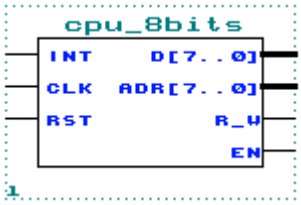
MICROPROCESSEURS
IMPLANTATION DANS UNE EPLD ALTERA

1. Objectifs

Il s'agit d'étudier et de valider un symbole pour le logiciel Maxplus II d'Altera permettant d'implanter une unité centrale à microprocesseur simple dans une EPLD de cette marque.

On utilise une structure "Von Neumann", les mémoires "programme" et "données" étant externes.

2. Spécifications de l'U.C.



- **Nom du symbole** : "CPU_8bits"
- **Bus de données** : 8 bits
- **Bus d'adresse** : 8 bits
- **Signaux de contrôle** :
  - **RST** : mise à zéro du compteur ordinal PC.
  - **INT** : demande d'interruption (état "1" actif)
  - **R_W** : indique le sens du transfert ("0" → écriture, "1" → lecture)
  - **E** : validation de l'échange
- **Horloge CLK** : 0Hz à 20MHz. Suivant leurs complexités, les instructions s'exécutent en 2 à 4 cycles d'horloge.
- **Modèle de programmation**
  - Accumulateur A sur 8 bits
  - Compteur ordinal PC sur 8 bits
  - Pointeur de pile SP sur 8 bits
  - Registre d'états : Z : "Zéro"
    N : "Négatif"
    C : "Carry"

7	6	5	4	3	2	1	0	
7	6	5	4	3	2	1	0	A
7	6	5	4	3	2	1	0	PC
7	6	5	4	3	2	1	0	SP
C	N	Z						CCR

- **Vecteurs de Reset et d'interruption**
  - Reset : saut à l'adresse 00_H
  - Interruption : saut à l'adresse 02_H
- **Jeu d'instructions**

Mnémonique		Codage 1 ou 2 octets		Description	
LDA	#val	00_H	val	A ← val PC ← PC + 2	Affectation accumulateur A en mode immédiat
LDA	adr	01_H	adr	A ← (adr) PC ← PC + 2	Affectation accumulateur A avec le contenu de l'adresse "adr"
STA	adr	02_H	adr	(adr) ← A PC ← PC + 2	Affectation du contenu de l'adresse "adr" avec l'accu A
INCA		03_H		A ← A + 1 PC ← PC + 1	Incrémentation accu A.
DECA		04_H		A ← A – 1 PC ← PC + 1	Décrémentation accu A
ADD	#val	05_H	val	A ← A + val PC ← PC + 2	Addition accumulateur A en mode immédiat. Résultat dans accu A.
ADD	adr	06_H	adr	A ← A + (adr) PC ← PC + 2	Addition accumulateur A avec le contenu de l'adresse "adr" Résultat dans accu A.
SUB	#val	07_H	val	A ← A – val PC ← PC + 2	Soustraction accumulateur A en mode immédiat. Résultat dans accu A.
SUB	adr	08_H	adr	A ← A – (adr) PC ← PC + 2	Soustraction accumulateur A avec le contenu de l'adresse "adr". Résultat dans accu A.
AND	#val	09_H	val	A ← A AND val PC ← PC + 2	ET logique avec accumulateur A en mode immédiat. Résultat dans accu A.
AND	adr	$0A_H$	val	A ← A AND (adr) PC ← PC + 2	ET logique accumulateur A avec le contenu de l'adresse "adr". Résultat dans accu A.
OR	#val	$0B_H$	val	A ← A OR val PC ← PC + 2	OU logique avec accumulateur A en mode immédiat. Résultat dans accu A.

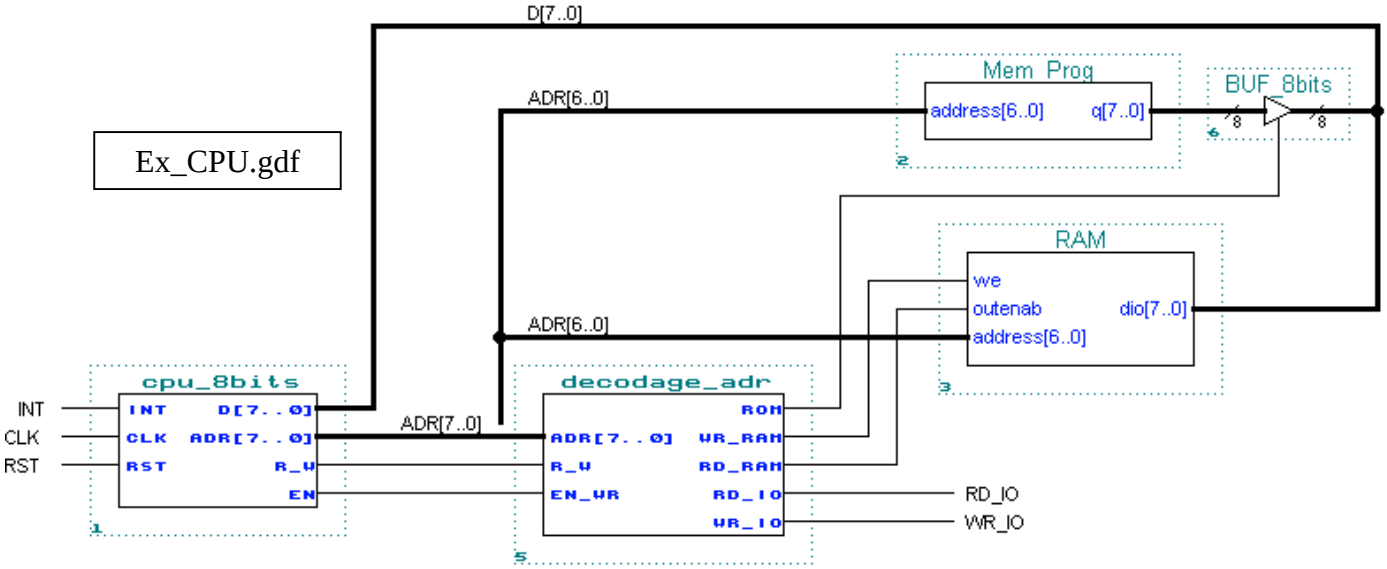
OR	adr	$0C_H$	val	A ← A OR (adr) PC ← PC + 2	OU logique accumulateur A avec le contenu de l'adresse "adr". Résultat dans accu A.
XOR	#val	$0D_H$	val	A ← A XOR val PC ← PC + 2	OU exclusif avec accumulateur A en mode immédiat. Résultat dans accu A.
XOR	adr	$0E_H$	val	A ← A XOR (adr) PC ← PC + 2	OU exclusif accumulateur A avec le contenu de l'adresse "adr". Résultat dans accu A.
CLR		$0F_H$		A ← 00_H PC ← PC + 1	Remise à zéro accumulateur A
LSL		10_H		A ← A + A PC ← PC + 1	Décalage à gauche accumulateur A
LSR		11_H		A ← A DIV 2 PC ← PC + 1	Décalage à droite accumulateur A
CMP	#val	12_H	#val	Z, N ← A – val PC ← PC + 2	Comparer accumulateur A en mode immédiat. Affectation des indicateurs Z et N.
CMP	adr	13_H	adr	Z, N ← A – (adr) PC ← PC + 2	Comparer accu A avec le contenu de l'adresse "adr". Affectation des indicateurs Z et N.
PSHA		14_H		SP ← SP – 1 (SP) ← A	Sauvegarde de l'accu A dans la pile
PULA		15_H		A ← (SP) SP ← SP + 1	Restitution de l'accu A depuis la pile
JMP	adr	16_H	adr	PC ← (adr)	Saut à l'adresse "adr".
JEQ	adr	17_H	adr	Z=1: PC ← (adr) Z=0: PC ← PC + 2	Si Z=1 alors saut à l'adresse "adr" sinon instruction suivante.
JNE	adr	18_H	adr	Z=0: PC ← (adr) Z=1: PC ← PC + 2	Si Z=0 alors saut à l'adresse "adr" sinon instruction suivante.
JMI	adr	19_H	adr	N=1: PC ← (adr) N=0: PC ← PC+2	Si N=1 alors saut à l'adresse "adr" sinon instruction suivante.
JPL	adr	$1A_H$	adr	N=0: PC ← (adr) N=1: PC ← PC+2	Si N=0 alors saut à l'adresse "adr" sinon instruction suivante.
JCS	adr	$1B_H$	adr	C=1: PC ← (adr) C=0: PC ← PC+2	Si C=1 alors saut à l'adresse "adr" sinon instruction suivante.
JCC	adr	$1C_H$	adr	C=0: PC ← (adr) C=1: PC ← PC+2	Si C=0 alors saut à l'adresse "adr" sinon instruction suivante.
JSR	adr	$1D_H$	adr	SP ← SP – 1 (SP) ← PC + 2 PC ← (adr)	Appel de sous-programme : saut à l'adresse "adr" et sauvegarde de l'adresse de retour dans la pile.
RTS		$1E_H$		PC ← (SP) SP ← SP + 1	Retour de sous programme : restitution de l'adresse de retour depuis la pile
INT		$1F_H$		SP ← SP – 1 (SP) ← CCR SP ← SP – 1 (SP) ← PC PC ← 02_H	Appel de sous-programme de traitement d'interruption. Celui-ci commence à l'adresse 02_H. Utilisée par l'entrée INT. Ne pas inclure dans les programmes.
RTI		20_H		PC ← (SP) SP ← SP + 1 CCR ← (SP)	Retour d'interruption
SEI		21_H		EN_INT ← "1" PC ← PC + 1	Mise à "1" de la bascule de validation d'interruption
CLI		22_H		EN_INT ← "0" PC ← PC + 1	Mise à "0" de la bascule de validation d'interruption
CLR	Adr	23_H	Adr	(Adr) ← 0 PC ← PC + 2	Mise à zéro de la variable (Adr)
DEC	Adr	24_H	Adr	(Adr) ← (Adr) - 1 PC ← PC + 2	Décrémentation de la variable (Adr)
INC	Adr	25_H	Adr	(Adr) ← (Adr) + 1 PC ← PC + 2	Incrémentation de la variable (Adr)
LDX	#Val	26_H	Adr	X ← val PC ← PC + 2	Affectation registre X en mode immédiat

LDX	adr	27_H	adr	X ← (adr) PC ← PC + 2	Affectation registre X avec le contenu de l'adresse "adr"
INX		28_H		X ← X + 1 PC ← PC + 1	Incrémentation registre X.
DEX		29_H		X ← X – 1 PC ← PC + 1	Décrémentation registre X.
LDA	0,X	$2A_H$		A ← (X) PC ← PC + 1	Affectation accumulateur A avec le contenu de l'adresse X
TAX		$2B_H$		X ← A PC ← PC + 1	Transfert accu A vers registre X

3. Exemple d'utilisation

Voici la structure d'un petit ordinateur utilisant les composants suivants :

- CPU_8BITS : l'UC objet de ce document
- Mem_PROG : mémoire "programme" de 128 octets, non équipé d'une sortie "3 états"
- RAM : mémoire vive de 128 octets. Ecriture asynchrone avec "we" et validation du buffer 3 états avec "outenab"
- Decodage_adr : fonction de logique combinatoire assurant le décodage d'adresse



Decodage_adr.tdf

```
-- Table de décodage :
--    ROM programme : 00H à 7FH en lecture
--    RAM           : 81H à FFH en lecture et écriture
--    Port IO       : 80H en lecture et écriture
subdesign Decodage_Adr
(
  ADR[7..0] : input;   -- Bus d'adresse
  R_W       : input;
  EN_WR     : input;   -- Validation d'écriture asynchrone en RAM
  ROM       : output;  -- Controle le buffer 3 états de la ROM "programme"
  WR_RAM    : output;  -- Ecriture asynchrone en RAM
  RD_RAM    : output;  -- Controle le buffer 3 états de la RAM
  RD_IO     : output;  -- Validation de lecture d'un port 8 bits
  WR_IO     : output;  -- Validation d'écriture d'un port 8 bits
)
variable IO : node;
begin
 IO  = ADR[7..0]==H"80";
 ROM = !ADR7 AND R_W;
 RD_IO  =  R_W AND IO;
 WR_IO  = !R_W AND IO;
 RD_RAM = ADR7 AND !IO AND R_W;
 WR_RAM = EN_WR AND ADR7 AND !IO AND !R_W;
end;
```

