Architecture Interne AM2901

Université Pierre et Marie Curie
Master ACSI
Outils pour la Conception VLSI

AM2900

- Advanced Micro Devices, 1975

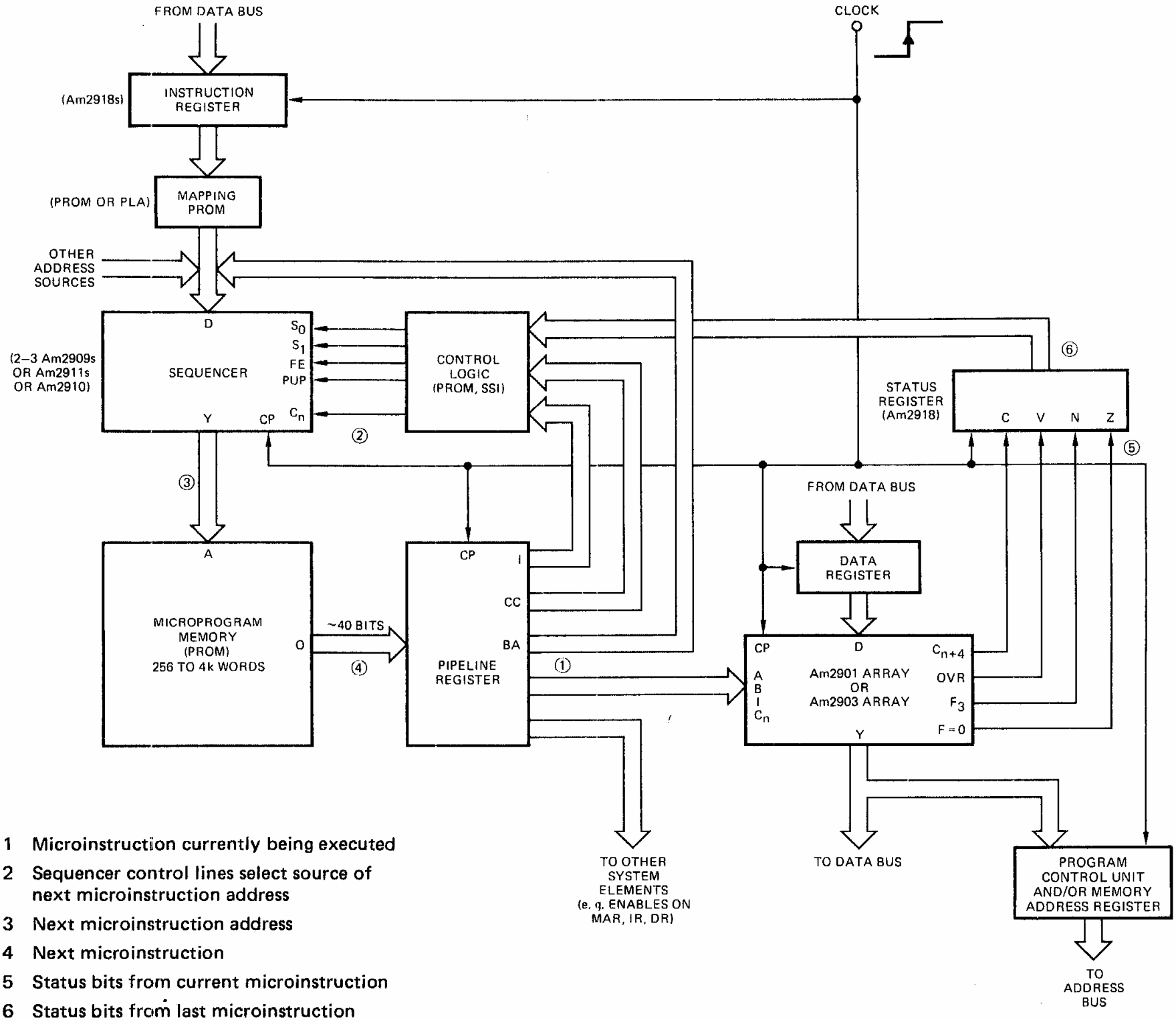
- Famille de circuits pour processeurs en tranche

  AM2901 : ALU + décalage

  \+ AM2910 : micro-séquenceur

  \+ plusieurs dizaines d'autres

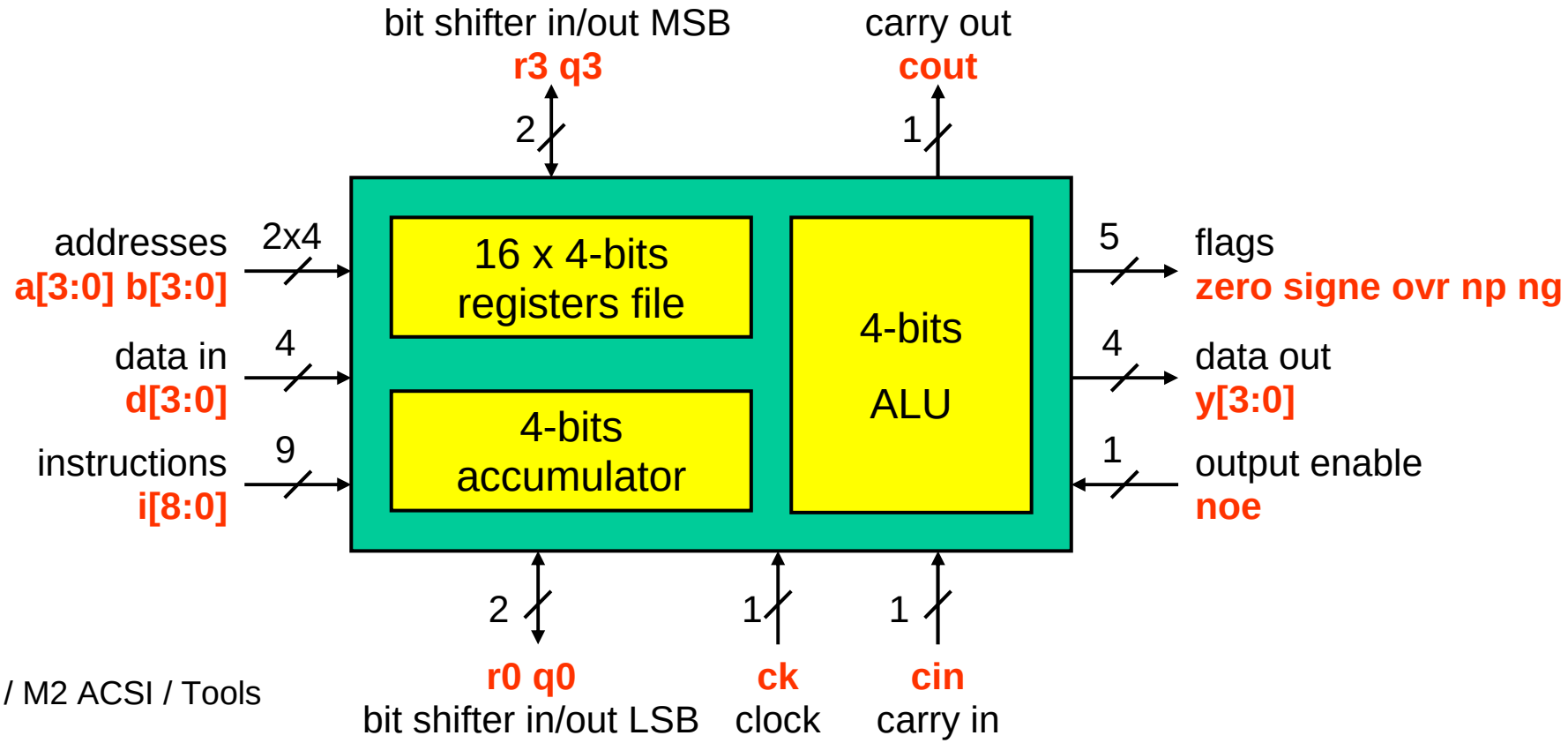
  Processeur



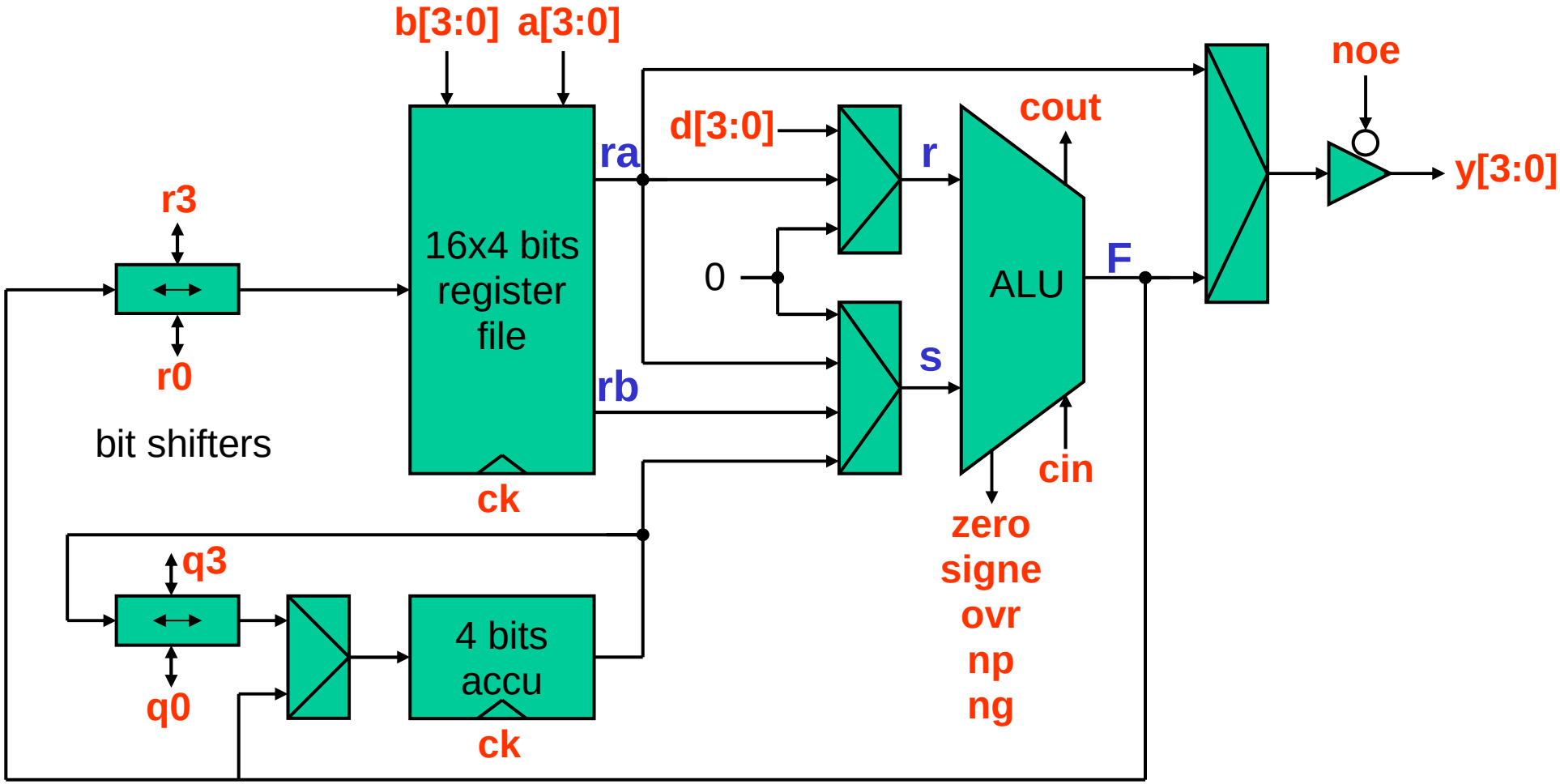
1 Microinstruction currently being executed
2 Sequencer control lines select source of next microinstruction address
3 Next microinstruction address
4 Next microinstruction
5 Status bits from current microinstruction
6 Status bits from last microinstruction

