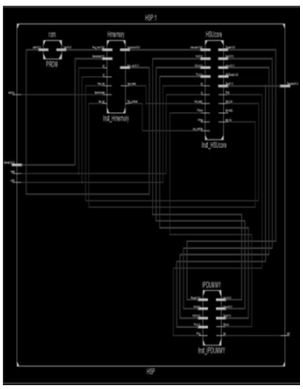
LE PROCESSEUR HOMADE

Le cœur Homade

- UltraRISC processor
  - 11 instructions
- It can do nothing alone
- You can add 2048 IPs for your needs

Smaller = Bigger

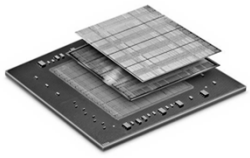


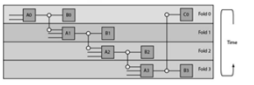
New 3D technologies arriving

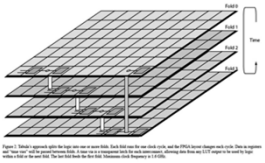
2,5D FPGA: virtex 7

New reconfigurable path could become parallel between 2 dies

Modele of computation has to be ready for that!!!

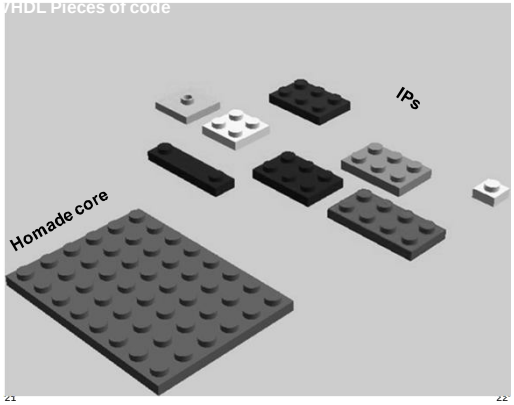






21

VHDL Pieces of code

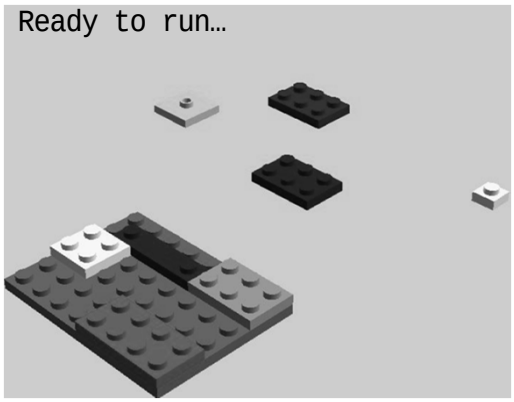


21 22

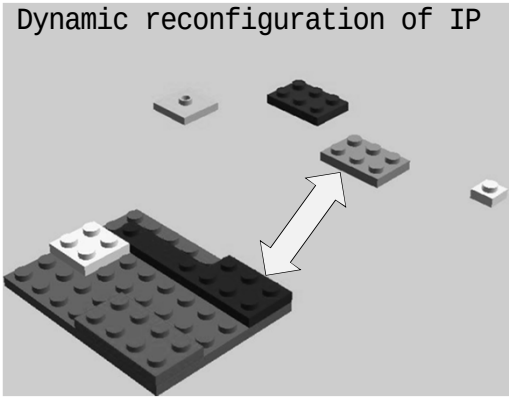
- **Model of computation**
  - **=> Hardware programming**
  - **=> Software progamming**
- **Parallel hardware when you need it!**
  - **=>Partial reconfiguration must be parallel**
- **Dynamicity + complexity**
  - **=>Verifications**