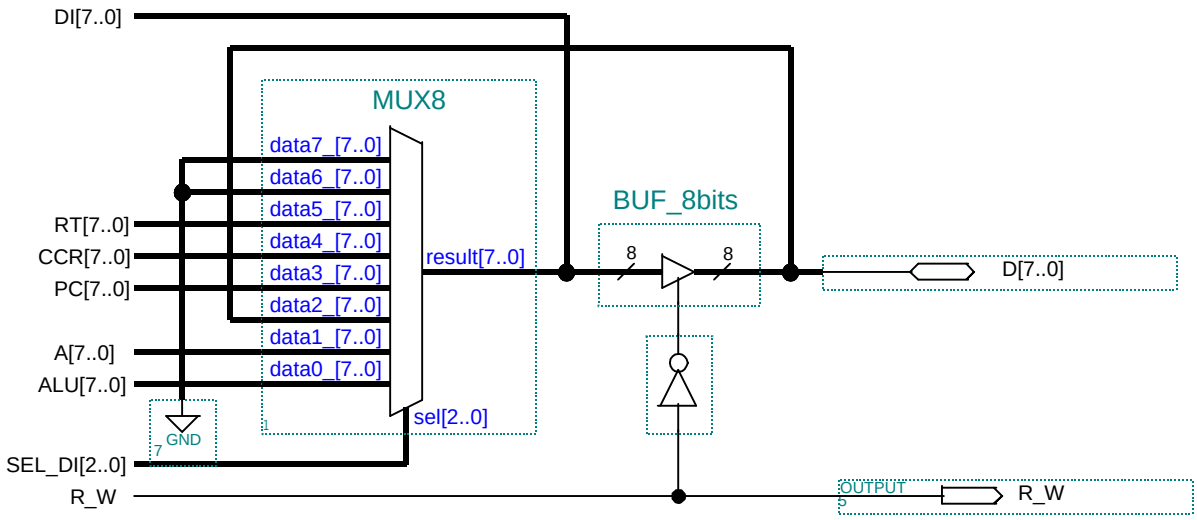
Notes :

- Les composants entourant l'UC dans cet exemple ne sont en fait pas externes à l'EPLD. On a utilisé ici, pour faciliter les tests, un modèle permettant d'intégrer des mémoires (mortes et vives). Mais dans d'autres cas, rien n'empêche d'utiliser des ROM et RAM externe.
- Le port IO n'est pas exploité. Il fera l'objet d'un exercice. L'analyse du décodage d'adresse montre qu'il fait perdre un octet dans la RAM sur les 128 disponibles.

4. Structure interne

Par principe, toutes les structures séquentielles de la CPU sont synchrones de l'horloge CLK. On optimise ainsi l'implantation dans l'EPLD (on utilise pour CLK une entrée "Global Clock") et on limite les aléas dus aux temps de propagation.

4.1 Bus de données internes



Description des signaux :

- D[7..0] : bus de données externes
- DI[7..0] : bus de données internes
- PC[7..0] : état du compteur ordinal
- A[7..0] : état de l'accumulateur A
- RT[7..0] : état du registre temporaire
- ALU[7..0] : résultat de l'opération exécutée par l'ALU (Unité Arithmétique et Logique)
- CCR[7..0] : registre de codes conditions (en fait C, N et Z mémorisés après calcul)
- SEL_DI[2..0] : sélection de la source du bus de données internes. Cette information est produite par le séquenceur d'instruction (voir plus loin)

Le bus de données externes D[7..0] est bidirectionnel (lecture et écriture). Il est donc séparé du bus de données internes par un buffer "3 états" contrôlé par le signal R_W .

Le multiplexeur MUX8 (8 entrées et une sortie sur 8 bits) permet de placer sur le bus de données internes l'information nécessaire à l'exécution de l'instruction en cours.

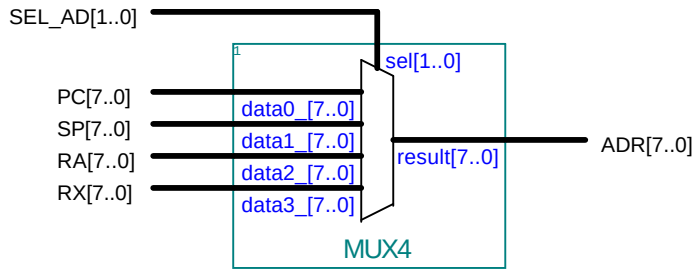
Les cas suivants peuvent se présenter pendant l'exécution d'une instruction :

- Lecture d'une mémoire externe → R_W = "1" ⇒ le buffer "3 états" est en haute impédance
  → SEL_DI[2..0] = 2 ⇒ **DI[7.. 0]** ← **D[7.. 0]**
- Ecriture vers la mémoire externe → R_W = "0" ⇒ le buffer "3 états" est validé : **D[7.. 0]** ← **DI[7.. 0]**
  → Le signal SEL_DI[2..0] permet de placer sur le bus de données externes l'information à écrire (A[7..0], RT[7..0], CCR[7..0], PC[7..0] ou 00_H).
- Opérations internes (pas d'accès aux mémoires externes) :
  - Ecriture résultat ALU dans accu A → R_W = "1" par prudence
    → SEL_DI[2..0] = 0 ⇒ **DI[7.. 0]** ← **ALU[7.. 0]**
  - Ecriture état accu A dans registre X → R_W = "1" par prudence
    → SEL_DI[2..0] = 1 ⇒ **DI[7.. 0]** ← **A[7.. 0]**

4.2 Bus d'adresse externe ADR[7..0]

Le bus d'adresse externe permet d'accéder à chaque octet des mémoires "programme" et "données" ainsi qu'aux composants périphériques ajoutés en externe. Ces accès peuvent être de différents types :

- Recherche du code opération et de son éventuel opérande : **ADR[7.. 0] = PC[7.. 0]**
- Empilage et dépilage : **ADR[7.. 0] = SP[7.. 0]**
- Adressage direct des données : **ADR[7.. 0] = RA[7.. 0]**
- Adressage indexé : **ADR[7.. 0] = RX[7.. 0]**



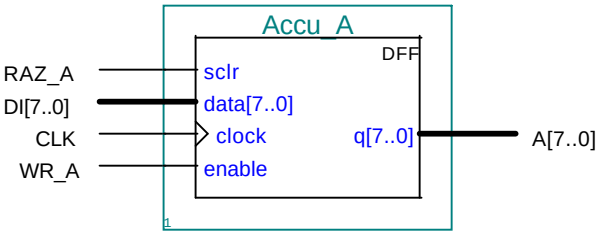
Description des signaux :

- PC[7..0] : état du compteur ordinal
- SP[7..0] : état du pointeur de pile
- RA[7..0] : état du registre d'adresse temporaire
- RX[7..0] : registre index. Utilisé pour l'adressage indexé.

4.3 Accumulateur A

Il s'agit d'un simple registre de 8 bascules D synchrones possédant 2 entrées de contrôle "enable" et "sclr" :

"enable" : "0" → les 8 bascules **restent dans leurs états**
"1" → "sclr" = "1" ⇒ **q[7.. 0]** ← **0** au ↑ de "clock"
→ "sclr" = "0" ⇒ **q[7.. 0]** ← **data[7.. 0]** au ↑ de "clock"



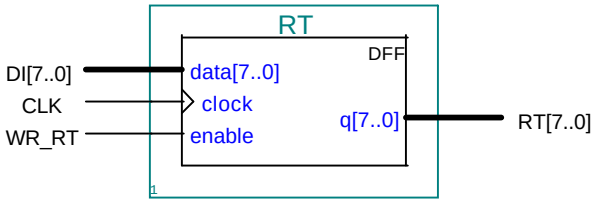
Description des signaux :

- CLK : horloge synchrone
- DI[7..0] : bus de données internes
- RAZ_A : mise à zéro de l'accu A. Signal produit par le séquenceur d'instructions
- WR_A : affectation de l'accu A avec DI. Signal produit par le séquenceur d'instructions
- A[7..0] : état de l'accu A. Appliqué sur l'entrée "A" de l'ALU (pour d'éventuelles opérations) et sur l'entrée "1" du multiplexeur MUX8 de DI (voir §4.1)

4.4 Registre temporaire RT

Ce registre est utilisé par certaines instructions pour mémoriser des informations temporaires.
Il s'agit d'un simple registre de 8 bascules D complétées avec une entrée d'affectation synchrone :

"enable" : "0" → les 8 bascules **restent dans leurs états**
"1" → **q[7.. 0]** ← **data[7.. 0]** au ↑ de "clock"

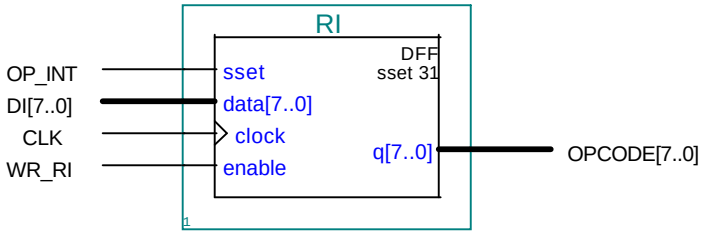


Description des signaux :

- CLK : horloge synchrone
- DI[7..0] : bus de données internes
- WR_RT : affectation du registre avec DI. Signal produit par le séquenceur d'instructions
- RT[7..0] : état du registre temporaire. Appliqué sur l'entrée "B" de l'ALU (pour certaines opérations) et à l'entrée 5 du multiplexeur MUX8 pendant certaines instructions.

4.5 Registre d'instruction RI

Ce registre mémorise le code opération de l'instruction en cours pendant toute la durée de son exécution.
Il s'agit du même type de registre que celui utilisé pour réaliser RT.



Description des signaux :

- CLK : horloge synchrone
- DI[7..0] : bus de données internes
- WR_RI : affectation du registre avec DI. Signal produit par le séquenceur d'instructions
- OP_INT : affectation synchrone du code opération INT ($1F_H = 31_{10}$) à la prise en compte d'une demande d'interruption
- OPCODE[7..0] : état du registre d'instruction. Utilisé par le séquenceur d'instructions.

4.6 Compteur ordinal PC

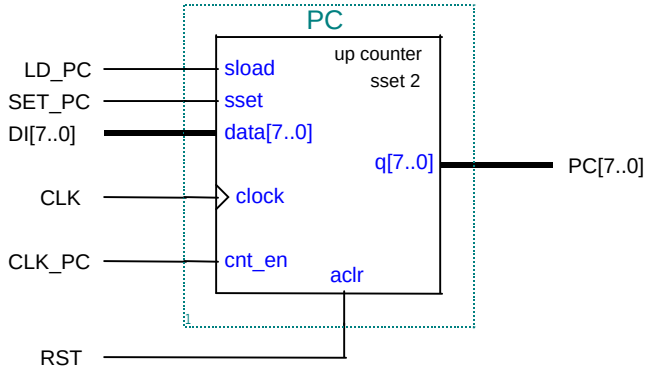
Le compteur ordinal (ou compteur programme) produit l'adresse dans la mémoire programme de l'instruction à exécuter. Les instructions sont placées dans l'ordre des adresses croissantes, ce qui justifie l'utilisation d'un compteur qu'il suffit d'incrémenter pour pointer l'instruction suivante.

Pour les instructions de saut (rupture du séquencement naturel des instructions), on utilise les entrées de prépositionnement du compteur :

- Raz pour le branchement à l'adresse 00_H suite à un "Reset",
- Affectation à la valeur 2 pour réaliser le branchement à l'adresse 02_H à la prise en compte d'une interruption,
- Affectation à une valeur lue depuis la mémoire programme pour réaliser un branchement à une adresse quelconque (instructions JMP, JSR, JEQ, JNE, …).