Vue d'ensemble AM2901

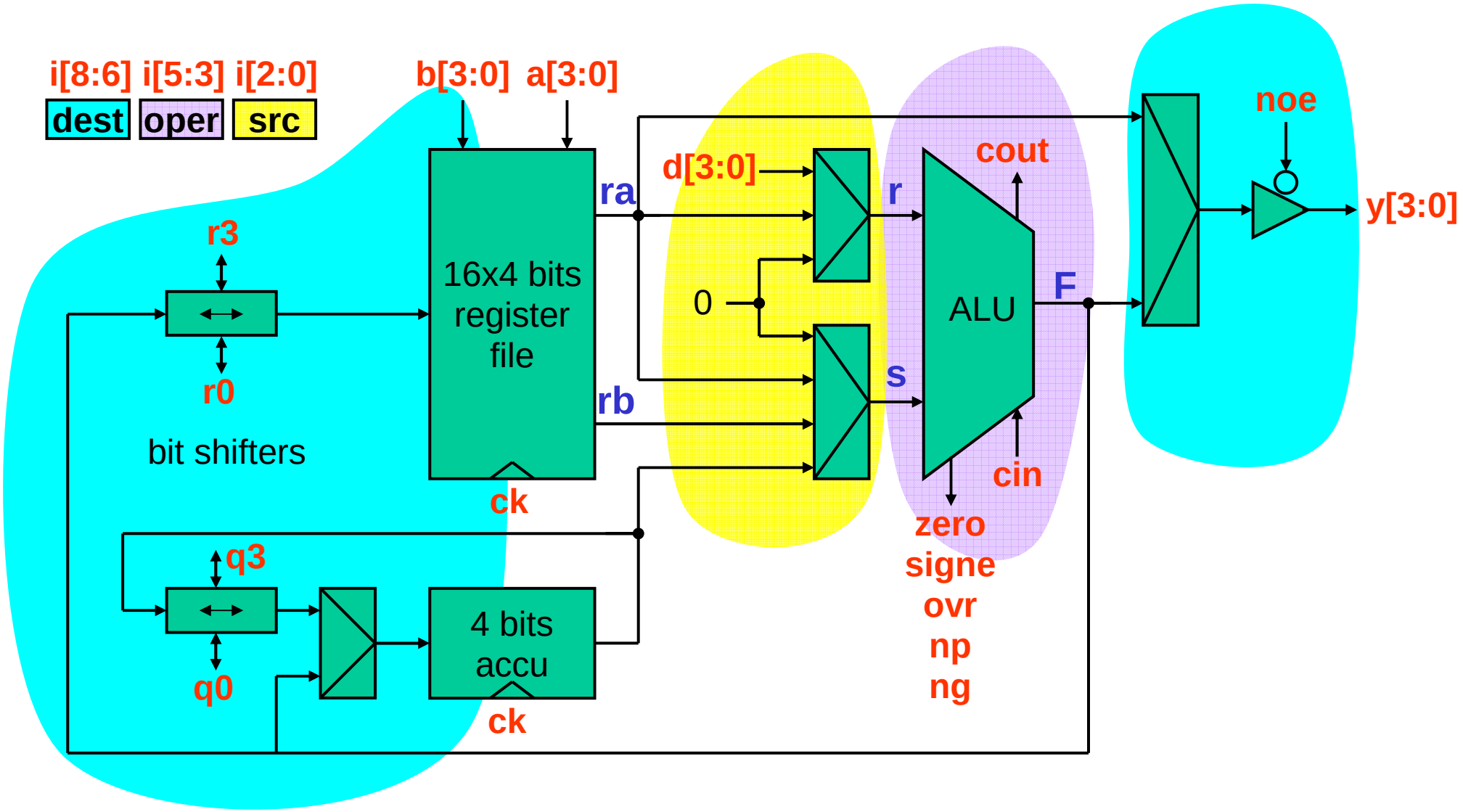
- ALU 4 bits: Arithmetical and Logical Unit
- 16 registres généraux 4 bits
- 1 accumulateur 4 bits
- 8 opérations + 4 décalages de bits
- 40 broches



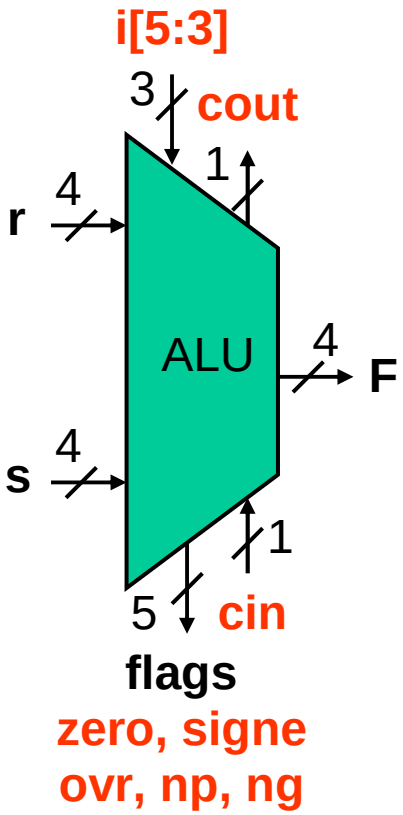
Architecture interne



Architecture interne



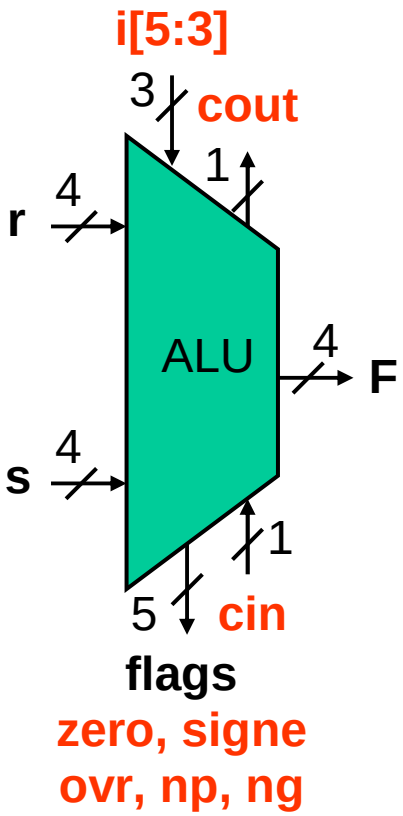
ALU



- Boîte à opérations

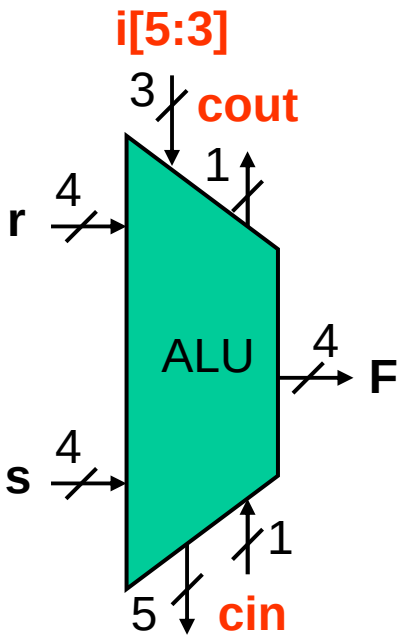
  en fonction de l'instruction **i[5:3]**
  + - OR AND XOR NXOR
- **cin** et **cout** pour mettre en cascade les ALU
- les drapeaux donnent des informations sur le résultat **zero, signe, ovr, np, ng**

ALU : instructions



instruction	Effet
0	**r** + **s**
1	**s** - **r**
2	**r** - **s**
3	**r** OR **s**
4	**r** AND **s**
5	$\overline{\textbf{r}}$ AND **s**
6	**r** XOR **s**
7	**r** NXOR **s**

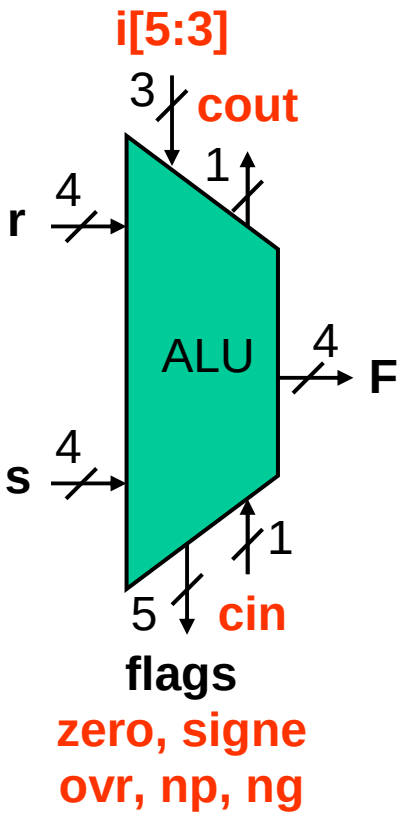
ALU : flags



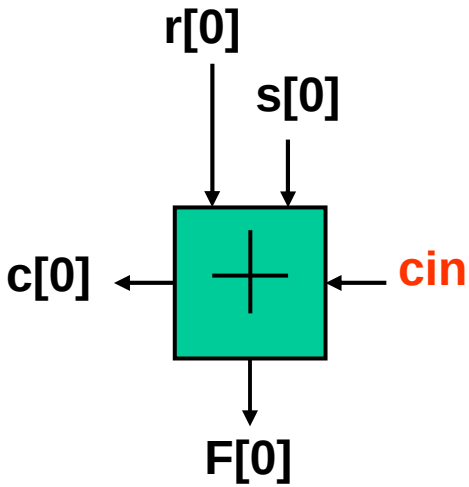
flags	définition
signe	bit de poids fort de **F** : **F[3]**
zero	à 1 si **F**==0
overflow **ovr**	dépassement de capacité en cpl. à 2 i.e. le résultat a le bon signe.
propagation **np**	à 0 si les données **r** et **s** permettent la propagation d'une éventuelle retenue. si **np**==0 alors si **cin**==1 alors **cout**=1
génération **ng**	à 0 si les données **r** et **s** génèrent une retenue en sortie quelquesoit la retenue en entrée. si **ng**==0 alors **cout**=1