21 22

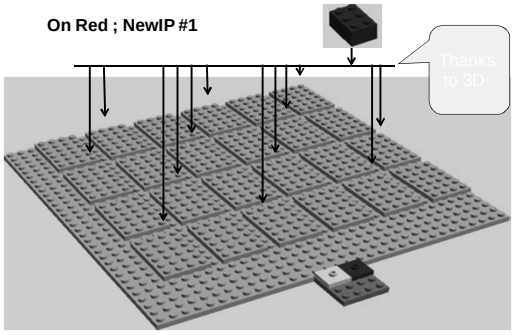
Ready to run…



Dynamic reconfiguration of IP



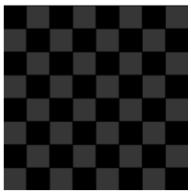
On Red ; NewIP #1



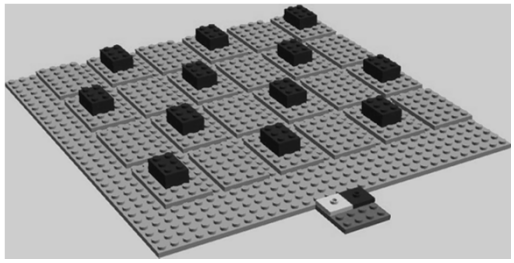
Toward MSPMD model of computation and architecture

- Specific IP and instructions
  - Programmable dynamicity
  - Call SPMD / ORTREE
- MSPMD ex:

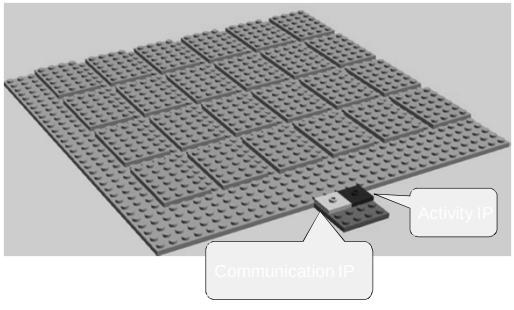
  On Red -- IP ON
  NewIP #1
  SPMD #1
  On black
  NewIP #2
  SPMD #2
  On all
  ORTREE



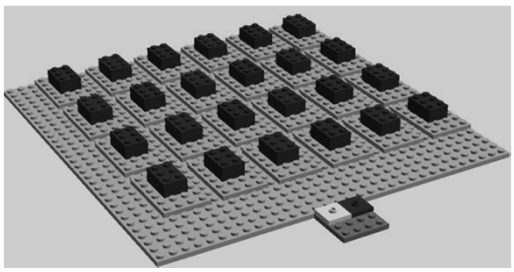
SPMD #1



6 x 4 Homades + Master Homade LEGO prototype

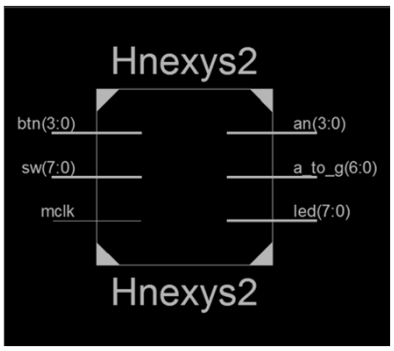


On Black ; NewIP #2, SPMD #2

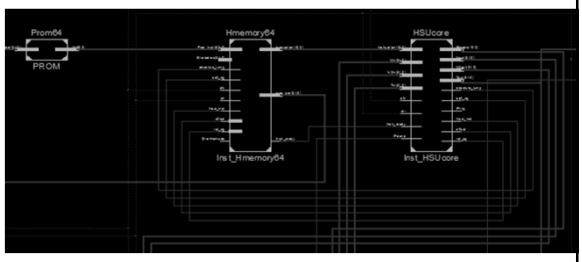


This is Multi-SPMD execution!!