Il s'agit donc d'un classique compteur 8 bits synchrone à prépositionnements synchrones et raz asynchrone :

- "clock" : horloge synchrone
- "cnt_en" : validation de l'incrémentation
- "data[7..0]" : données de prépositionnement
- "sload" : validation du prépositionnement synchrone
- "aclr" : raz asynchrone



Description des signaux :

- CLK : horloge synchrone
- DI[7..0] : bus de données internes
- RST : raz asynchrone du compteur. Activé au "reset" du microprocesseur.
- CLK_PC : validation de l'incrémentation synchrone
- LD_PC : validation du prépositionnement synchrone
- SET_PC : validation du prépositionnement synchrone à 02_H, adresse du programme d'interruption
- PC[7..0] : état du compteur ordinal. Appliqué généralement sur le bus d'adresse externe via le multiplexeur MUX4 (§4.2).

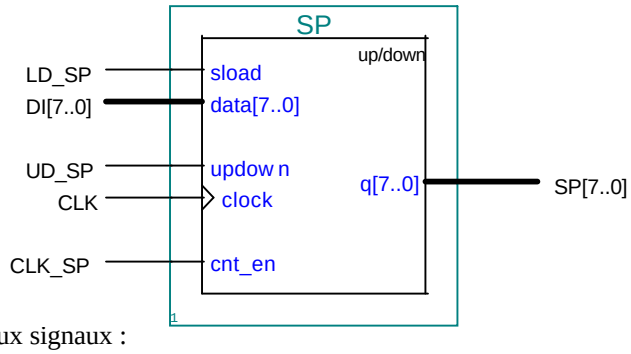
4.7 Pointeur de pile SP (Stack Pointer)

Cette fonction est utilisée par les couples d'instructions JSR-RTS et INT-RTI pour sauvegarder dans la RAM (à l'appel) puis restituer (au retour) le contexte du CPU (adresse de retour et registre CCR pour l'interruption).

La pile du CPU est placée dans les derniers octets de la RAM du système. Le pointeur de pile fournit l'adresse dernière case mémoire occupée : il doit donc être **décrémenté avant** l'empilage et **incrémenté après** le dépilage.

On utilise ici un compteur-décompteur synchrone prépositionnable analogue au compteur PC et équipé d'une entrée de sélection de mode :

- "updown" : sélection comptage ("0") ou décomptage ("1").

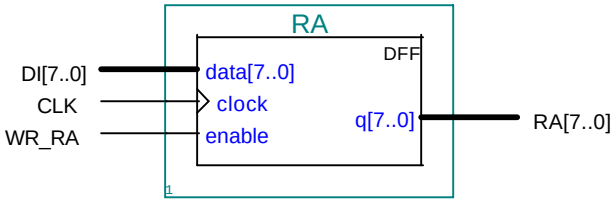


Description des nouveaux signaux :

- UD_SP : sélection comptage ("0") ou décomptage ("1").
- CLK_SP : validation de l'incrémentation ou de la décrémentation synchrone
- LD_SP : validation du prépositionnement synchrone
- SP[7..0] : état du pointeur de pile. Appliqué sur le bus d'adresse externe via le multiplexeur MUX3 (voir §4.2) pendant les phases d'empilage ou de dépilage.

4.8 Registre d'adresse RA

Ce registre mémorise une adresse temporaire nécessaire à l'exécution de certaines instructions.
Il s'agit du même type de registre que celui utilisé pour réaliser RT.

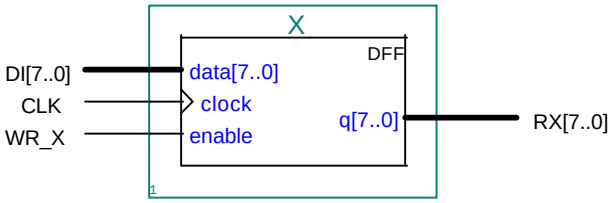


Description des nouveaux signaux :

- WR_RA : affectation du registre avec DI. Signal produit par le séquenceur d'instructions
- RA[7..0] : état du registre d'adresse temporaire. Appliqué sur le bus d'adresse externe via le multiplexeur MUX3 (voir §4.2) au cours de certaines instructions

4.9 Registre d'index X

Ce registre permet l'adresse indexé utilisé par certaines instructions. L'ALU décrite plus loin peut opérer des incrémentations ou des décrémentations du contenu de ce registre.

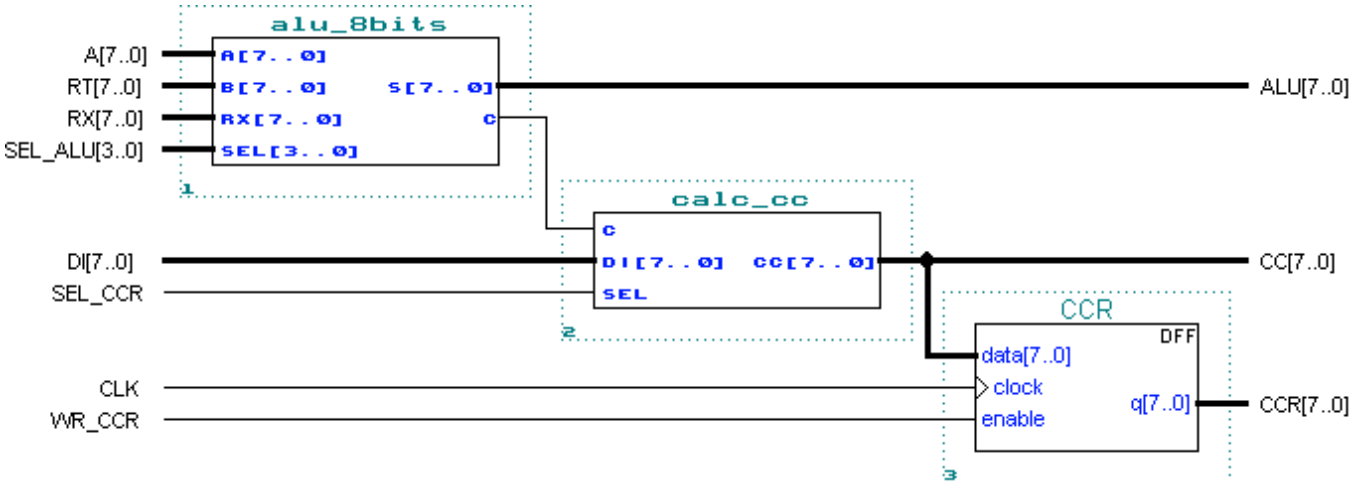


Description des nouveaux signaux :

- WR_RX : validation de l'affectation synchrone de ce registre : RX[7..0] ← DI[7..0]
- RX[7..0] : l'état du registre d'index peut être placé sur le bus d'adresse via le multiplexeur MUX4

4.10 Unité arithmétique et logique : ALU

Cette fonction réalise les opérations arithmétiques et logiques nécessaires à l'exécution des instructions.



La fonction "CALC_CC" calcule les indicateurs C' (Carry), Z' (Zéro) et N' (Négatif) de la façon suivante :

- SEL = "0" : C' = CC7 = retenue de l'ALU
  N' = CC6 = DI7 (Négatif en C2)
  Z' = CC5 = indicateur de DI[7..0]=0
- SEL = "1" : CC[7..0] ← DI[7..0]. Utilisé pendant RTI pour restituer le registre d'état

La fonction "CCR" est le registre mémoire des indicateurs dont l'écriture est validée par WR_CCR.

Les indicateurs CCR7 (**C**), CCR6 (**N**) et CCR5 (**Z**) sont utilisés par le séquenceur d'instruction pour réaliser les sauts conditionnels (JEQ, JNE, …).

Tableau des opérations de l'ALU 8bits :

SEL[3..0]	S[7..0]	C
1	**NOT** A[7..0]	0
2	A[7..0] **AND** B[7..0]	0
3	A[7..0] **OR** B[7..0]	0
4	A[7..0] **XOR** B[7..0]	0
5	RX[7..0] + 1	0
6	RX[7..0] – 1	0
9	A[7..0] + 1	X
10	A[7..0] – 1	X
11	A[7..0] + B[7..0]	X
12	A[7..0] – B[7..0]	X
13	B[7..0] + 1	X
14	B[7..0] – 1	X
15	Décalage à droite	X

Toutes ces opérations de l'ALU sont réalisées par des structures combinatoires comme le montre la description AHDL suivante:

ALU_8bits.tdf

```
subdesign ALU_8bits
(
 A[7..0]  : input;  -- Opérande A
 B[7..0]  : input;  -- Opérande B
 RX[7..0] : input;  -- Registre X
 SEL[3..0]: input;  -- Sélection de l'opération
 S[7..0]  : output; -- Sortie
 C        : output; -- Retenue
)
variable AE[8..0], BE[8..0],SE[8..0] : NODE;
begin
 defaults
  SE8=gnd;
 end defaults;
```

```
AE[7..0]=A[7..0]; AE8=gnd; BE[7..0]=B[7..0]; BE8=gnd;
case SEL[] is
 when 1  => SE[7..0] =!A[]         ; -- Complémentation
 when 2  => SE[7..0] = A[] and B[] ; -- ET
 when 3  => SE[7..0] = A[] or  B[] ; -- OU
 when 4  => SE[7..0] = A[] xor B[] ; -- OU exclusif
 when 5  => SE[7..0] = RX[] +  1   ; -- Incrémentation X
 when 6  => SE[7..0] = RX[] -  1   ; -- Décrémentation X
 when 9  => SE[]     = AE[] +  1   ; -- Incrémentation A
 when 10 => SE[]     = AE[] -  1   ; -- Décrémentation A
 when 11 => SE[]     = AE[] +  BE[]; -- Addition
 when 12 => SE[]     = AE[] -  BE[]; -- Soustraction
 when 13 => SE[]     = BE[] +  1   ; -- Incrémentation B
 when 14 => SE[]     = BE[] -  1   ; -- Décrémentation B
 when 15 => SE[6..0] = A[7..1];      -- \
            SE7=gnd;                 --  > Décalage à droite
            SE8=A0;                  -- /
end case;
S[7..0]=SE[7..0];
C=SE8; end;
```

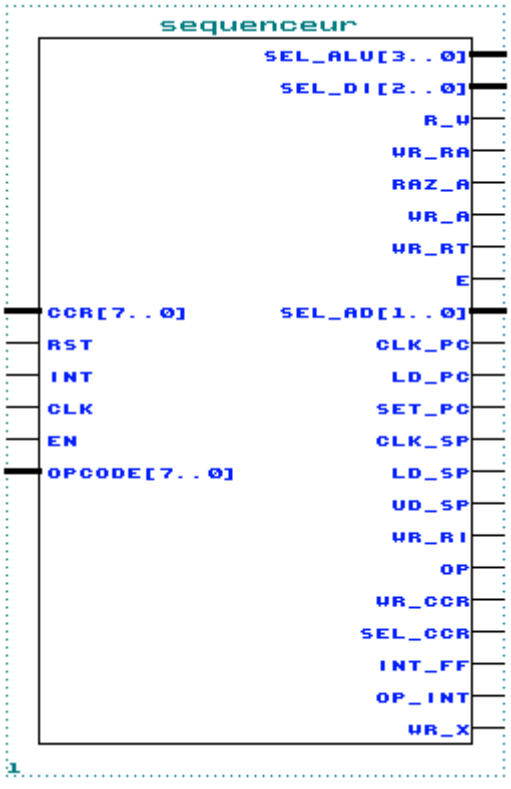
La seule remarque concerne la déclaration des variables AE[8..0], BE[8..0] et SE[8..0]. En effet, elles sont définies sur 9 bits pour produire la retenue C de poids 2^8.

CALC_CC.tdf

```
subdesign CALC_CC
 (
  C : input;          -- Retenue
  DI[7..0] : input;   -- Bus de données internes
  SEL      : input;   -- Sélection source : "0" -> normal
                      --                    "1" -> DI[] (pour restitution RTI)
  CC[7..0] : output; -- C=CC7, N=CC6, Z=CC5
 )
begin
 if SEL then CC[7..0]=DI[7..0];
        else CC7=C; CC6=DI7; CC5=DI[]==0; CC[4..0]=gnd;
 end if;
end;
```

4.11 Séquenceur d'instruction

On aborde maintenant la fonction la plus complexe du module "microprocesseur". Comme son nom l'indique, elle séquence les opérations nécessaires à l'exécution de chaque instruction.