$cout = \overline{ng} + \overline{np} \cdot cin$

ALU : addition bit à bit

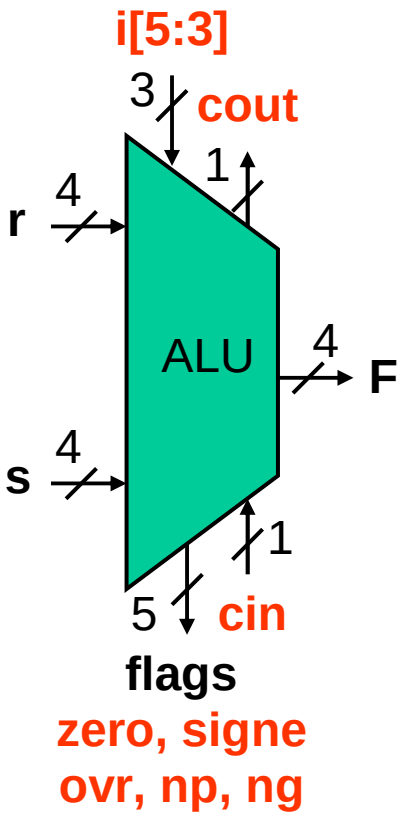


	c[2]	c[1]	c[0]	**cin**
	r[3]	r[2]	r[1]	r[0]
+	s[3]	s[2]	s[1]	s[0]
cout	F[3]	F[2]	F[1]	F[0]



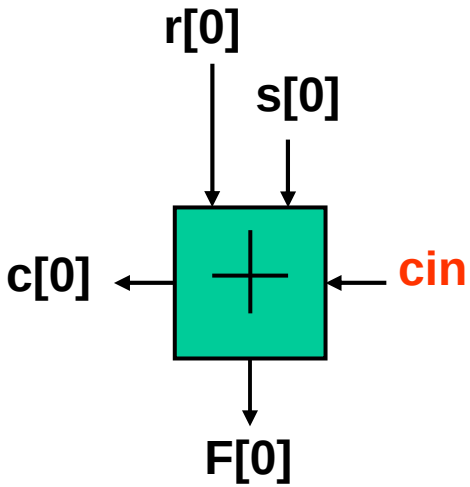
Additionneur
3 nombres de 1 bit

ALU : addition bit à bit



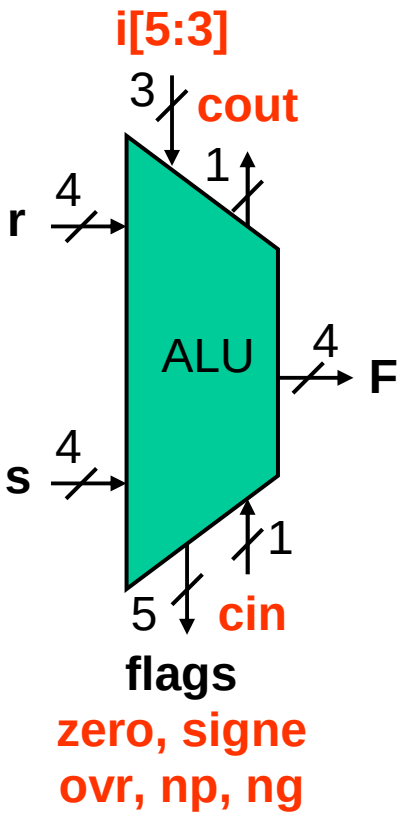
	c[2]	c[1]	c[0]	**cin**
	r[3]	r[2]	r[1]	r[0]
+	s[3]	s[2]	s[1]	s[0]
cout	F[3]	F[2]	F[1]	F[0]

cin	r[0]	s[0]	c[0]	F[0]
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Additionneur
3 nombres de 1 bit

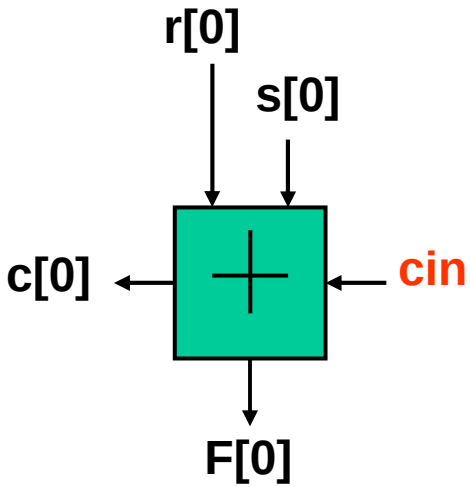
ALU : addition bit à bit



	c[2]	c[1]	c[0]	**cin**
	r[3]	r[2]	r[1]	r[0]
+	s[3]	s[2]	s[1]	s[0]
cout	F[3]	F[2]	F[1]	F[0]

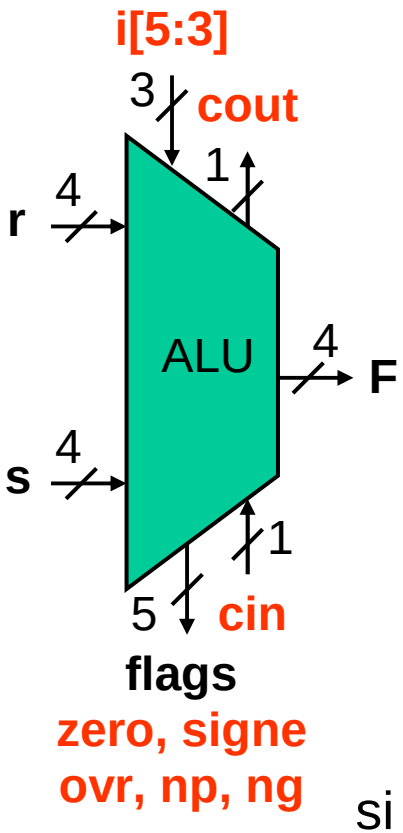
$F[0] = cin \oplus r[0] \oplus s[0]$

$c[0] = cin \,.\, (r[0]+s[0]) + r[0].s[0]$



Additionneur
3 nombres de 1 bit

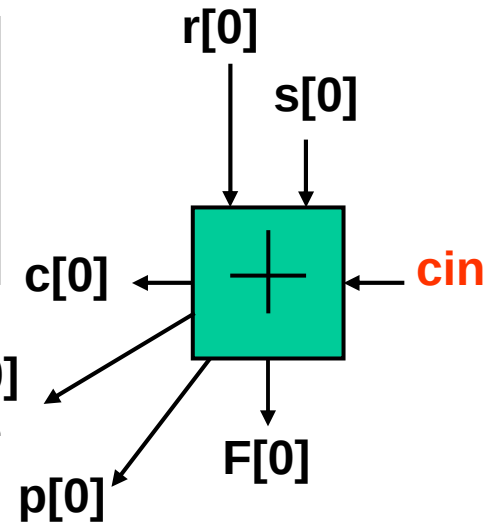
ALU : propagation et génération



	c[2]	c[1]	c[0]	**cin**
	r[3]	r[2]	r[1]	r[0]
+	s[3]	s[2]	s[1]	s[0]
cout	F[3]	F[2]	F[1]	F[0]

$$F[0] = cin \oplus r[0] \oplus s[0]$$

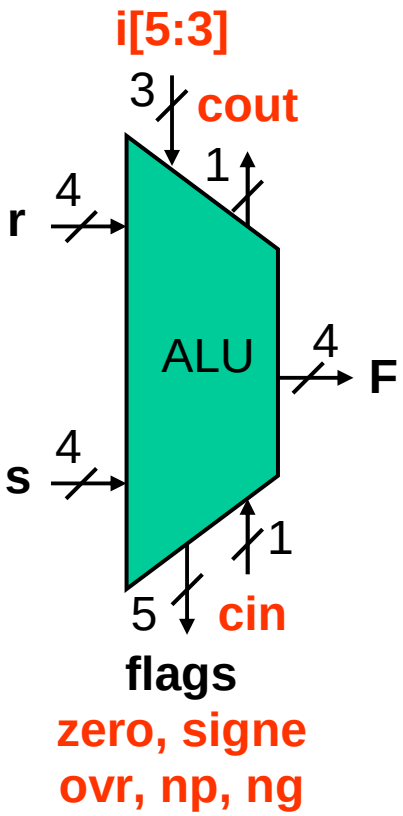
$$c[0] = cin \,.\, \underbrace{(r[0]+s[0])}_{p[0]} + \underbrace{r[0].s[0]}_{g[0]}$$

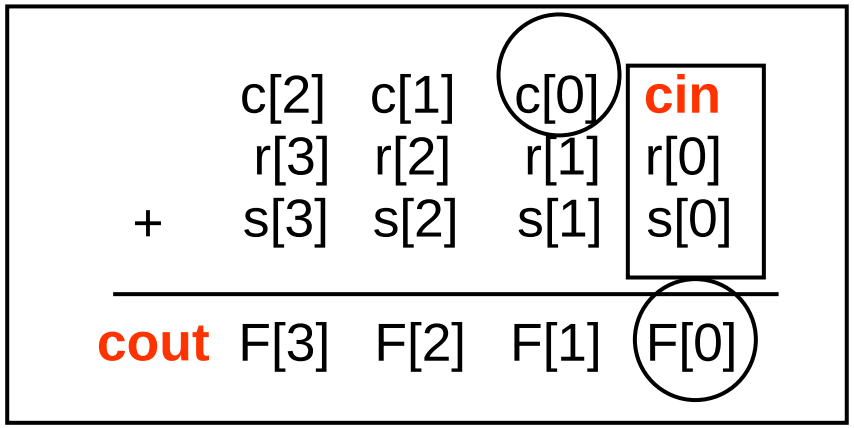


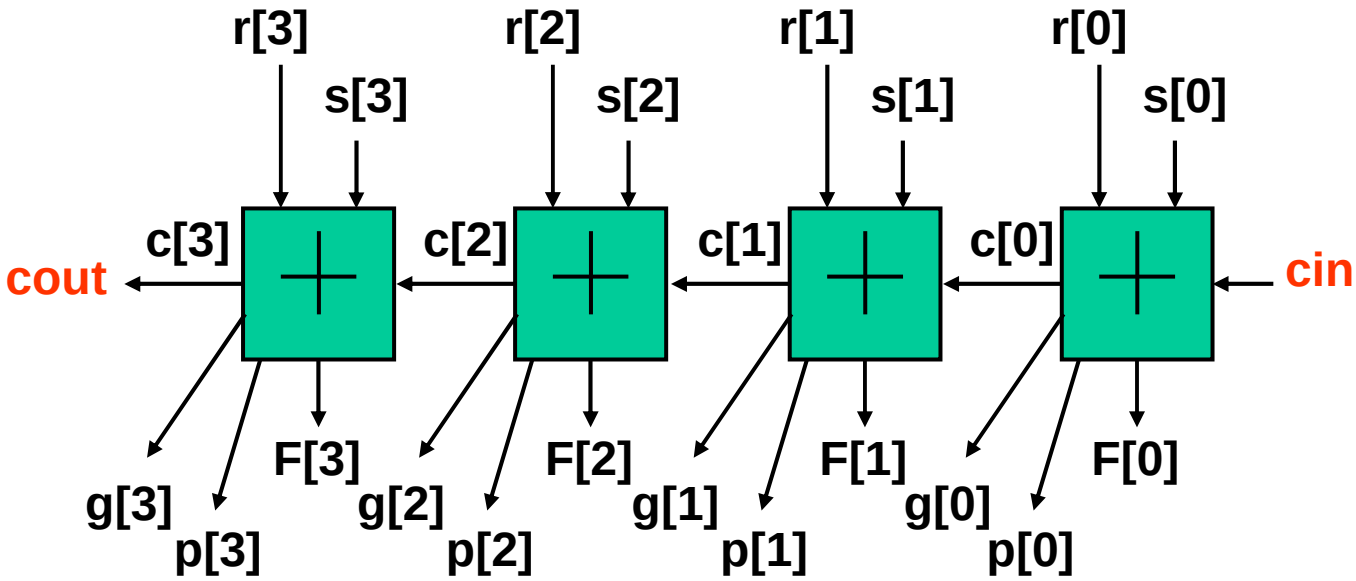
si g[0] est à 1, l'étage génère une retenue