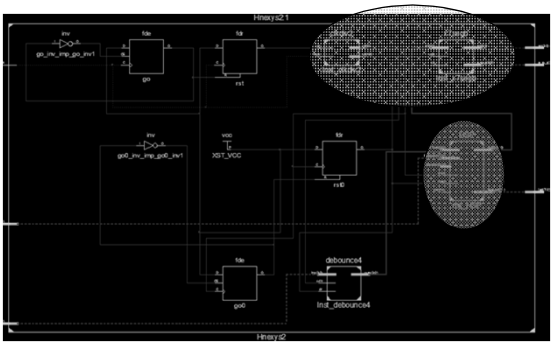
Homade sous ISE



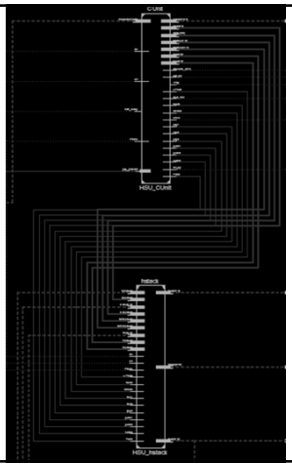
La structure Homade



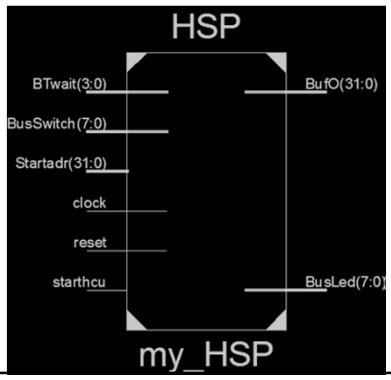
Inside nexys2 ou 3



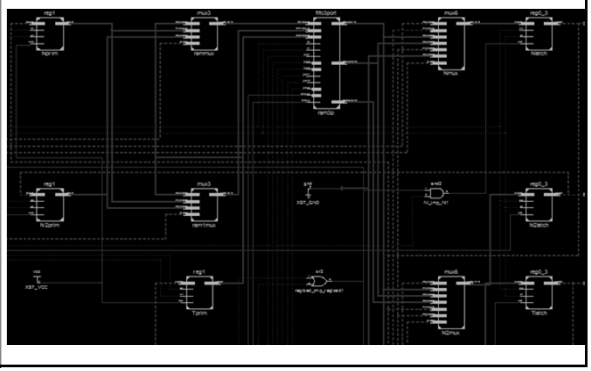
Le HSU



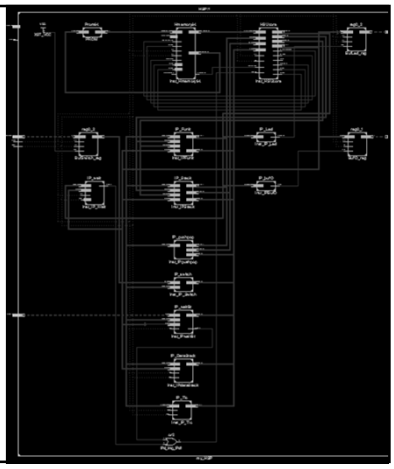
Le cœur Homade



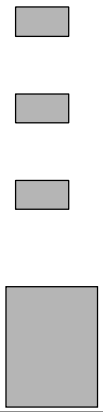
La stack



Avec nos Ips!!!

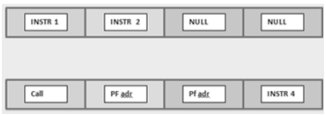


Pop 0
push 0



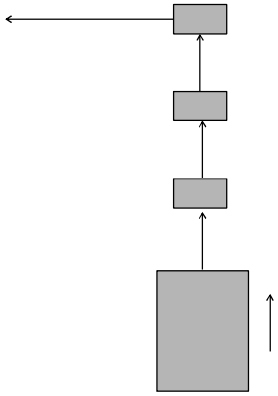
Le jeu d'instructions

- Toutes les instructions sont codées sur des mots de 16 bits
- instructions étendues : 3 mots de 16 bits
  - Elles sont alignées sur des mots de 64 bits



- L'instruction de remplissage est codée '1_11_11_1_1111111111'

1 0



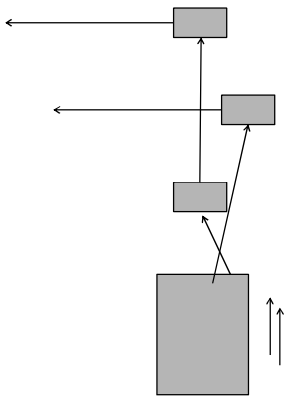
Instruction IP

- Son format sur 16 bits :