Cette fonction produit tous les signaux de contrôle déjà cités partir de :

- CLK : horloge principale
- RST : "reset" du CPU
- OPCODE[7..0] : le code opération de l'instruction à exécuter. Il est mémorisé dans le registre RI.
- C, N, Z : indicateurs mémorisés sur le résultat de l'ALU (Carry, Négatif et Zéro) dans CCR[7..0]
- INT : demande d'interruption

Le signal EN est un signal de validation d'horloge permettant d'augmenter la durée des cycles sans réduire l'horloge. Il n'est pas utilisé dans les descriptions du document. Pour en tenir compte, il suffira de remplacer toutes les affectations à Vcc par EN.

Compte tenu de la structure adoptée pour la CPU, aucune instruction ne peut être réalisée en un seul cycle d'horloge. Suivant leurs complexités, elles nécessitent 2 ou 4 cycles. Pour ce faire, on réalise une "machine d'états" synchrone à 2 ou 4 états suivant les instructions à exécuter. Chaque état (S1 à S4) est décodé pour chaque instruction pour séquencer les opérations nécessaires à son exécution.

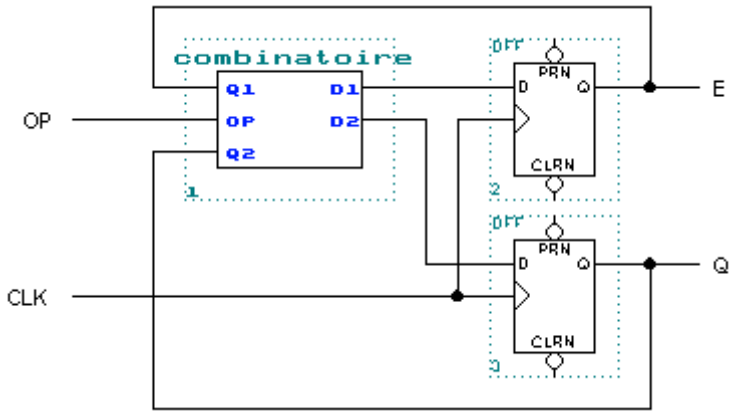
4.11.1 Machine d'état : SM

Elle est définie par une table d'états qui se succèdent à chaque flanc montant du signal d'horloge CLK. Le signal OP est positionné suivant le nombre de cycles nécessaires à l'exécution de l'instruction (2 ou 4) :

OP="0" : S1 → S2 → S1 → S2 → S1 → S2 …

OP="1" : S1 → S2 → S3 → S4 → S1 → S2 → S3 → S4 …

OP="0"			OP="1"		
Etat	**E**	**Q**	**Etat**	**E**	**Q**
S1	1	0	S1	1	0
S2	0	0	S2	0	0
			S3	1	1
			S4	0	1



L'implantation de machines d'états est prévue en AHDL. Il est donc inutile de concevoir la fonction de logique combinatoire, le compilateur d'Altera s'en charge. Il suffit de donner les tables d'états en respectant la syntaxe.

```
variable
 SM : machine of bits (E,Q) with states
                              ( S1 = B"10", -- \                    \
                                S2 = B"00", -- / Inst. sur 1 cycle  |
                                S3 = B"11", --                       > Sur 2 cycles
                                S4 = B"01");--                      /
begin
...
 SM.clk=CLK; SM.reset=RST; -- Le signal RESET est obligatoire
 table
 -- Etat actuel => Etat suivant
         SM ,OP => SM;
         S1 ,X  => S2;
         S2 ,0  => S1;
         S2 ,1  => S3;
         S3 ,X  => S4;
         S4 ,X  => S1;
 end table;
...
end;
```

Pour simplifier la structure du circuit combinatoire, on peut remarquer que :

- La transition S1 → S2 est présente dans les 2 cycles, elle doit donc être réalisée quel que soit l'état de OP. Ceci justifie la valeur X (indifférent) dans la définition de table ci-dessus.
- Les états S3 et S4 n'existent que dans le cycle à 4 phases. L'état du signal OP est donc également indifférent dans les transitions S3 → S4 et S4 → S1.

4.11.2 Exécution des instructions

Les codes opérations des instructions à exécuter sont placés dans la mémoire programme. La première action à l'exécution d'une instruction consiste donc à lire le code opération dans la mémoire programme et à le ranger dans le registre d'instruction RI.

Le séquenceur exécute alors une suite d'actions dépendant du code opération mémorisé dans RI.

4.11.3 Recherche du code opération

L'exécution de chaque instruction commence ainsi par la lecture de son code opération dans la mémoire programme pour le ranger dans le registre d'instruction RI. Le compteur ordinal PC indique son adresse dans la mémoire programme.

Pour ce faire, le séquenceur doit :

- Appliquer l'état du compteur ordinal sur le bus d'adresse : SEL_AD[1..0] ← 0 (voir §4.2)
  ⇒ ADR[7..0] = PC[7..0]
- Lire l'octet pointé dans la mémoire programme et le placer à l'entrée "data" du registre d'instruction :
  R_W ← "1" : lecture sur le bus de données externes (voir §4.1)

SEL_DI[1..0] ← 2 : DI[7..0] = D[7..0] (voir §4.1)

- Activer le signal WR_RI pour écrire cet octet (le code opération) dans le registre RI

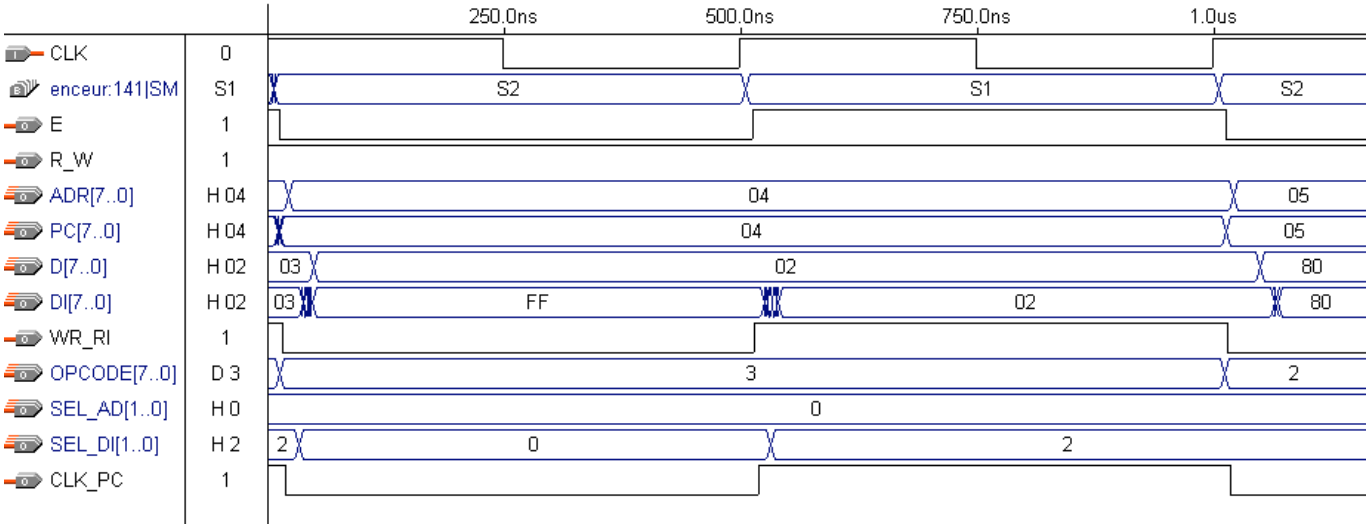
Ces opérations sont effectuées pendant l'état S1 de la machine d'états. La description AHDL est la suivante :

```
defaults
  SEL_ALU[]=0;          -- ALU[]=A[] pour affecter CC[]
  SEL_DI[]=2; R_W=Vcc;-- DI[]=D[] et R_W="1" pour lecture mémoire externe
  SEL_AD[]=0;           -- ADR[]=PC[]
  WR_RA=gnd; RAZ_A=gnd; WR_A=gnd; -- Accu A non modifié
  WR_RT=gnd;                      -- Registre RT non modifié
  WR_RI=gnd; OP_INT=gnd;          -- Registre RI non modifié
  WR_CCR=gnd;                     -- Registre CCR non modifié
  WR_X=gnd;                       -- Registre X non modifié
  SEL_CCR=gnd;                    -- Entrée CCR normale
  CLK_PC=gnd; LD_PC=gnd; SET_PC=gnd; -- Etats de repos signaux PC
  CLK_SP=gnd; LD_SP=gnd; UD_SP=gnd;  -- Etats de repos signaux SP
  OP=gnd;                       -- Machine d'état SM sur 2 cycles
end defaults;
…
if SM==S1 then WR_RI=Vcc; -- Mémorisation code opération à la fin de S1
               CLK_PC=Vcc;-- Incrémentation PC à la fin de S1
end if;
```

Notes :

- La structure AHDL "defaults … end defaults" permet de définir les états par défaut des sorties de la fonction quand elles ne sont pas affectées par ailleurs.
- Du fait de la synchronisation de toutes les fonctions séquentielles, la mémorisation du code opération dans RI est réalisée au flanc montant de CLK à la fin de la phase S1. Le temps d'accès max de la mémoire programme est donc égal à une période d'horloge.
- L'incrémentation du compteur ordinal PC permet de pointer l'opérande éventuel de l'instruction actuelle ou le code opération de l'instruction suivante (si l'instruction actuelle est codée sur un seul octet).

Le chronogramme ci-dessous représente les signaux cités produit par la fonction "séquenceur" implantée dans une EPLD de type EPF10K20RC240-4. Le curseur de lecture d'états est placé à 750nS.



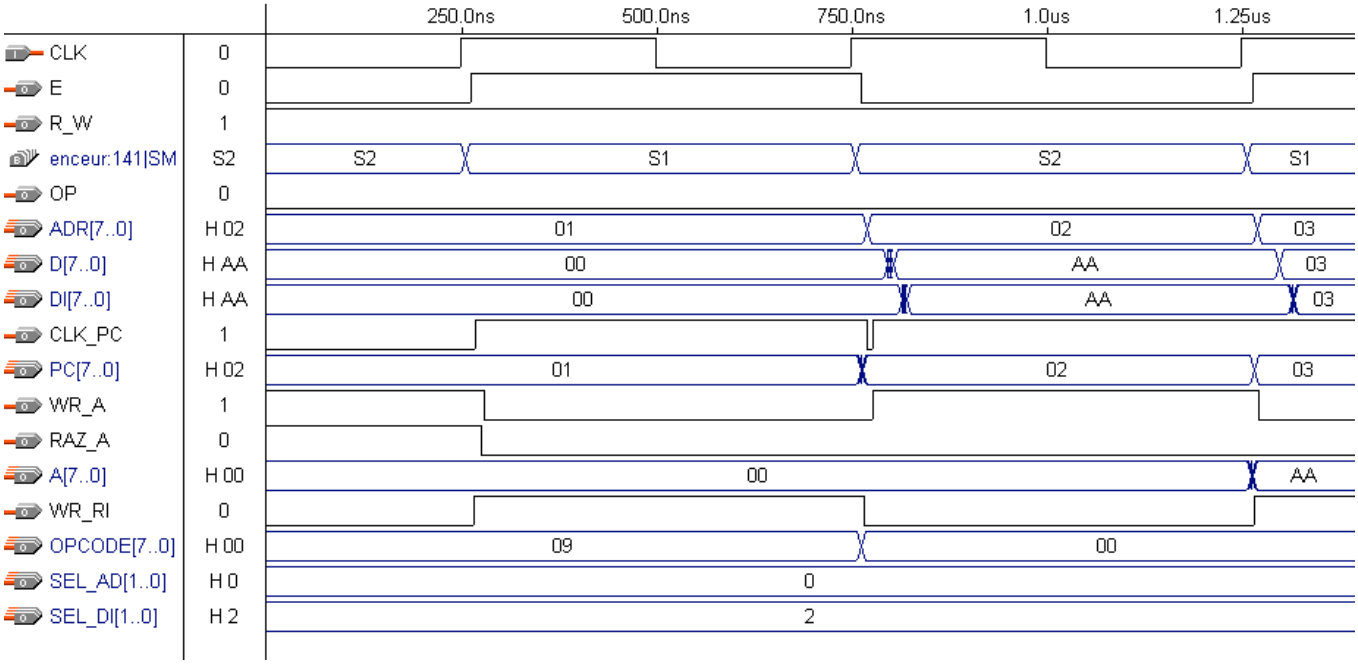
On observe les événements suivants **pendant la phase S1** :

- E = "1". Ceci est conforme à la machine d'état (voir §4.11.1)
- R_W = "1" : état par défaut. Le bus de données externe est en haute impédance.
- SEL_AD[1..0] = 0 : valeur par défaut. Le multiplexeur MUX4 aiguille l'état du compteur ordinal PC sur le bus d'adresse ADR (voir §4.2).
- ADR[7..0] = 04_H. Il s'agit de l'état du compteur ordinal PC. A l'adresse 04_H de la mémoire "programme" externe est placé le code opération 02_H de l'instruction à exécuter (STA adr). L'instruction précédente du programme (INC = 03_H) est rangée à l'adresse précédente, soit 03_H.
- SEL_DI[2..0] = 2 : valeur par défaut. Le multiplexeur MUX8 aiguille le bus de données externe D[7..0] vers le bus de données internes DI[7..0]. La mémoire programme externe place alors sur le bus de données le contenu de l'adresse 04_H, soit 02_H dans l'exemple. Ce code opération se retrouve à l'entrée du registre d'instruction RI.
- WR_RI = "1" : le code opération (02_H)est mémorisé dans RI à la fin de la phase S1.