si p[0] est à 1, l'étage propage la retenue entrante

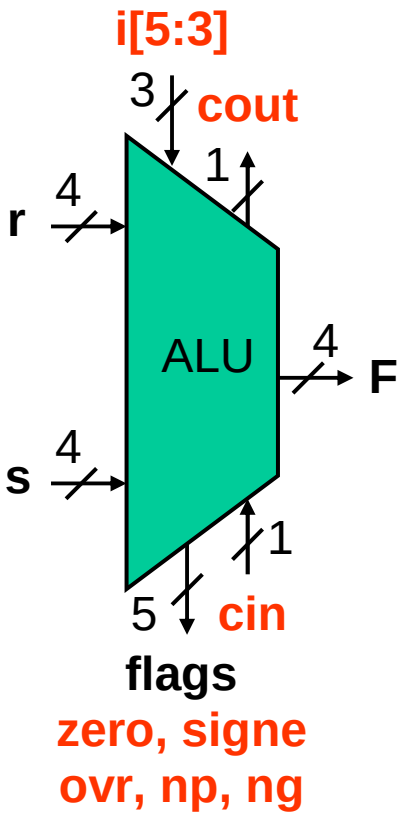
ALU : cascade d'additionneurs







ALU : addition en VHDL



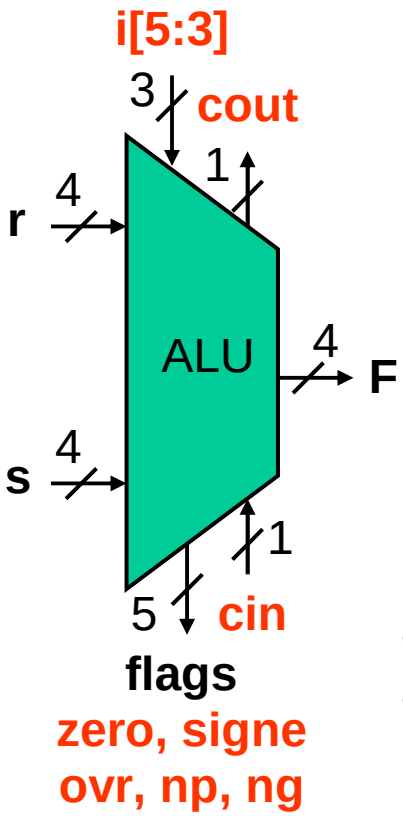
	c[2]	c[1]	c[0]	**cin**
	r[3]	r[2]	r[1]	r[0]
+	s[3]	s[2]	s[1]	s[0]
cout	F[3]	F[2]	F[1]	F[0]

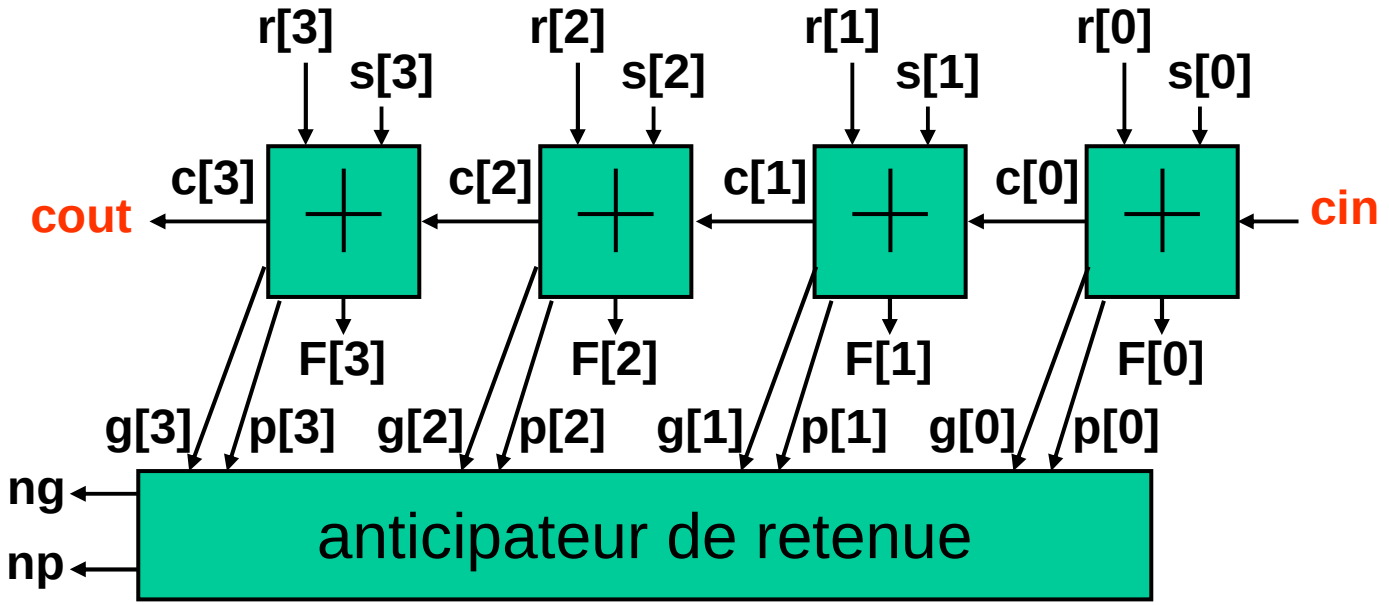
$F[0] = cin \oplus r[0] \oplus s[0]$
$p[0] = r[0] + s[0]$
$g[0] = r[0] \ . \ s[0]$
$c[0] = cin \ . \ p[0] + g[0]$

En VHDL on utilise les vecteurs de bits

```
F   <= (c[2 downto 0] & cin)  XOR  r  XOR  s;
p   <= r  OR  s;
g   <= r  AND  s;
c   <= ((c[2 downto 0] & cin)  AND  p )  OR  g ;
```

ALU : anticipateur de retenue





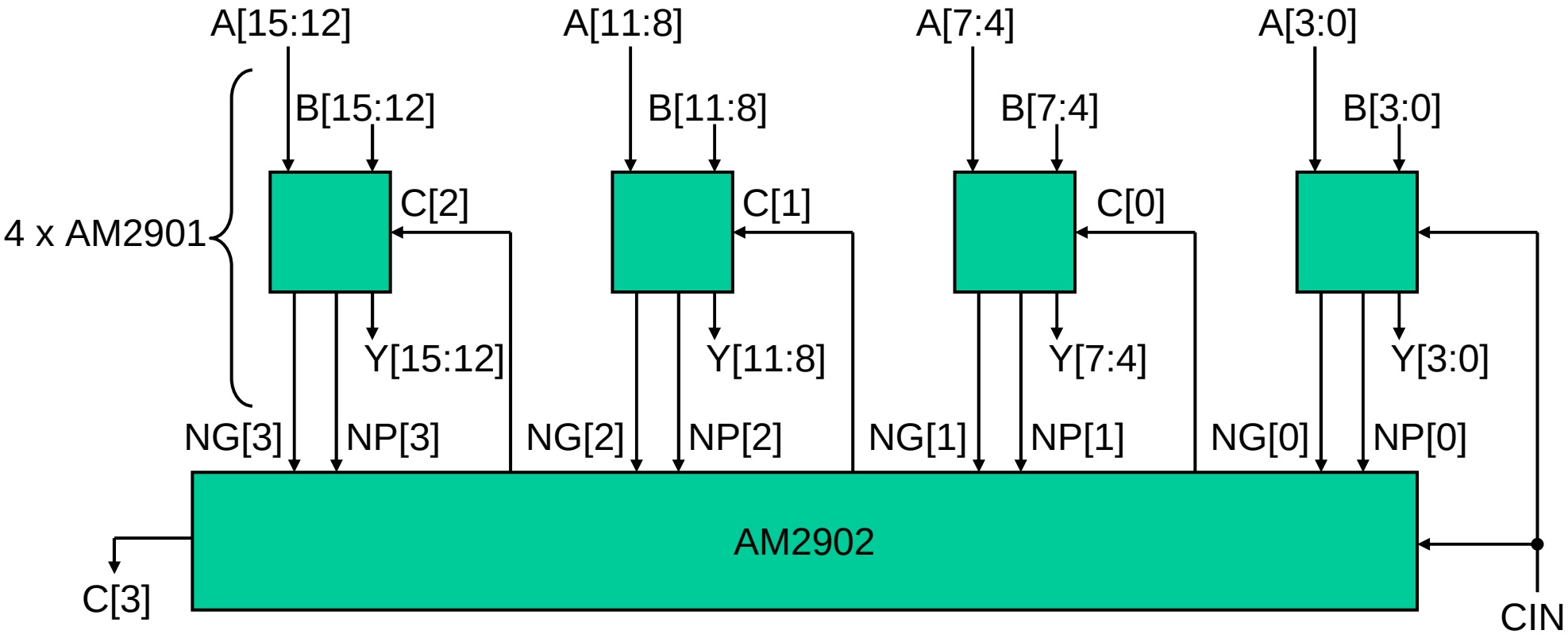
- l'ensemble propage la retenue si tous les étages la propage
- l'ensemble génère une retenue le dernier étage en génère une

ou si l'avant dernier étage en génère une
et que le dernier étage la propage

etc…

np = p[3].p[2].p[1].p[0]
ng = g[3] + p[3].g[2] + p[3].p[2].g[1] + p[3].p[2].p[1].g[0]

ALU : avantage de l'anticipateur



$$C[0] = \overline{NG[0]} + \overline{NP[0]}.CIN$$

$$C[1] = \overline{NG[1]} + \overline{NG[0]}.\overline{NP[1]} + \overline{NP[1]}.\overline{NP[0]}.CIN$$