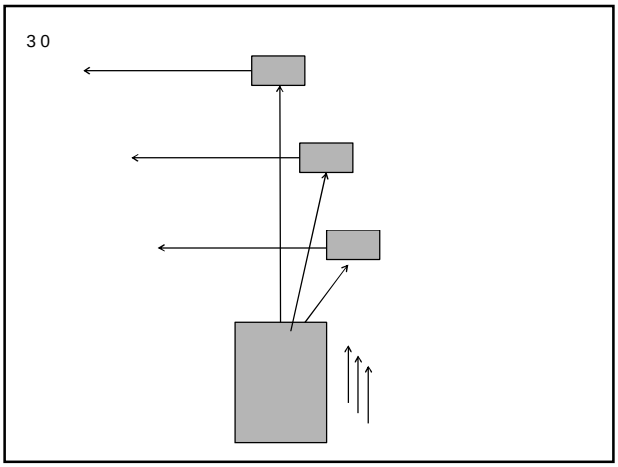
1	X	X	Y	Y	S	I	I	I	I	I	I	I	I	I	I

- XX : Pop 0, 1, 2 ou 3 valeurs retirées de la pile vers l'IP
- YY : Push 0, 1, 2 ou 3 valeurs envoyées par l'IP sur la pile
- S :
  - ' 0' indique un IP qui s'exécute en moins d'un cycle : Short_IP.
  - '1' signifie que l'IP s'exécute sur plus de 1 cycle : Long_IP. signal IPdone est déclenché par l'IP pour signaler la fin d'exécution.
- IIIIIIIIII : sur 10 bits, le numéro de l'Ip à déclencher. On a donc 1024 IP en un cycle et 1024 IP en plus de un cycle.