- CLK_PC = "1" : le compteur ordinal est incrémenté à la fin de S1 pour pointer l'opérande "adr" dans la mémoire programme (valeur 80_H à l'adresse 05_H).

4.11.4 Exemple d'exécution d'une instruction

On décrit ci-dessous le séquencement détaillé de l'instruction : **LDA #val**, de la recherche du code opération à l'exécution effective de l'opération.



- Phase S1 :
  - Recherche du code opération (00_H) à l'adresse 01_H (idem §4.11.3).
  - CLK_PC = "1" pour pointer l'opérande "val" à l'adresse 02_H dans la mémoire programme
- Phase S2 : lecture de l'opérande "val" (AA_H dans cet exemple) à l'adresse 02_H de la mémoire programme.
  - SEL_AD[1..0] = 0 pour placer l'état du PC sur le bus d'adresse ADR[7..0]
  - R_W = "1" pour placer le bus de données en haute impédance
  - SEL_DI[2..0] = 2 pour obtenir DI[7..0] = D[7..0] (voir MUX8 §4.1) et ainsi placer l'octet lu depuis la mémoire externe à l'entrée du registre "Accu_A"
  - WR_A = "1" : le registre "Accu_A" est affecté au flanc montant de CLK à la fin de S2. L'opération de l'instruction "LDA #val" est réalisée : l'accu A est affecté avec la valeur AA_H.
  - CLK_PC = "1" : le compteur ordinal est incrémenté à la fin de S2 pour préparer la recherche du code opération de l'instruction suivante.

Note : le temps d'accès de la mémoire "externe" est particulièrement court dans cet exemple. En fait, le modèle EP10K20 permet l'implantation de cette mémoire (ROM et RAM) sur la puce elle-même. Leurs capacités sont limitées, mais les temps d'accès sont très faibles.

Description AHDL :

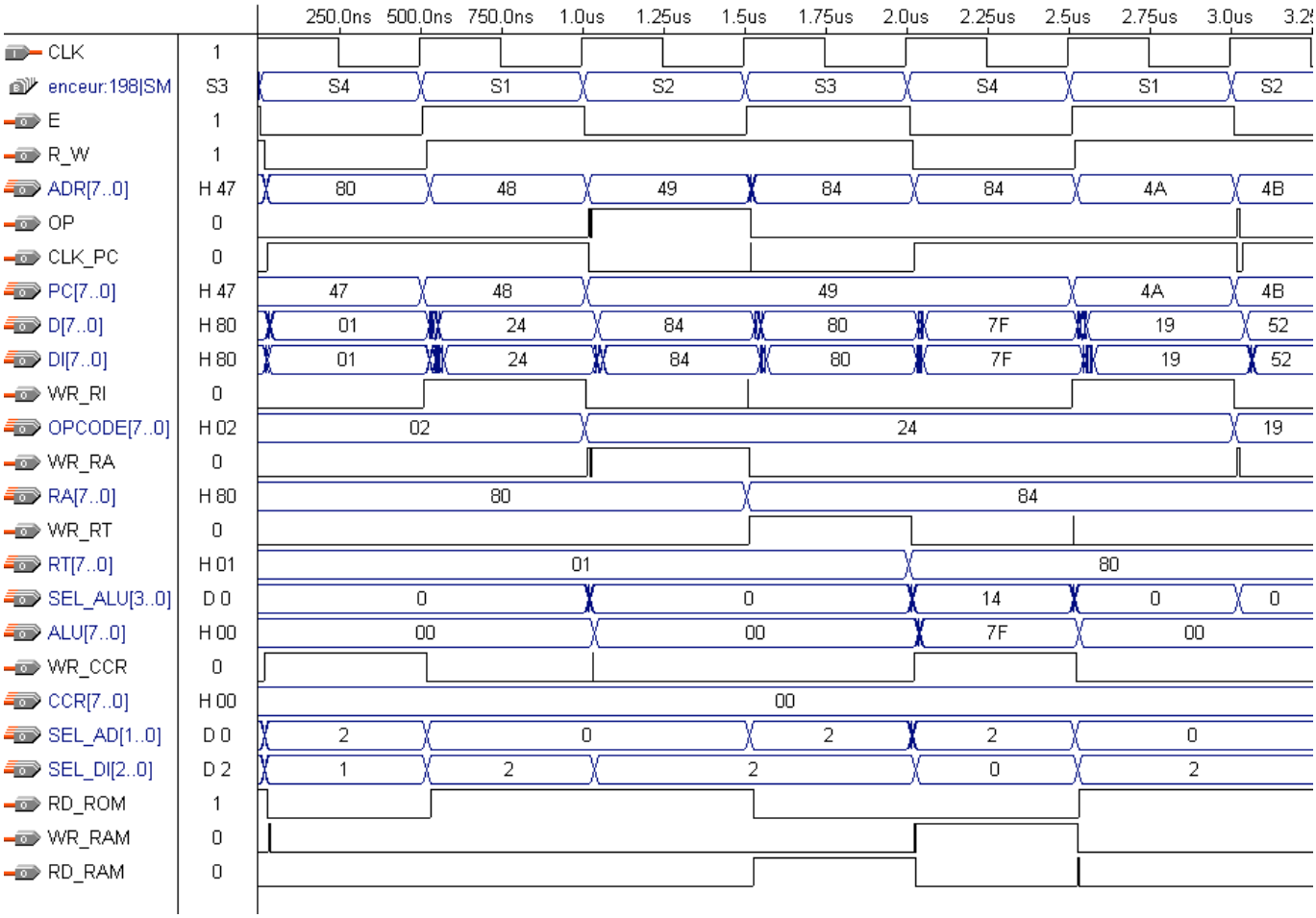
La structure "case … end case;" est utilisée ici pour réaliser le décodage de chaque instruction. Dans la description complète du séquenceur, il y a autant de "when" que d'instructions à décoder. On ne reprend ci-dessous qu'une partie de cette description.

```
case OPCODE[] is
 when H"00" =>  -- Code opération de l'instruction "Load immédiat accu A"
    case SM is
     when S2 => WR_A=Vcc;    -- Ecriture de l'opérande ds accu A à la fin de S2
                WR_CCR=Vcc; -- Mémorisation de C, N et Z de l'ALU à la fin de S2
                CLK_PC=Vcc; -- Incrémenter PC à la fin de l'instruction
    end case;
```

Note : L'écriture est simplifiée grâce à la définition des états par défaut (voir §4.11.3).
La phase S1 ne dépend pas du code opération et est déjà décrite (voir §4.11.3).

4.11.5 Exemple d'exécution d'une instruction plus complexe

On décrit ci-dessous le séquencement détaillé de l'instruction : **DEC adr**, de la recherche du code opération à l'exécution effective de l'opération.



Nouveaux signaux (produits par une fonction de décodage externe au CPU) :

- RD_ROM : validation de la sortie 3 états de la mémoire programme externe
- RD_RAM : validation de la sortie 3 états de la RAM externe
- WR_RAM : validation d'écriture dans la RAM externe

L'instruction **DEC adr** nécessite les 4 phases de la machine d'état SM :

- Phase S1 :
  - Recherche du code opération (24_H) à l'adresse 48_H . Mémorisation dans le registre RI (OPCODE[7..0]) à la fin de S1 (WR_RI="1").
  - CLK_PC = "1" pour lire durant S2 l'opérande "adr" à l'adresse suivante dans la mémoire programme (49_H)
- Phase S2 : lecture de l'adresse "adr" de l'opérande (84_H dans cet exemple) à l'adresse 49_H de la mémoire programme:
  - SEL_AD[1..0] = 0 pour placer l'état du PC (49_H) sur le bus d'adresse ADR[7..0]
  - R_W = "1" pour placer le bus de données en haute impédance
  - SEL_DI[2..0] = 2 pour appliquer l'adresse de l'opérande (ici 84_H) à l'entrée du registre RA
  - WR_RA = "1" pour affecter le registre RA avec cette valeur à la fin de S2
  - OP = "1" pour que la machine d'état SM passe à la phase S3
- Phase S3 : placer l'adresse "adr" de l'opérande sur le bus d'adresse ADR[7..0]
  - SEL_AD[1..0] = 2 pour placer l'état du registre RA sur le bus d'adresse ADR[7..0] : ici ADR[7..0] = 84_H
  - RD_RAM = "1" pour lire le contenu de cette case mémoire en RAM : ici D[7..0] = 80_H
  - SEL_DI[2..0] = 2 pour appliquer cet opérande (ici 80_H) à l'entrée du registre RT
  - WR_RT = "1" pour affecter le registre RT avec cette valeur à la fin de S3 : ici RT[7..0] = 80_H
- Phase S4 : réalisation de l'opération et mémorisation du résultat :
  - SEL_ALU = 14 (E_H) pour sélectionner l'opération : S[7..0] ← B[7..0] – 1, soit ALU[7..0] ← RT[7..0] – 1
  - SEL_DI = 0 pour appliquer le résultat de l'ALU sur DI[7..0] (ici DI[]=ALU[]=RT[] – $1=80_H – 1=7F_H$). On observe le temps de propagation de l'ALU et du multiplexeur MUX8.
  - WR_CCR = "1" pour mémoriser les indicateurs Z, C et N dans le registre CCR à la fin de S4 (ici il reste à 0)

- R_W = "0" pour valider le buffer BUF_8BITS et placer le résultat de l'ALU sur le bus de données externes D[7..0]
- SEL_AD[1..0] = 2 pour placer l'état du registre RA sur le bus d'adresse ADR[7..0], soit l'adresse **adr** de l'opérande
- WR_RAM = "1" pour valider l'écriture dans la RAM à la fin de S4
- CLK_PC = "1" : le compteur ordinal est incrémenté à la fin de S4 pour préparer la recherche du code opération de l'instruction suivante.