Etc…

Soustracteur

- Pour réaliser le soustracteur, on utilise l'additionneur…

  A – B = A + -B

  en notation complément à 2^n : -B = not(B) + 1

  … en faisant une négation de l'entrée soustraite et en ajoutant 1

  A – B = A + not(B) + 1

- ATTENTION

  L'ajout du 1 n'est pas fait dans l'AM2901

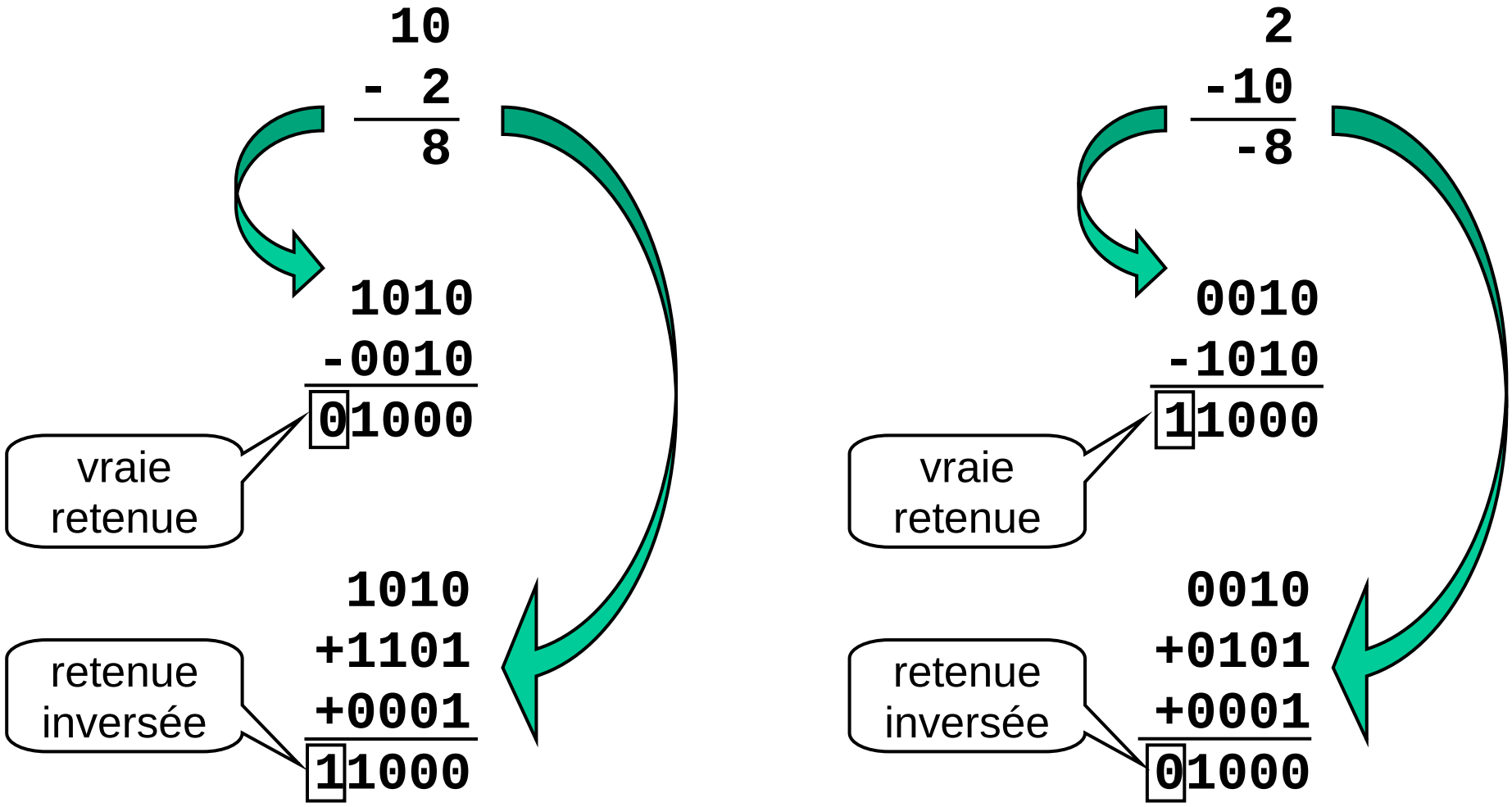
  ➔ si on veut faire une soustraction, il faut :

  1 - programmer l'instruction soustraction

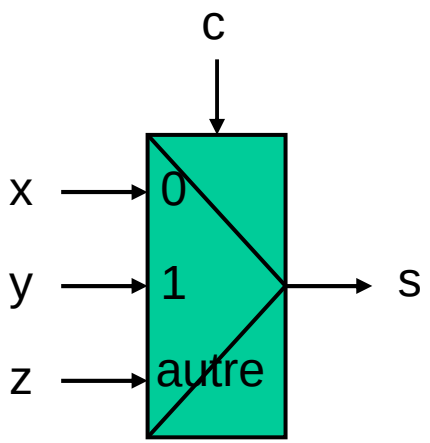
  2 - introduire 1 sur l'entrée CIN

  3 - la retenue sortante COUT doit être inversée avant d'être interprétée

Soustracteur exemples



Multiplexeur



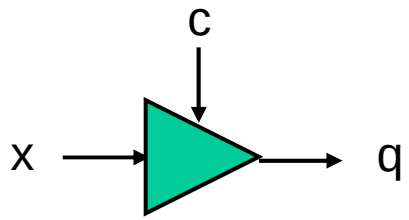
- permet le choix parmi plusieurs données en fonction d'une commande

```
with c select s<=
   x when B"00",
   y when B"01",
   z when others;
```

```
s  <=  x when c=B"00"
   else y when c=B"01"
   else z ;
```

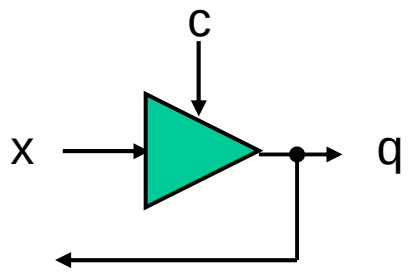
Portes 3 états et bidirectionnelles

- Pour la sortie des données



si c==1 alors q=x
si c==0 alors q=HZ

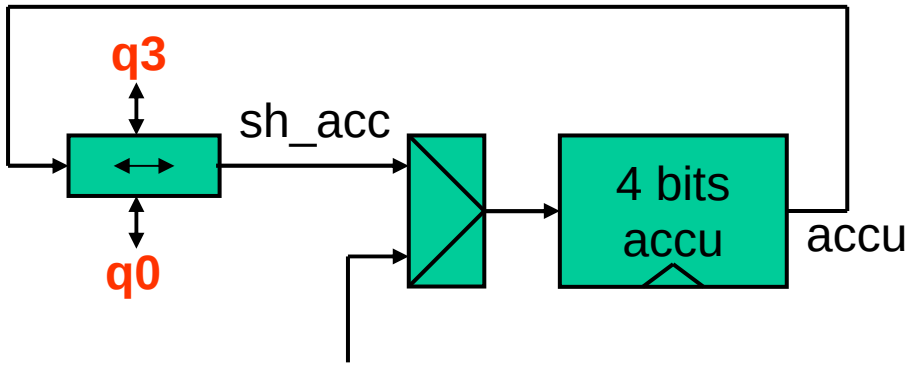
- Pour les entrées/sorties des décaleurs (bit shifters)

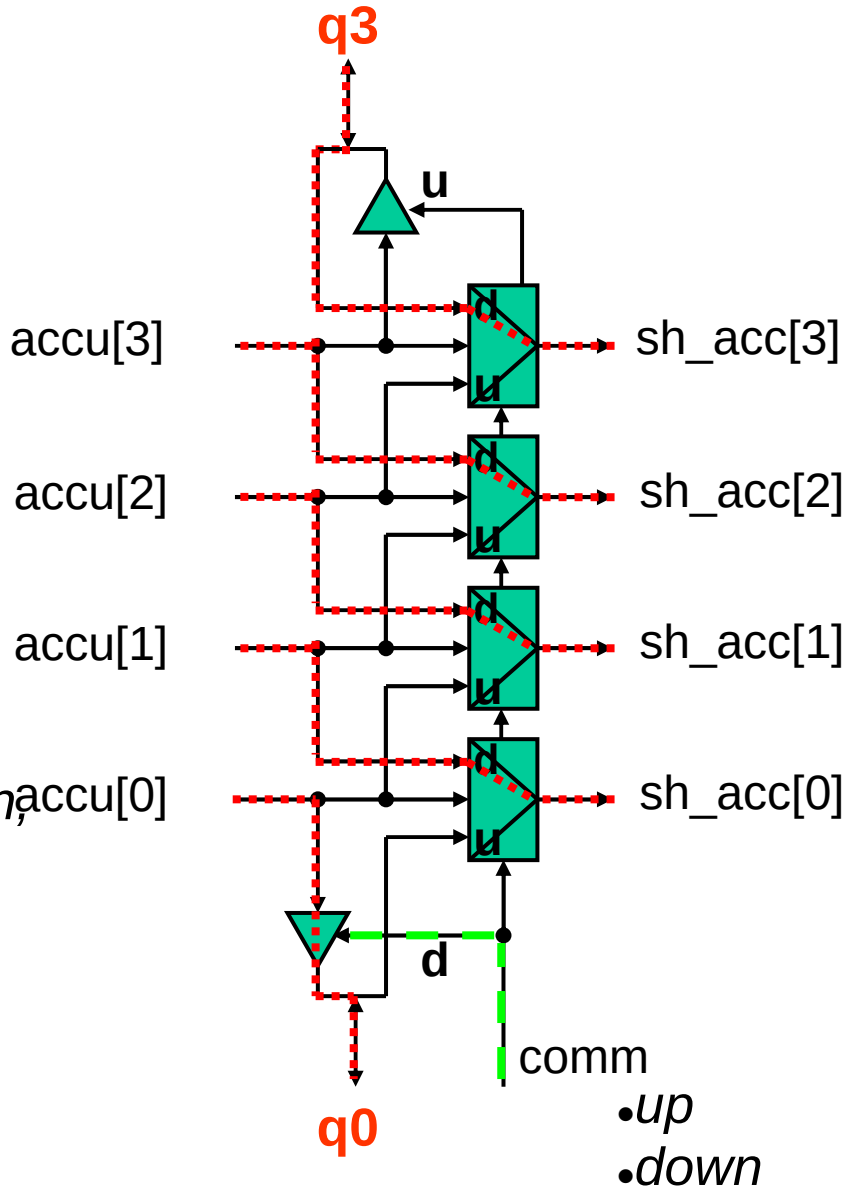



```
ts : block (c = '1')
begin
        q <= guarded x;
end block;
```

