u++;
}
```

On pourra utiliser une instruction comme `store r1,r2 // *r1=r2`

Corrigé

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






```
//  r0   r1   r2   r3   r5   r10
//  n    t    m    *t   1    *t-m
f:
   loadimm16 r5,1 // 1
   load r1,r2     // m=t[0];
   sub r0,r5,r0   // n--
   jc fin         // if(!n--) goto fin
debut:
   load r1,r3     // *t
   sub r3,r2,r10  // *t-m
   cmovg r3,r2    // if(*t>m) m=*t
   add r1,r5,r1   // t++
   sub r0,r5,r0   // n--
   jnc debut      // if(n--) goto debut
fin:
   mov r2,r0      // m
   ret            // return m

void f(int n, int *t)
{ int m=*t;
  while(n--)
  { if(*t>m) m=*t;
    t++;
  }
  return m;
}
```

```
void g(int n, int *t, int *u, int *v)
{ while(n--)
  { if(*t<*u) *v=*t;
    else      *v=*u;
    t++, u++, v++;
  }
}
//  r0  r1  r2  r3  r5  r6  r7  r10
//  n   t   u   v   1   *t  *v  *t-*u
g:
   loadimm16 r5, 1 // 1
   sub r0,r5,r0    // n--
   jc fin2         // if(!n--) goto fin
debut2:
   load r1,r6      // *t
   load r2,r7      // *u
   sub r6,r7,r10   // *t-*u
   cmovl r6,r7     // max(*t,*u)
   store r3,r7     // *v=max(*t,*u)
   add r1,r5,r1    // t++
   add r2,r5,r2    // u++
   add r3,r5,r3    // v++
   sub r0,r5,r0    // n--
   jnc debut2      // if(n--) goto debut
fin2:
   ret
```

La fonction `f` calcule le plus grand des `n` éléments du tableau `t`.

Si on remplace `cmovg` par `cmovge`, on remplace `if(*t>m) m=*t;` par `if(*t>=m) m=*t;`. Les deux font `m=max(m,*t);`. Cela ne change pas le temps d'exécution ni le résultat de la fonction.

Avec `cmovl` on a `if(*t<m) m=*t;` et la fonction calcule le plus petit élément.

Avec `cmova` la comparaison est non signée. Les `n` éléments du tableau `t` sont non signés et la fonction rend le plus grand d'entre eux.

On peut remplacer la ligne `cmovg r3,r2` par `jng suite; mov r3,r2; suite:`. Le programme est plus long et plus lent, car `cmovg` et `mov` sont deux intructions rapides qui prennent le même temps, mais `jng` peut être très lente si la prédiction des branchements est fausse.

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

Barème

1) 7pt=14x0.5pt

Chaque opération: $\pm 1/6 \pm 1/6 \pm 1/12 \pm 1/12$ tronqué dans $[0, 0.5]$.

2) 4.5pt

6 transistors commandés par A, B, C, $\bar{C}$, D et E : 1pt, -0.5pt par erreur

A et B en série: 0.5pt

C et D en parallèle: 0.5pt

E et $\bar{C}$ en parallèle: 0.5pt

CD et $E\bar{C}$ en série: 0.5pt

AB et $CDE\bar{C}$ en parallèle: 0.5pt

formule $S = (\bar{A} \vee \bar{B}) \wedge (\bar{C} \wedge \bar{D} \vee C \wedge \bar{E})$: 1pt - 0.5pt si not faux, -0.5pt par erreur (littéral, opérateur ou parenthèse)

3) 5.5pt

`f` en C	2pt
`f(n,t)` $= \max_{i=0}^{n-1}$ `t[i]`	1pt
`cmovge` inchangé	0.5pt
`cmovl` minimum	0.5pt
`cmova` non signé	0.5pt
`jng suite; mov r3,r2; suite:`	0.5pt
branchement conditionnel beaucoup plus lent	0.5pt

4) 3pt

On ne tient pas compte des commentaires. Chaque instruction assembleur fausse ou manquante -1/3pt.