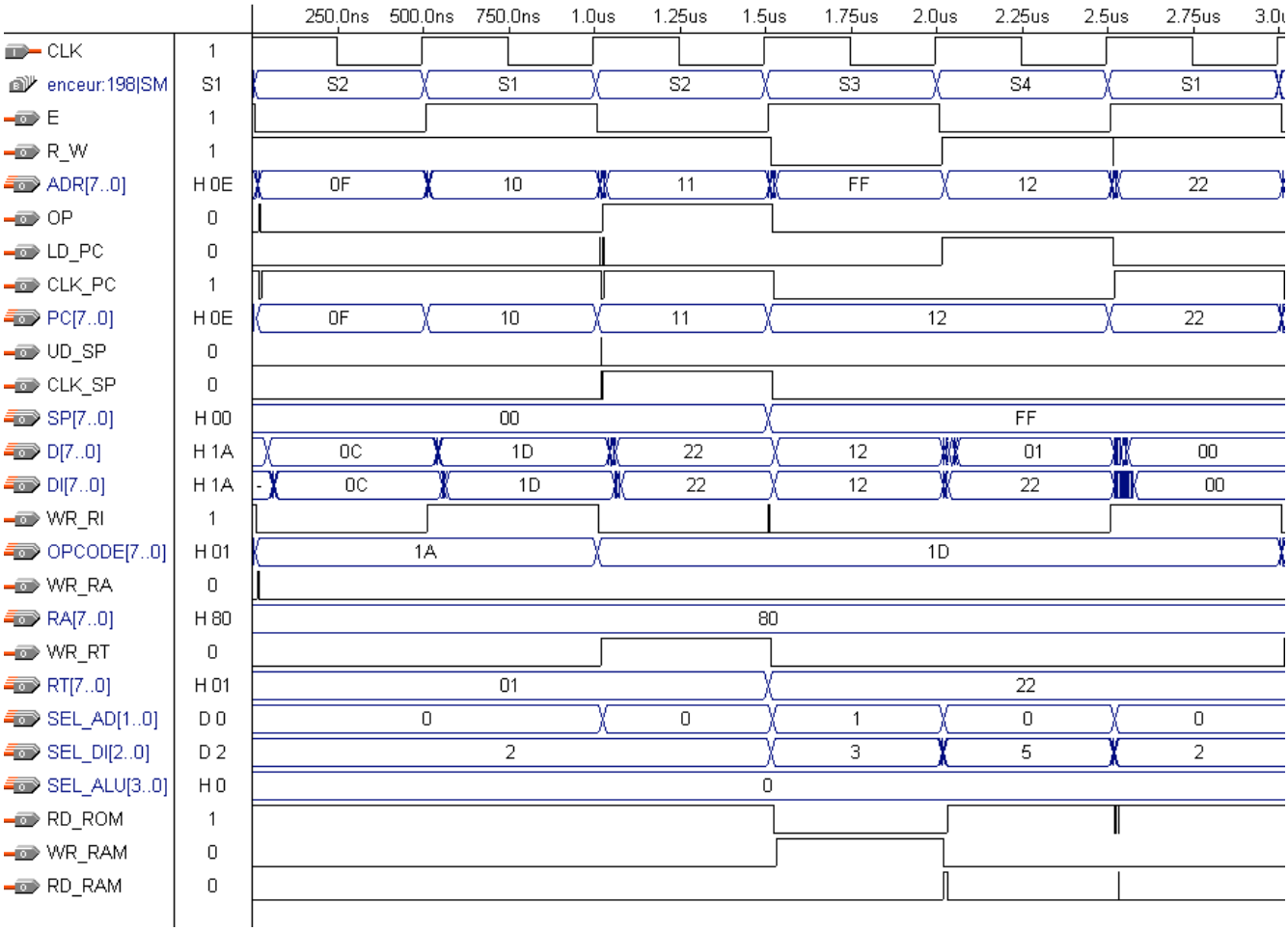
Description AHDL :

```
case OPCODE[] is
…
when H"24" =>  -- DEC Adr
  case SM is
     when S2 => OP=Vcc;      -- Pour aller jusqu'à S4
                WR_RA=Vcc;   -- RA[] <- Adr à la fin de S2
     when S3 => SEL_AD[]=2;  -- DI[]=(Adr)
                WR_RT=Vcc;   -- RT[] <- (Adr) à la fin de S2
     when S4 => SEL_ALU=14;  -- ALU[] <- RT[]-1
                R_W=gnd; SEL_AD[]=2; SEL_DI[]=0; -- (Adr) <- ALU[]
                WR_CCR=Vcc; -- Mémorisation de C, N et Z de l'ALU à la fin de S4
                CLK_PC=Vcc; -- Pour pointer l'instruction suivante
    end case;
```

4.11.6 Appel et retour de sous-programme

La particularité de ces instructions est l'utilisation de la pile pour la sauvegarde des adresses de retour.

Séquencement détaillé de l'instruction : **JSR adr : Jump Subroutine**



L'instruction **JSR adr** (ici "JSR 22_H") nécessite les 4 phases de la machine d'état SM :

- Phase S1 :
  - Recherche du code opération ($1D_H$) à l'adresse 10_H . Mémorisation dans le registre RI (OPCODE[7..0]) à la fin de S1 (WR_RI="1").
  - CLK_PC = "1" pour lire durant S2 l'opérande "adr" à l'adresse suivante (11_H) de la mémoire programme

- Phase S2 : lecture du contenu de l'adresse "adr" de la mémoire programme (22_H dans cet exemple) :
  - SEL_AD[1..0] = 0 pour placer l'état du PC (11_H) sur le bus d'adresse ADR[7..0]
  - R_W = "1" pour placer le bus de données en haute impédance
  - SEL_DI[2..0] = 2 pour appliquer le contenu de l'adresse 11_H (soit "adr"=22_H) à l'entrée du registre RT. Il s'agit de l'adresse du sous-programme appelé.
  - WR_RT = "1" pour affecter le registre RT avec cette adresse à la fin de S2, soit RT[7..0]=22_H
  - CLK_SP = "1" et UD_SP = "0" pour décrémenter le pointeur de pile SP à la fin de S2 : le pointeur de pile pointe la prochaine case libre dans la RAM
  - CLK_PC = "1" pour incrémenter PC à la fin de S2. Il indique alors l'adresse de retour (12_H)
  - OP = "1" pour que la machine d'état SM passe à la phase S3
- Phase S3 : sauvegarde de l'adresse de retour dans la pile :
  - SEL_AD[1..0] = 1 pour placer l'état du pointeur de pile SP sur le bus d'adresse ADR[7..0]. Il indique l'adresse de sauvegarde dans la RAM de l'adresse de retour
  - SEL_DI[2..0] = 3 et R_W = "0" pour appliquer l'état du PC (c'est à dire l'adresse de retour, ici 12_H) sur le bus de données externes D[7..0]
  - R_W = "0" et WR_RAM = "1" pour valider l'écriture dans la RAM à la fin de S3 (empilage de l'adresse de retour)
- Phase S4 : saut à l'adresse du sous-programme
  - SEL_DI[2..0] = 5 pour appliquer l'état du registre RT (soit l'adresse du sous-programme : ici 22_H) à l'entrée de prépositionnement du compteur programme PC.
  - LD_PC = "1" pour affecter le PC avec l'adresse du sous-programme à la fin de S4
  - SEL_AD[1..0] = 0 pour placer l'état du PC sur le bus d'adresse ADR[7..0]

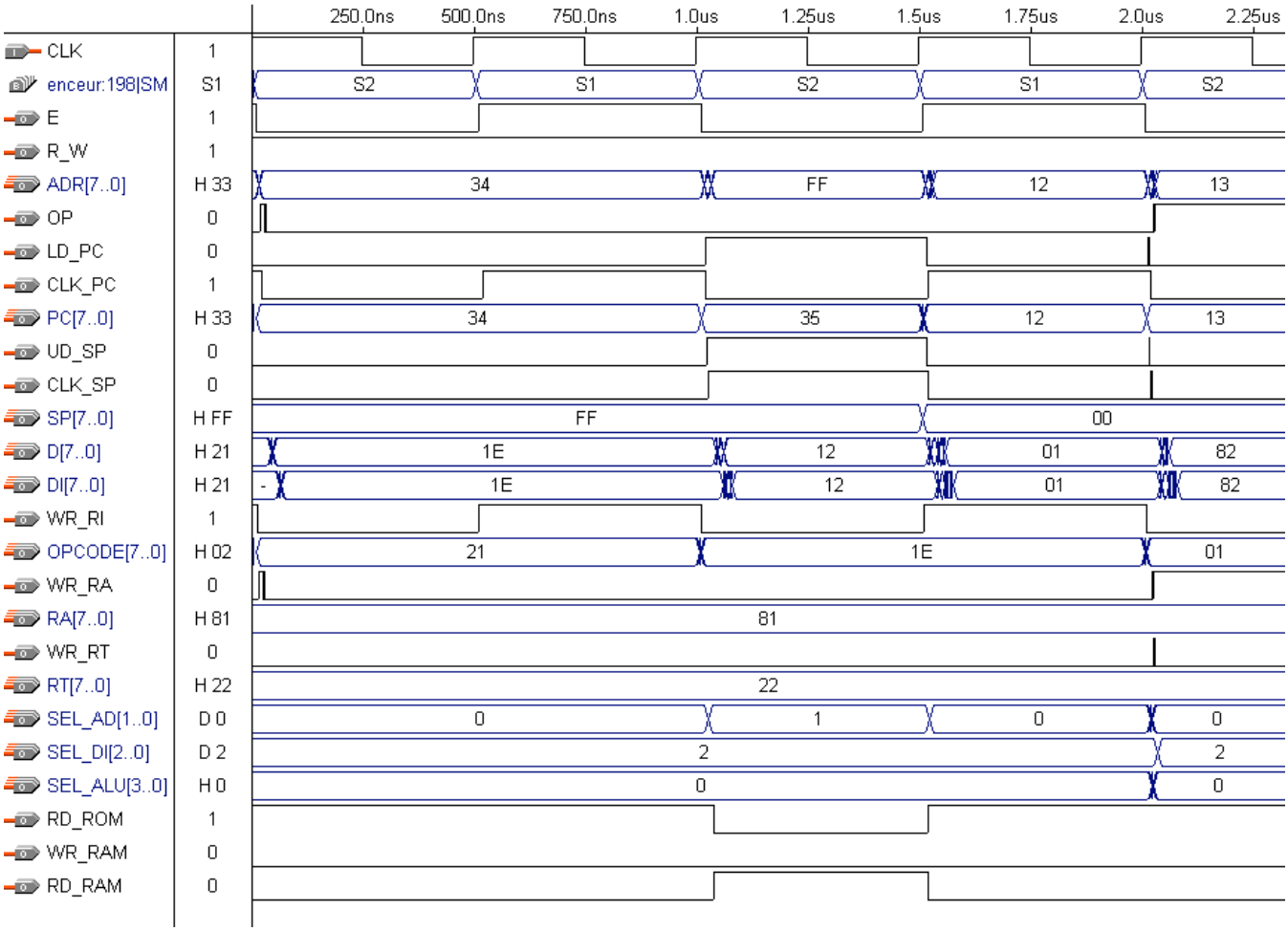
La machine d'état SM passe alors naturellement à la phase S1 et exécute l'instruction pointée par l'état actuel du PC (ici 22_H), soit la première instruction du sous-programme.

Description AHDL :

```
case OPCODE[] is
…
 when H"1D" =>  -- JSR Adr
   case SM is
     when S2 => OP=Vcc;     -- Pour aller jusqu'à S4
                CLK_PC=Vcc; -- PC indique l'adresse de retour à la fin de S2
                WR_RT=Vcc;  -- Ecriture adresse de branchement ds RT à la fin de S2
                CLK_SP=Vcc; -- Décrémenter SP à la fin de S2
     when S3 => SEL_DI[]=3; -- PC[] = adresse de retour sur DI
                SEL_AD[]=1; -- SP sur bus d'adresse
                R_W=gnd;    -- Ecriture adresse de retour ds la pile
     when S4 => SEL_DI[]=5; -- DI[]=RT[]=adresse de branchement
                LD_PC=Vcc;  -- PC affecté par l'adr. de branchement à la fin de S4
    end case;
```

Séquencement détaillé de l'instruction : **RTS :**

L'instruction RTS (ReTurn Subroutine) est la dernière d'un sous-programme. Elle réalise le retour au programme appelant. Voici un exemple de traitement :



L'instruction **RTS** ne nécessite que 2 phases de la machine d'état SM :

- Phase S1 :
  - Recherche du code opération (ici $1E_H$) à l'adresse PC[7..0] (ici 34_H). Mémorisation dans le registre RI (OPCODE[7..0]) à la fin de S1 (WR_RI="1").
  - CLK_PC = "1" : inutile mais sans conséquence car PC est affecté pendant S2
- Phase S2 : lecture de l'adresse de retour dans la pile et affectation du PC
  - SEL_AD[1..0] = 1 pour placer l'état du pointeur de pile SP (FF_H) sur le bus d'adresse ADR[7..0]. C'est l'adresse à laquelle est mémorisée l'adresse de retour. Le décodage d'adresse externe active RD_RAM pour lire cette case mémoire.
  - UD_SP = "1" et CLK_SP = "1" pour incrémenter le pointeur de pile SP à la fin de S2 (mise à jour du pointeur de pile après dépilage).
  - SEL_DI[2..0] = 2 (valeur par défaut) pour appliquer l'octet lu (12_H) à l'entrée de prépositionnement du PC via DI[7..0]
  - LD_PC = "1" pour affecter le PC avec cette valeur à la fin de S2. On aura alors réalisé le saut à l'adresse de retour du sous-programme.