q est un signal de type :
- mux_bit bus
- wor_bit bus

Bit – shifter

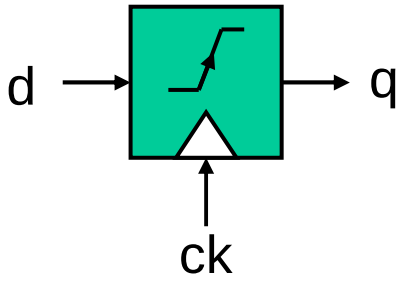


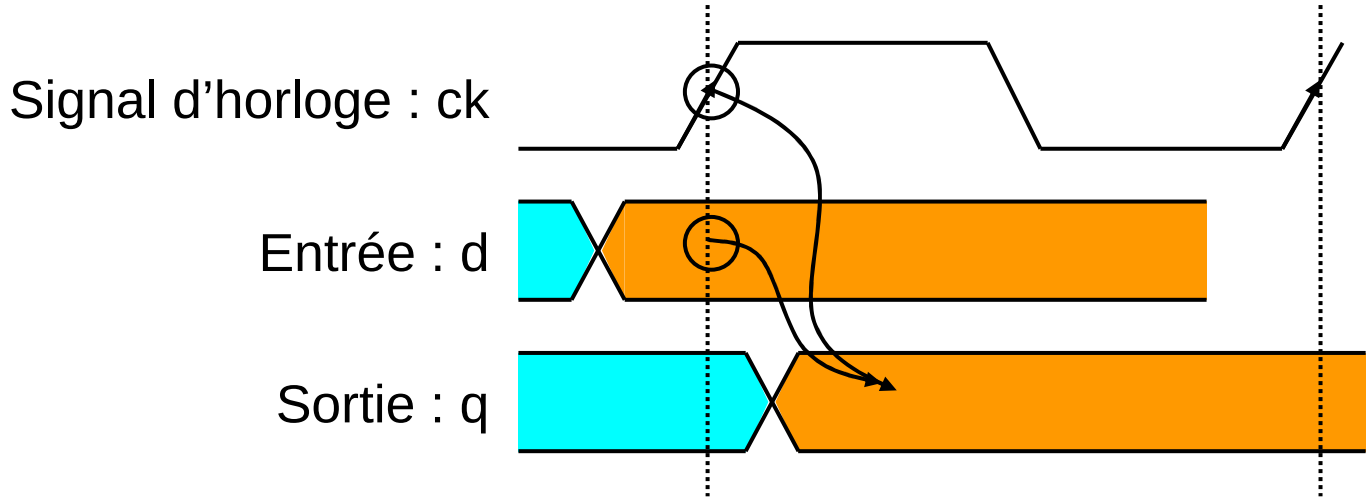



```
with comm select sh_acc <=
        (q3 & accu(3 downto 0)) when down,
        (accu(2 downto 0) & q0) when up
        accu when others;
```

Bascule D

Mémoire enregistrement sur front

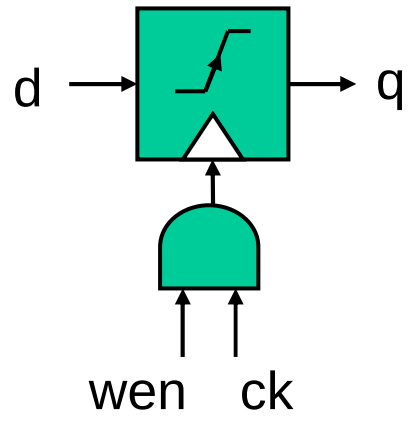




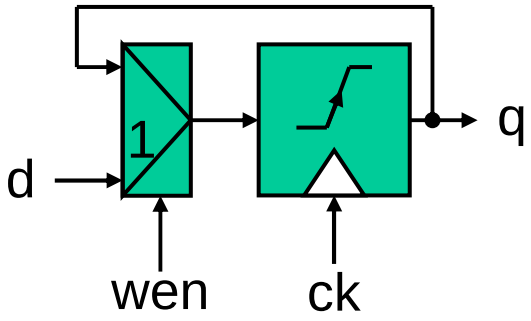

```
lab : block (ck = '1' AND ck'event)
begin
        q <= guarded d;
end block;
```

Bascule D

écriture conditionnelle : deux méthodes



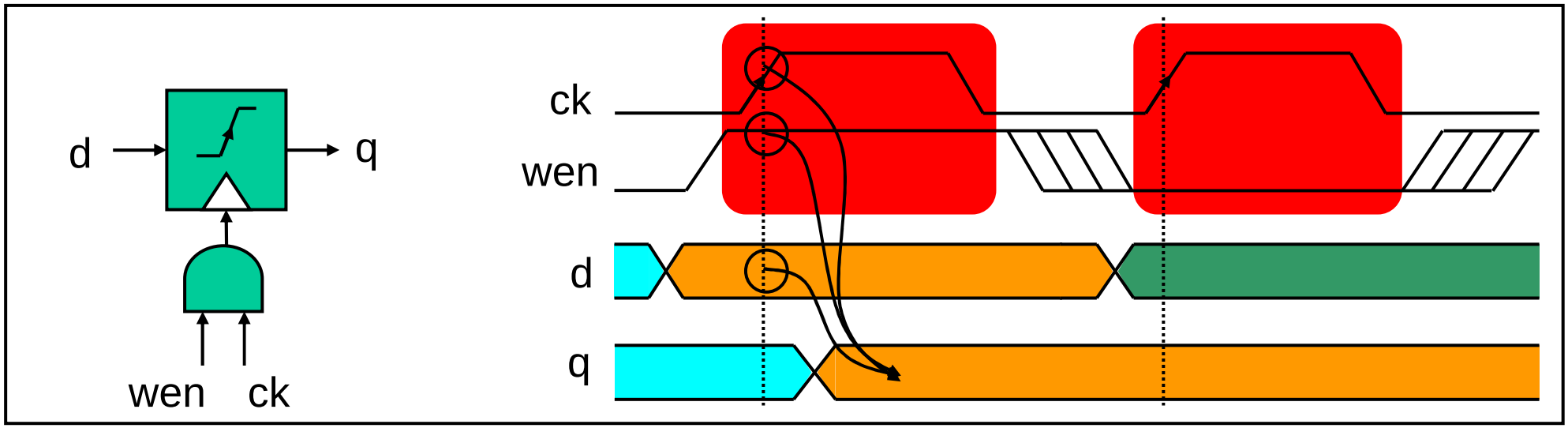

```
lab : block ((ck and wen)= ‘1’ AND ck’event)
begin
        q <= guarded q;
end block;
```

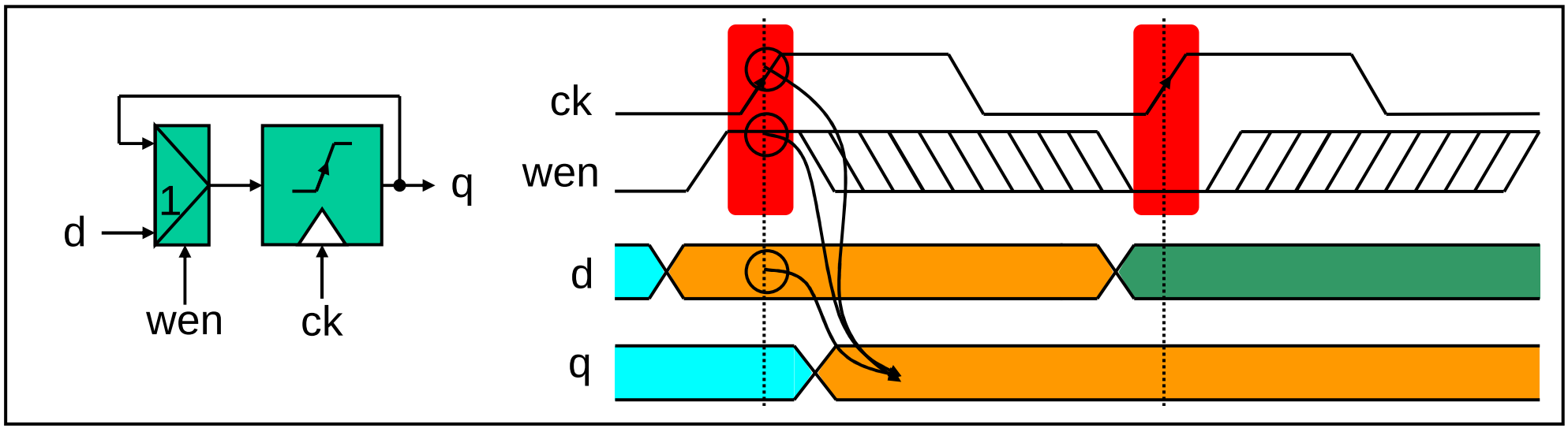



```
lab : block (ck = ‘1’ AND ck’event)
begin
        q <= guarded when wen=‘1’ else q;
end block;
```

Bascule D

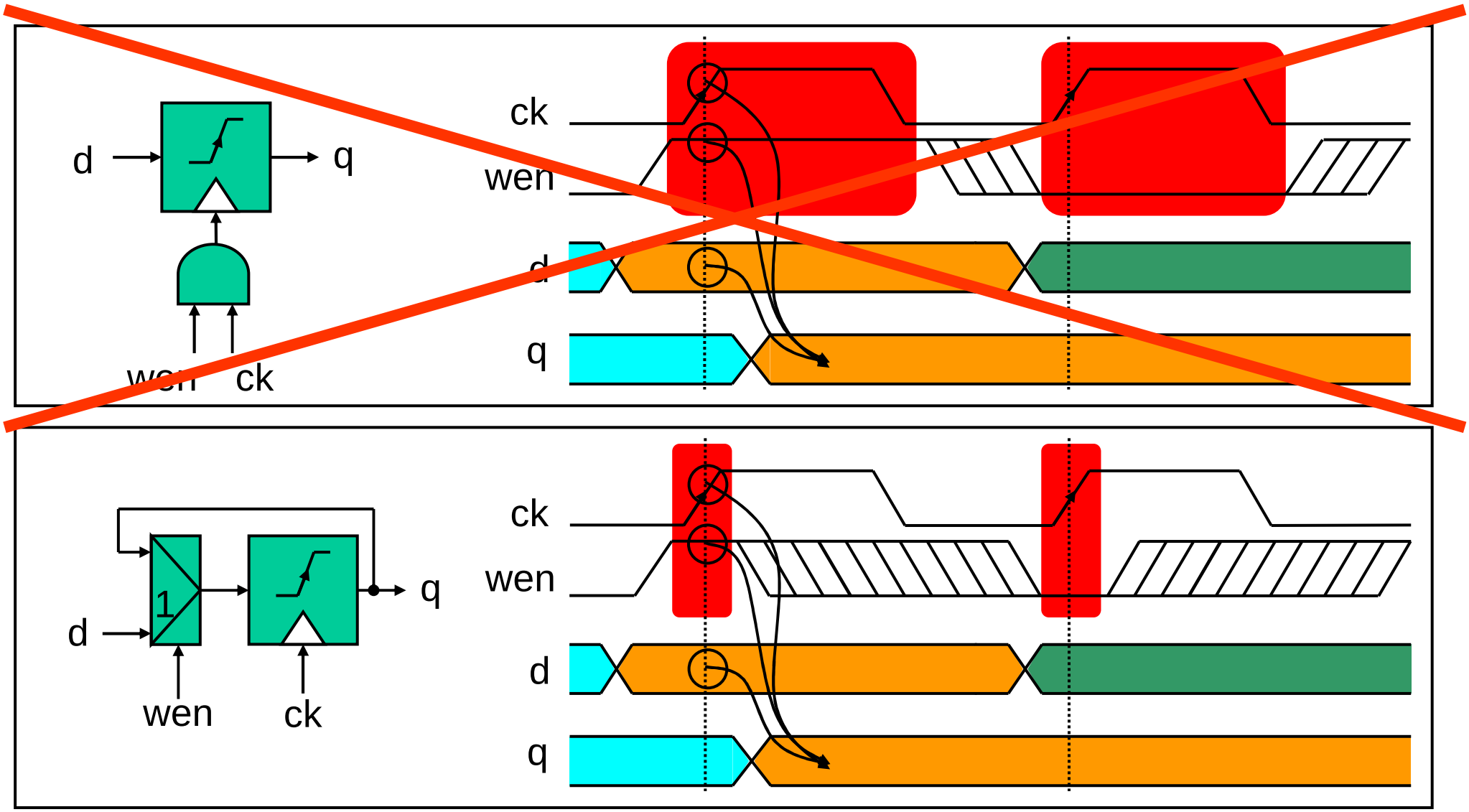
écriture conditionnelle : problème de stabilité de wen