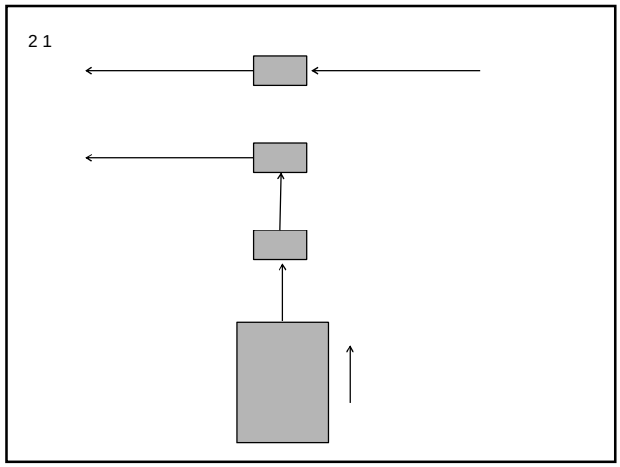
2 0



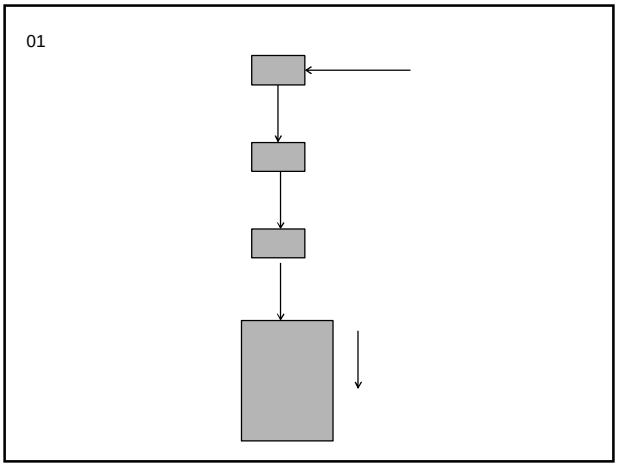
3 0



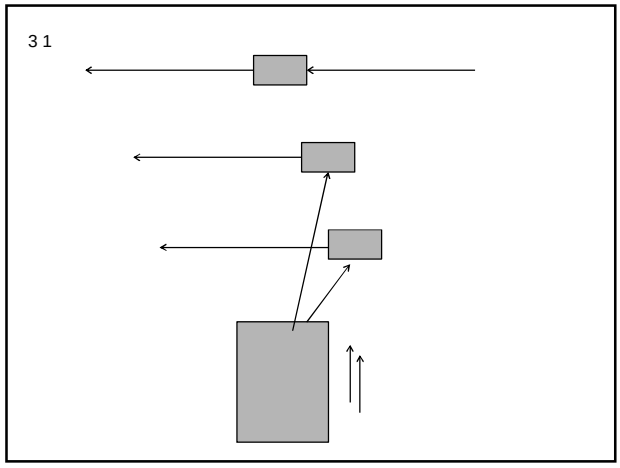
2 1



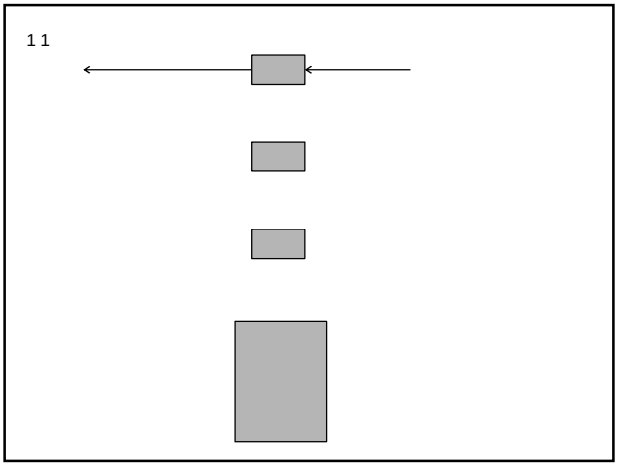
01



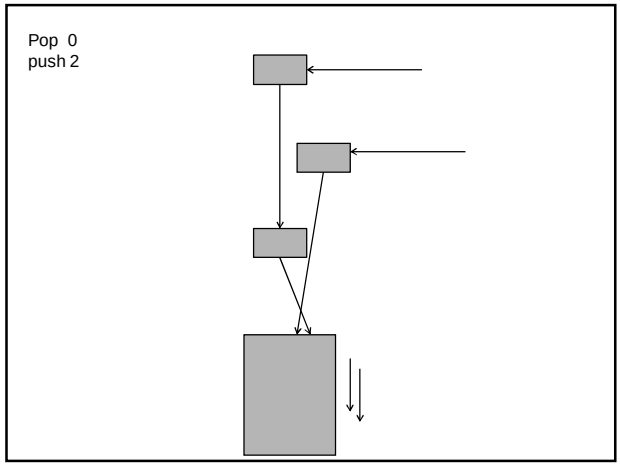
3 1