La machine d'état SM passe alors naturellement à la phase S1 et exécute l'instruction pointée par l'état actuel du PC (ici 12_H), soit l'instruction du programme appelant qui suit JSR adr.

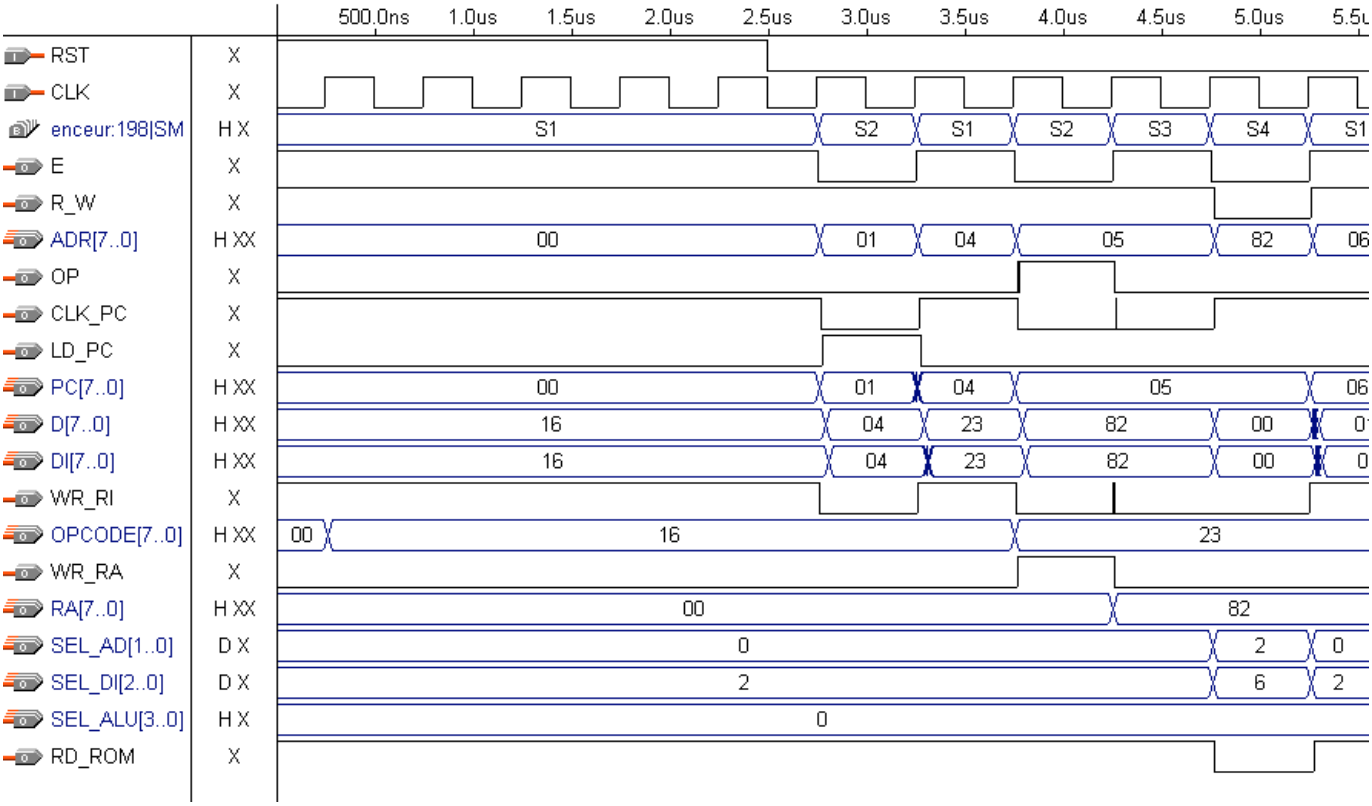
Description AHDL :

```
case OPCODE[] is
…
 when H"1E" =>  -- RTS
   case SM is
     when S2 => SEL_AD[]=1; -- SP sur bus d'adresse
                LD_PC=EN;   -- Affectation adresse de retour
                UD_SP=Vcc; CLK_SP=Vcc; -- Incrémentation SP
   end case;
```

4.11.7 Reset

Le signal "reset" est indispensable à toute fonction séquentielle. Le CPU comporte donc une entrée RST qui a les effets suivants :

- Raz asynchrone du PC. La première instruction du programme est donc à l'adresse 0.
- Reset de la machine d'état SM du séquenceur d'instruction : forcer la phase S1
- Validation du chargement synchrone du registre d'instruction RI avec le code opération de la première instruction placée à l'adresse 00_H (soit 16_H dans l'exemple).



A la libération du Reset (RST = "0"), la machine d'état SM fonctionne normalement et exécute l'instruction placée dans le registre RI (soit JMP = 16_H dans l'exemple).

4.11.8 Interruption

Le séquenceur d'instruction comporte une entrée de demande d'interruption du programme : INT. Cette demande n'est prise en compte que si le signal de validation interne EN_INT est à "1". Le séquenceur d'instruction réagit alors de la façon suivante :

- L'instruction en cours se termine
- Sauvegarde de l'état du registre CCR et de l'adresse de retour d'interruption dans la pile
- Saut à l'adresse 2 (c'est l'adresse de la première instruction du programme d'interruption; il s'agit normalement de JMP)
- Traitement du programme d'interruption
- Retour au programme interrompu et restitution de l'état du registre CCR avec l'instruction RTI, la dernière du programme de traitement de l'interruption.

4.11.8.1 Signal de validation interne : EN_INT

Il s'agit d'une bascule RS synchrone dont l'état est contrôlé par 2 instructions :

- SEI (21_H) : SEt Interrupt = mise à "1" de EN_INT
- CLI (22_H) : CLear Interrupt = mise à "0" de EN_INT

Descriptions AHDL :

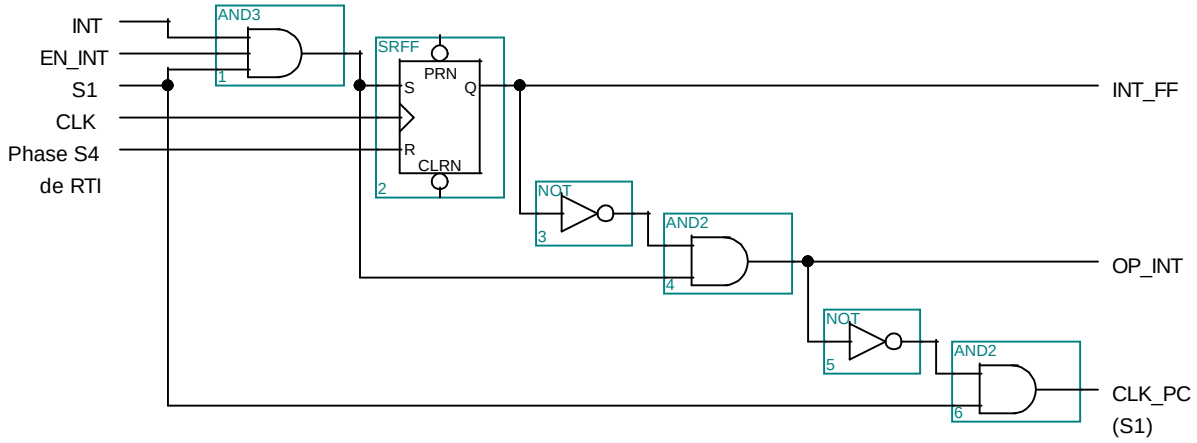
```
variable
 EN_INT : SRFF; -- Déclaration de la bascule RS de validation d'interruption
…
defaults
 …
 EN_INT.R=gnd; EN_INT.S=gnd; -- Etats par défaut des entrées S et R
end defaults;
…
EN_INT.clk=CLK;              -- Horloge bascule RS validation d'interruption
…
case OPCODE[] is
…
 when H"21" =>  -- SEI
   case SM is
     when S2 => EN_INT.S=Vcc ; -- Mise à "1" bascule RS -> validation INT
   end case;
 when H"22" =>  -- CLI
   case SM is
     when S2 => EN_INT.R=Vcc ; -- Mise à "0" bascule RS -> inibition INT
   end case;
```

4.11.8.2 Traitement de la demande d'interruption

La demande d'interruption n'est prise en compte que pendant la phase S1 de la machine d'état SM (recherche du code opération de la prochaine instruction) pour que le traitement de l'instruction en cours se termine.

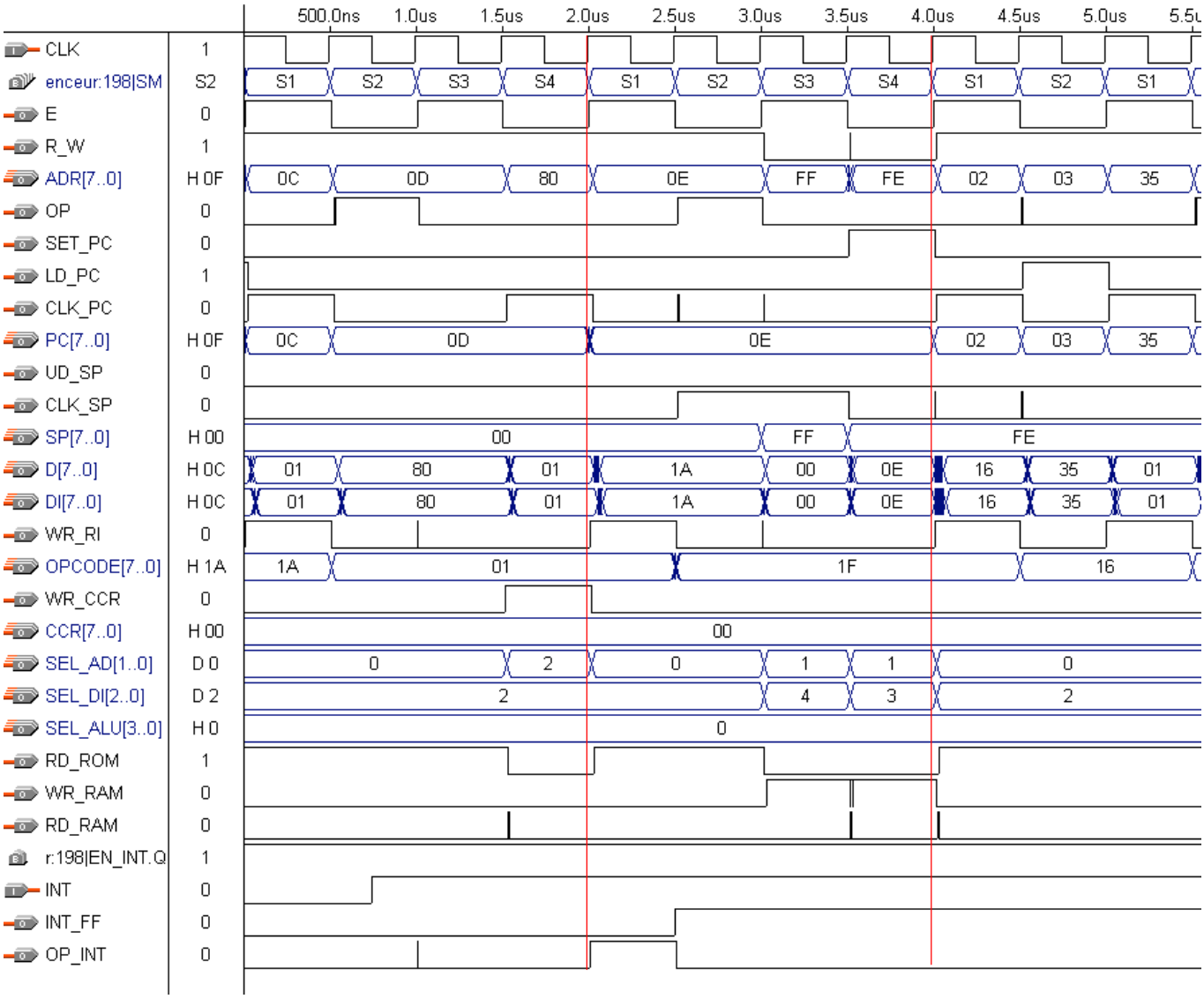
Cette demande doit être mémorisée pour éviter la prise en compte d'une deuxième demande avant que la première soit complètement traitée.