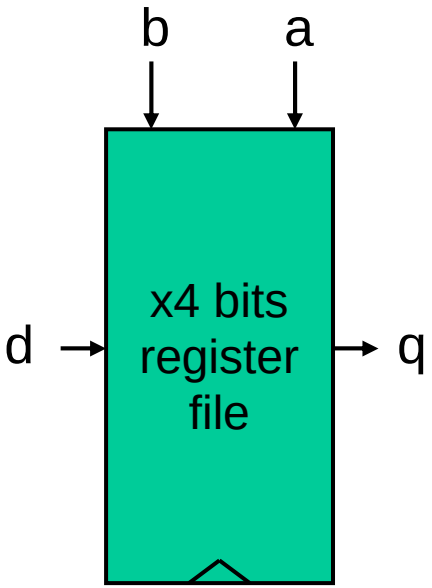
Bascule D

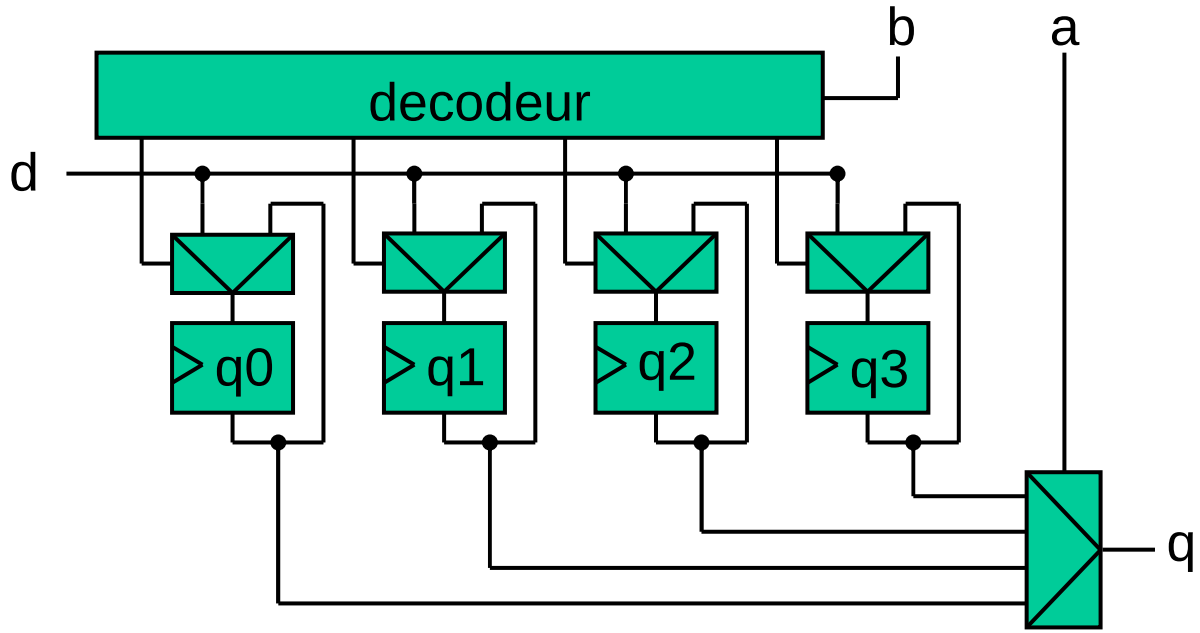
écriture conditionnelle : **JAMAIS DE LOGIQUE SUR L'HORLOGE**



Register File

banc de registres 1 lecture – 1 écriture



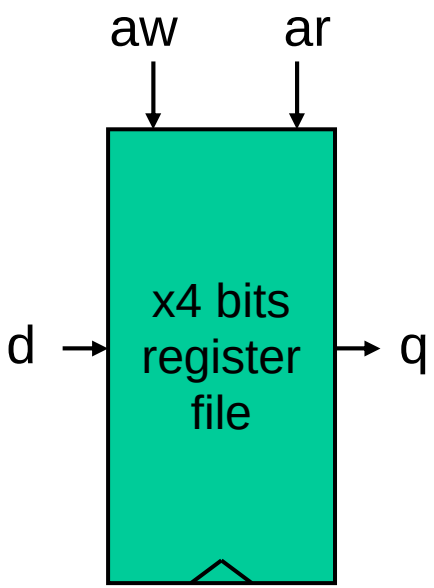


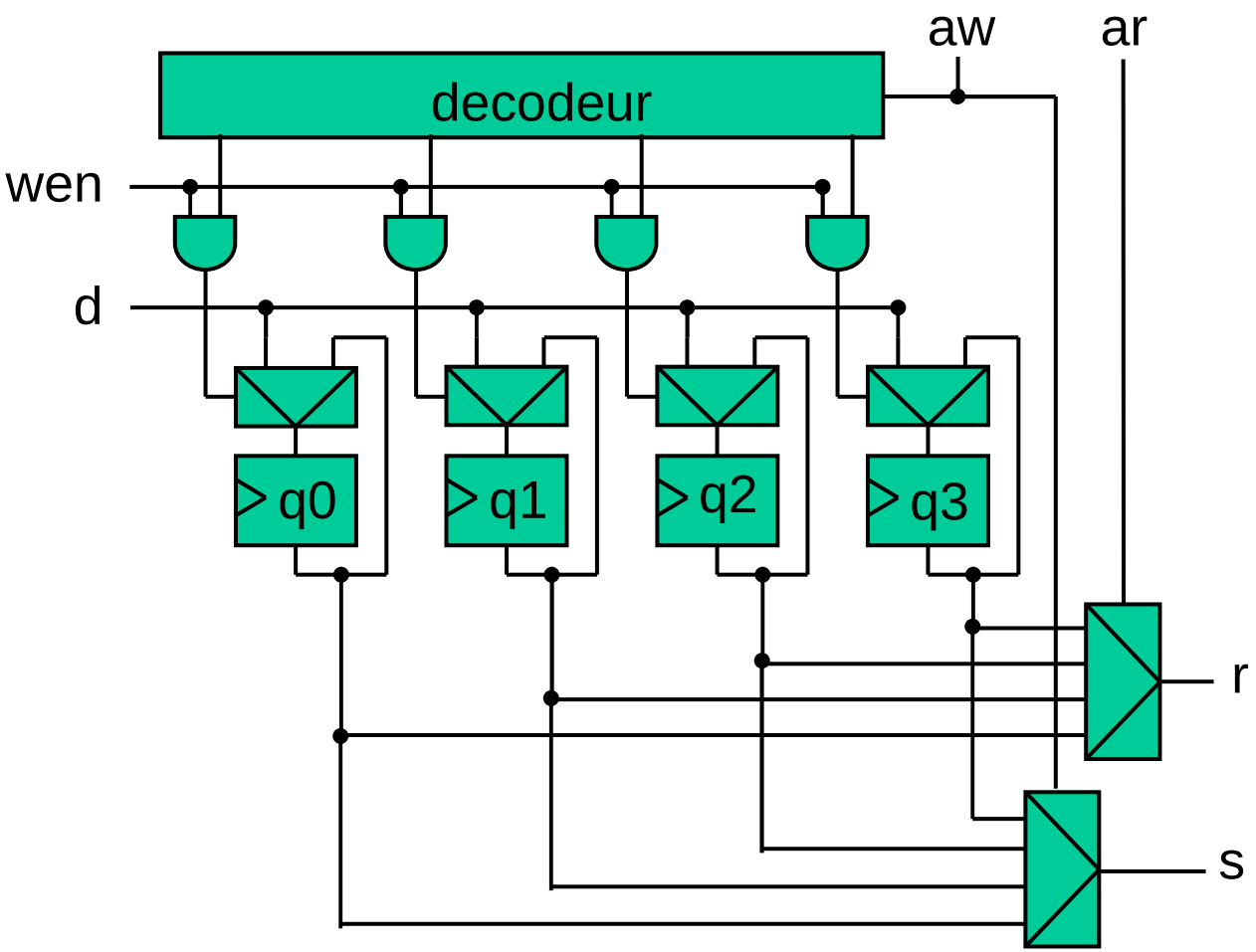

```
RF : block (ck='1' and ck'event)
begin
        q0 <= guarded d when aw=B"00" else q0;
        q1 <= guarded d when aw=B"01" else q1;
        q2 <= guarded d when aw=B"10" else q2;
        q3 <= guarded d when aw=B"11" else q3;
end block RF;
```

```
with ar select q <=
        q0 when B"00" else
        q1 when B"01" else
        q2 when B"10" else
        q3 ;
```