Le schéma logique associé est le suivant :



- INT_FF : mémoire synchrone de la demande d'interruption :
  - Mise à "1" à la fin de la phase S1 de la machine d'état SM si INT = "1" (évidemment) et que la prise en compte est validée (EN_INT = "1").
  - Mise à "0" à la fin de phase S4 de l'instruction RTI (fin du traitement de l'interruption)
- OP_INT : valide l'affectation synchrone du registre RI avec le code opération de l'instruction INT ($1F_H$) à la fin de la phase S1. En effet, cette instruction n'est pas, par principe, placée dans la mémoire "programme", on utilise alors l'entrée "sset" du registre RI (voir §4.5) qui permet de l'affecter à une valeur quelconque.
  Le signal OP_INT ne passe à "1" que pendant la **première** phase S1 qui suit la demande d'interruption. Pour les phases S1 suivantes, INT_FF est à "1", ce qui force OP_INT à "0".
- Normalement CLK_PC est à "1" pendant les phases S1 des autres instructions. Mais, pour INT, il faut éviter l'incrémentation du PC à la fin de la phase S1; il n'indiquerait plus alors l'adresse de retour d'interruption à sauvegarder dans la pile.

Exemple de traitement :



Commentaires :

- La bascule RS EN_INT a été préalablemement mise à "1" avec l'instruction SEI
- Le signal d'entrée INT est activé en plein milieu d'une instruction à 4 phases (cet événement peut arriver à tout moment)
- La bascule INT_FF n'est mise à "1" qu'à la fin de la première phase S1 qui suit la demande INT. On attend bien la fin de l'instruction en cours.
  L'instruction suivante dans le programme interrompu se trouve à l'adresse $0E_H$; elle ne sera exécutée qu'au retour du programme d'interruption.
- Pendant la première phase S1 qui suit la demande, les conditions sont réunies pour que OP_INT passe à "1" (INT à "1", EN_INT à "1", phase S1 et bascule INT_FF encore à "0").
- Le registre d'instruction RI est alors affecté à la fin de cette phase S1 par la valeur 31 (voir §4.5), c'est à dire avec le code opération de l'instruction INT ($1F_H = 31_{10}$).
- CLK_PC ne passe pas à "1" pendant cette phase S1 pour éviter l'incrémentation du PC car il indique l'adresse de retour à mémoriser dans la pile
- Le séquenceur d'instruction traite alors l'instruction INT pendant les phases S2 à S4 suivantes :
  - S2 : OP = "1" pour que la machine d'états aille jusqu'à S4
    UD_SP = "0" et CLK_SP = "1" pour décrémenter le pointeur de pile SP à la fin de S2 : préparation à l'empilage de l'état de CCR
  - S3 : SEL_DI[] = 4 pour placer l'état du registre CCR sur DI[]
    SEL_AD[] = 1 pour placer l'état du pointeur de pile sur le bus d'adresse ADR[]
    R_W = "0" pour valider l'écriture de l'état du CCR dans la RAM (empilage) : 00_H à l'adresse FF_H dans l'exemple.
    UD_SP = "0" et CLK_SP = "1" pour décrémenter le pointeur de pile SP à la fin de S3 : préparation à l'empilage de l'adresse de retour.
  - S4 : SEL_DI[] = 3 pour placer l'état du compteur programme PC sur DI[]. Il s'agit de l'adresse de retour
    SEL_AD[] = 1 pour placer l'état du pointeur de pile sur le bus d'adresse ADR[]

R_W = "0" pour valider l'écriture de l'adresse de retour dans la RAM (empilage) : $0E_H$ à l'adresse FE_H dans l'exemple.
SET_PC = "1" : pour affecter le compteur programme avec 02_H à la fin de S4 (voir §4.6). Il s'agit de l'adresse de la première instruction du programme d'interruption.

- Le séquenceur continue alors son fonctionnement normal avec la première instruction du programme d'interruption (ici 16_H = JMP, ce qui est le cas général).

Description AHDL :

```
case OPCODE[] is
…
 when H"1F" =>              -- INT = Interruption
   case SM is
     when S2 => OP=Vcc;     -- Pour aller jusqu'à S4
                CLK_SP=Vcc; -- Décrémentation SP avant sauvegarde CCR
     when S3 => SEL_DI[]=4; -- DI[] <- CCR[]
                SEL_AD[]=1; -- SP sur bus d'adresse
                R_W=gnd;    -- Ecriture adresse de retour ds la pile
                CLK_SP=Vcc; -- Décrémentation SP avant sauvegarde PC
     when S4 => SEL_DI[]=3; -- PC[] = adresse de retour sur DI[]
                SEL_AD[]=1; -- SP sur bus d'adresse
                R_W=gnd;    -- Ecriture adresse de retour ds la pile
                SET_PC=Vcc; -- PC[] <- 2 à la fin de S2
   end case;
```

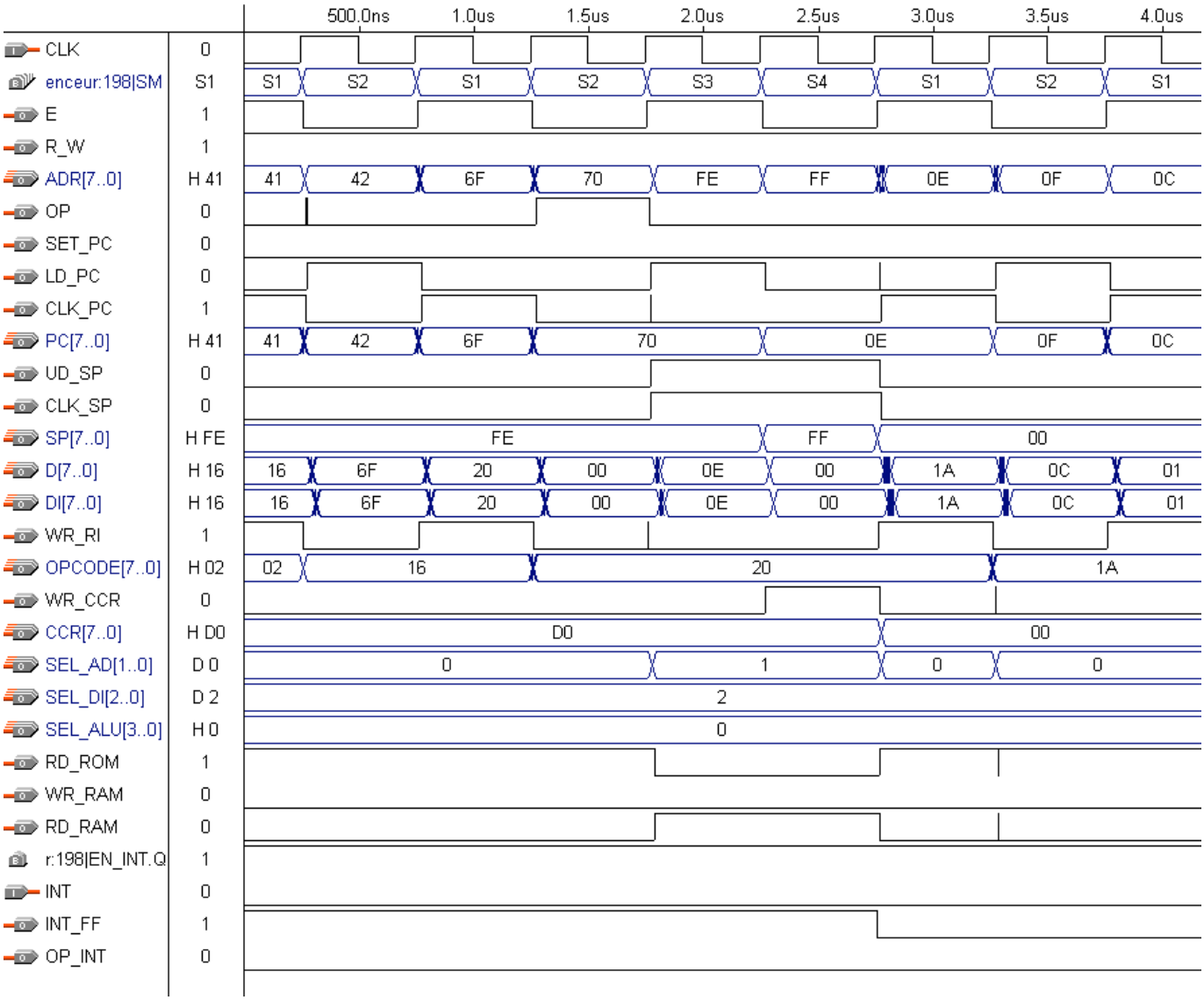
4.11.8.3 Traitement de l'instruction RTI

Cette instruction est la dernière du programme d'interruption, elle doit restituer le contexte du programme interrompu. Voici un exemple (voir le chronogramme page suivante) :
Les phases centrales S1 à S4 sont celles du traitement l'instruction **RTI** par le séquenceur :

- Phase S1 :
  - Recherche du code opération (20_H) à l'adresse $6F_H$. Mémorisation dans le registre RI (OPCODE[7..0]) à la fin de S1 (WR_RI="1").
  - CLK_PC = "1", comme dans toutes les phases S1. Mais l'incrémentation du PC est sans importance ici.
- Phase S2 : restitution du registre CCR depuis la pile
  - OP = "1" pour que la machine d'état SM passe à la phase S3 (cycle à 4 phases)
- Phase S3 : restitution de l'adresse de retour :
  - SEL_AD[1..0] = 1 pour placer l'état du pointeur de pile SP sur le bus d'adresse ADR[7..0]. Il indique l'adresse où est sauvegardé en RAM l'adresse de retour (l'adresse FE_H contient $0E_H$ dans l'exemple)
  - SEL_DI[2..0] = 2 et R_W = "1" pour appliquer l'état du bus de données externes D[7..0] sur le bus de données internes DI[7..0] et donc à l'entrée de prépositionnement du PC.
  - LD_PC = "1" pour affecter le PC avec cette adresse de retour à la fin de S3 : "dépiler" l'adresse de retour
  - UD_SP = "1" et CLK_SP = "1" pour incrémenter le pointeur de pile SP à la fin de S3. Il pointe alors l'état de CCR sauvegardé dans la pile.
- Phase S4 : restitution du registre CCR
  - SEL_AD[1..0] = 1 pour placer l'état du pointeur de pile SP sur le bus d'adresse ADR[7..0]. Il indique l'adresse où est sauvegardé en RAM l'état du registre CCR (l'adresse FF_H contient 00_H dans l'exemple)
  - SEL_DI[2..0] = 2 et R_W = "1" pour appliquer l'état du bus de données externes D[7..0] sur le bus de données internes DI[7..0] et donc à l'entrée du CCR.
  - SEL_CCR = "1" pour indiquer à la fonction CALC_CC d'effectuer : CC[7..0] ← DI[7..0]
  - WR_CCR = "1" pour affecter le registre CCR à la fin de S4 avec la valeur lue dans la pile (ici 00_H)
  - UD_SP = "1" et CLK_SP = "1" pour incrémenter le pointeur de pile SP à la fin de S3. Il retrouve alors sa valeur d'origine (FF_H).

La machine d'état passe alors naturellement à la phase S1 pour exécuter l'instruction placée à l'adresse $0E_H$ (ici $1A_H$) : le programme interrompu reprend son cours.

Exemple de traitement de **RTI** :



Description AHDL :

```
case OPCODE[] is
…
when H"20" =>                                -- RTI
  case SM is
     when S2 => OP=Vcc;                      -- Pour aller jusqu'à S4
     when S3 => SEL_AD[]=1;                  -- SP sur bus d'adresse
                LD_PC=Vcc;                   -- Affectation adresse de retour
                UD_SP=Vcc; CLK_SP=Vcc;       -- Incrémentation SP
       when S4 => SEL_AD[]=1;                -- SP sur bus d'adresse
                SEL_CCR=Vcc; WR_CCR=Vcc;     -- Restitution CCR
                INT_FF.R=Vcc;                -- Raz bascule INT
                UD_SP=Vcc; CLK_SP=Vcc;       -- Incrémentation SP
    end case;
```