Register File

banc de registres 2 lectures – 1 écriture conditionnelle





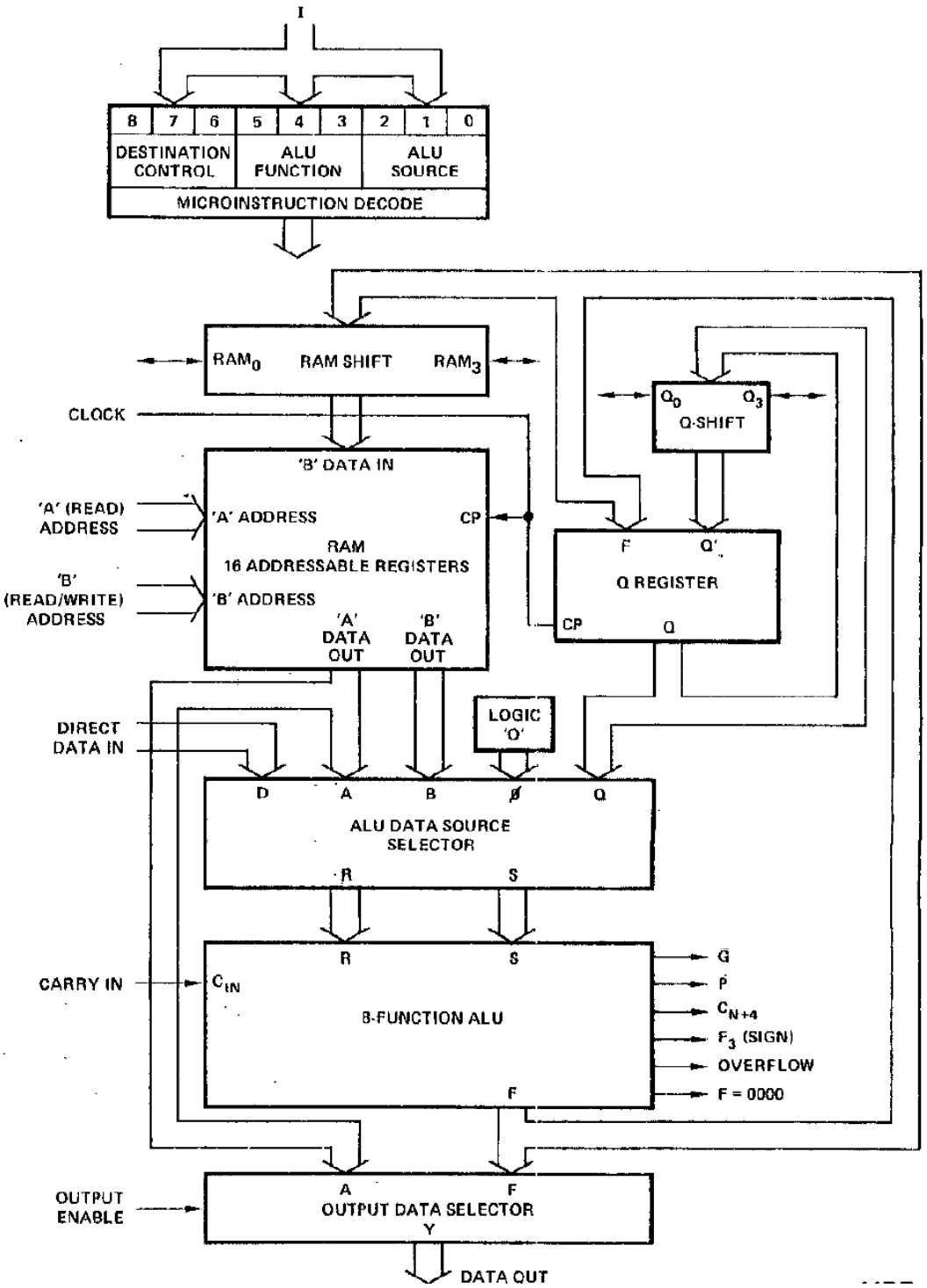
Documents fournis

- Spécification de l'AM2901
  - architecture interne
  - comportement possibles : instructions, assemblage
  - caractéristiques temporelles et électriques
- AM2901.vbe buggé
- jeux de patterns

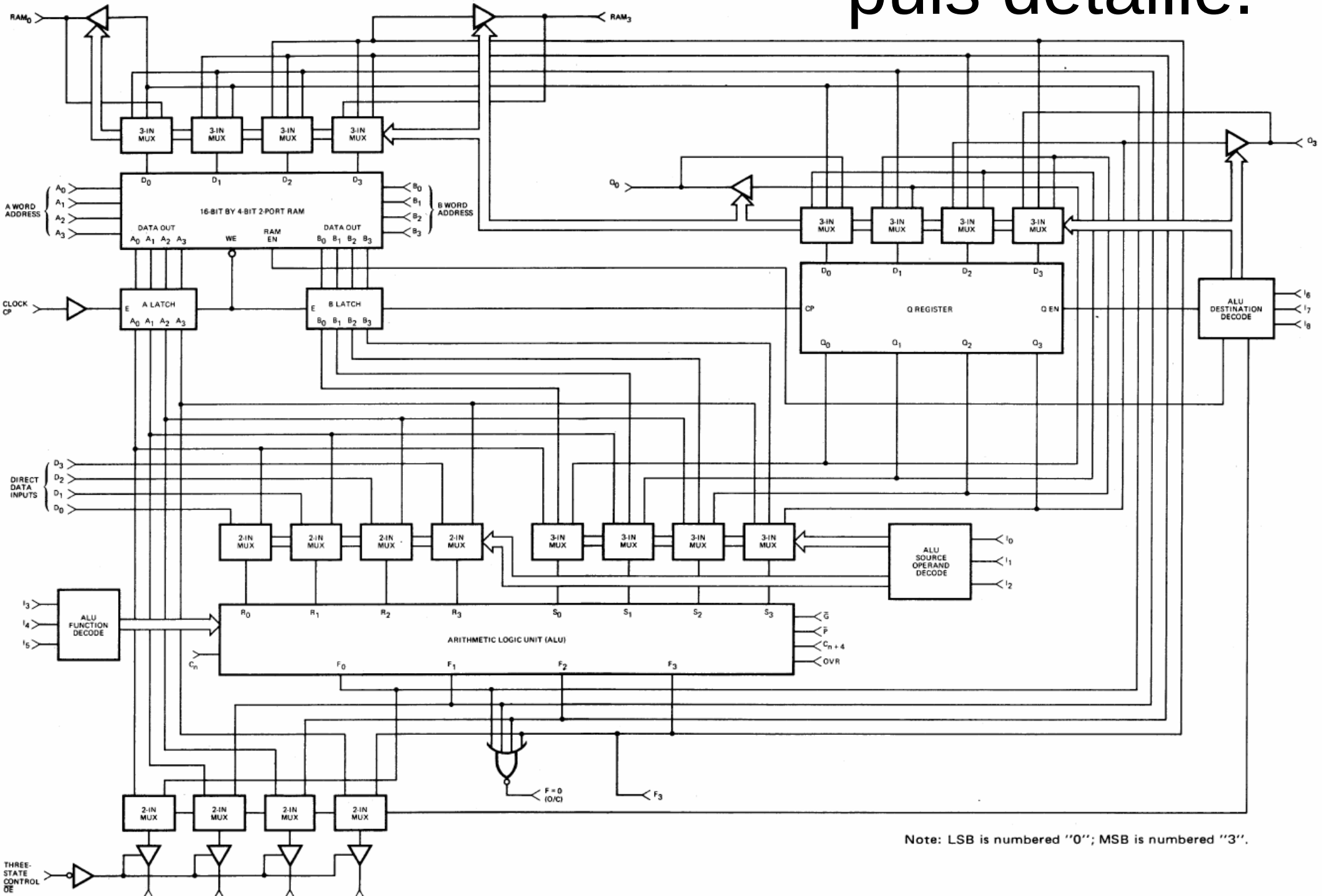
Schéma simplifié…

Remarques :

- Les noms de signaux et de connecteurs ne correspondent pas avec ceux du vbe.
- La décomposition en composants élémentaires n'est pas toujours identique avec celle du vbe



puis détaillé.



Note: LSB is numbered "0"; MSB is numbered "3".

une série de tables…

Les sources

	MICRO CODE				ALU SOURCE OPERANDS	
Mnemonic	I_2	I_1	I_0	Octal Code	R	S
AQ	L	L	L	0	A	Q
AB	L	L	H	1	A	B
ZQ	L	H	L	2	0	Q
ZB	L	H	H	3	0	B
ZA	H	L	L	4	0	A
DA	H	L	H	5	D	A
DQ	H	H	L	6	D	Q
DZ	H	H	H	7	D	0

Les destinations

	MICRO CODE				RAM FUNCTION		Q-REG. FUNCTION		Y OUTPUT	RAM SHIFTER		Q SHIFTER	
Mnemonic	I_8	I_7	I_6	Octal Code	Shift	Load	Shift	Load		RAM_0	RAM_3	Q_0	Q_3
QREG	L	L	L	0	X	NONE	NONE	F→Q	F	X	X	X	X
NOP	L	L	H	1	X	NONE	X	NONE	F	X	X	X	X
RAMA	L	H	L	2	NONE	F→B	X	NONE	A	X	X	X	X
RAMF	L	H	H	3	NONE	F→B	X	NONE	F	X	X	X	X
RAMQD	H	L	L	4	DOWN	F/2→B	DOWN	Q/2→Q	F	F_0	IN_3	Q_0	IN_3
RAMD	H	L	H	5	DOWN	F/2→B	X	NONE	F	F_0	IN_3	Q_0	X
RAMQU	H	H	L	6	UP	2F→B	UP	2Q→Q	F	IN_0	F_3	IN_0	Q_3
RAMU	H	H	H	7	UP	2F→B	X	NONE	F	IN_0	F_3	X	Q_3

Les instructions

Octal	C_n = 0 (Low)		C_n = 1 (High)	
I_{543}, I_{210}	Group	Function	Group	Function
0 0	ADD	A+Q	ADD plus one	A+Q+1
0 1		A+B		A+B+1
0 5		D+A		D+A+1
0 6		D+Q		D+Q+1
0 2	PASS	Q	Increment	Q+1
0 3		B		B+1
0 4		A		A+1
0 7		D		D+1
1 2	Decrement	Q−1	PASS	Q
1 3		B−1		B
1 4		A−1		A
2 7		D−1		D
2 2	1's Comp.	−Q−1	2's Comp. (Negate)	−Q
2 3		−B−1		−B
2 4		−A−1		−A
1 7		−D−1		−D
1 0	Subtract (1's Comp)	Q−A−1	Subtract (2's Comp)	Q−A
1 1		B−A−1		B−A
1 5		A−D−1		A−D
1 6		Q−D−1		Q−D
2 0		A−Q−1		A−Q
2 1		A−B−1		A−B
2 5		D−A−1		D−A
2 6		D−Q−1		D−Q

Les drapeaux

I_{543}	Function	$\overline{P}$	$\overline{G}$	C_{n+4}	OVR
0	R + S	$\overline{P_3P_2P_1P_0}$	$\overline{G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0}$	C_4	$C_3 \forall C_4$
1	S − R	← Same as R + S equations, but substitute $\overline{R_i}$ for R_i in definitions →			
2	R − S	← Same as R + S equations, but substitute $\overline{S_i}$ for S_i in definitions →			
3	$R \vee S$	LOW	$P_3P_2P_1P_0$	$\overline{P_3P_2P_1P_0} + C_n$	$\overline{P_3P_2P_1P_0} + C_n$
4	$R \wedge S$	LOW	$\overline{G_3 + G_2 + G_1 + G_0}$	$G_3 + G_2 + G_1 + G_0 + C_n$	$G_3 + G_2 + G_1 + G_0 + C_n$
5	$\overline{R} \wedge S$	LOW	← Same as $R \wedge S$ equations, but substitute $\overline{R_i}$ for R_i in definitions →		
6	$R \forall S$	← Same as $\overline{R \forall S}$, but substitute $\overline{R_i}$ for R_i in definitions →			
7	$\overline{R \forall S}$	$G_3 + G_2 + G_1 + G_0$	$G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1P_0$	$\overline{G_3 + P_3G_2 + P_3P_2G_1} + P_3P_2P_1P_0 (G_0 + \overline{C_n})$	See note

Note: $[\overline{P}_2 + \overline{G}_2\overline{P}_1 + \overline{G}_2\overline{G}_1\overline{P}_0 + \overline{G}_2\overline{G}_1\overline{G}_0C_n] \forall [\overline{P}_3 + \overline{G}_3\overline{P}_2 + \overline{G}_3\overline{G}_2\overline{P}_1 + \overline{G}_3\overline{G}_2\overline{G}_1\overline{P}_0 + \overline{G}_3\overline{G}_2\overline{G}_1\overline{G}_0C_n]$ + = OR

Objectifs

- Comprendre la description vbe
- Trouver le(s) bug(s)

mais
ni copie, ni diff ça n'a aucun intérêt.