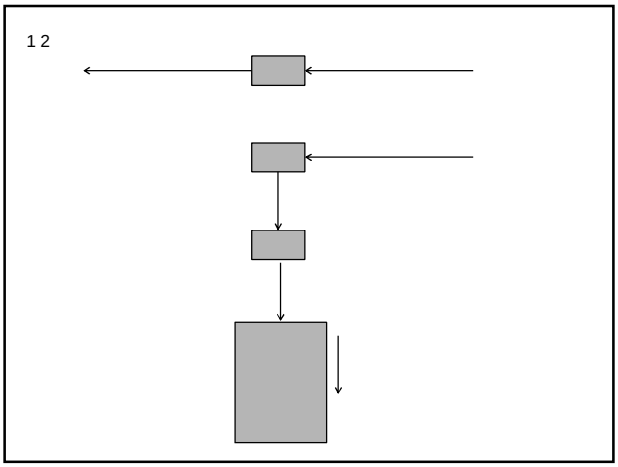
1 1



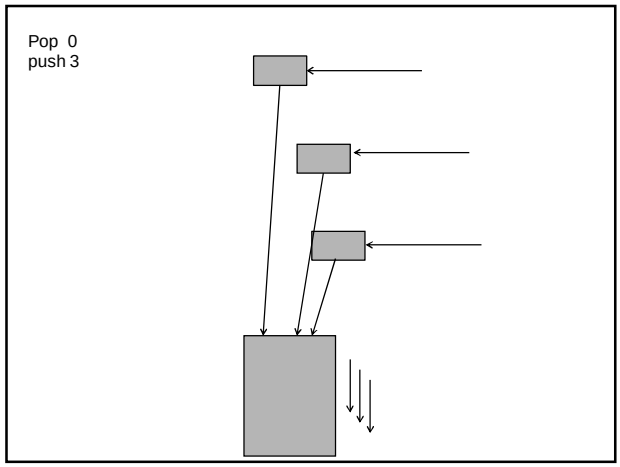
Pop 0
push 2



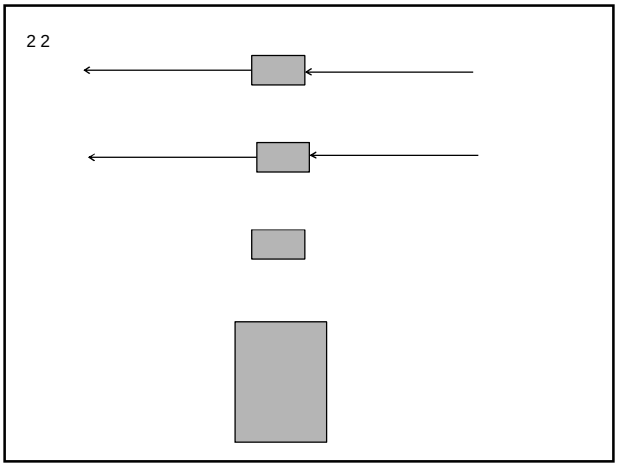
1 2



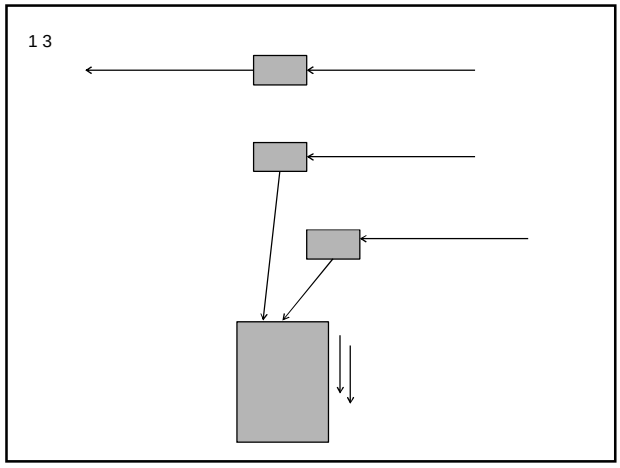
Pop 0
push 3



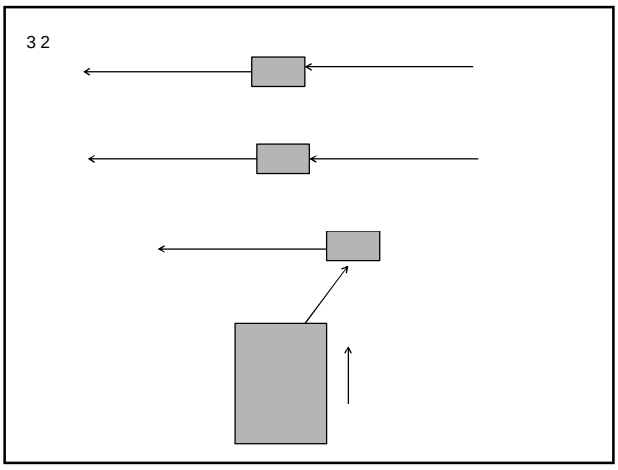
2 2



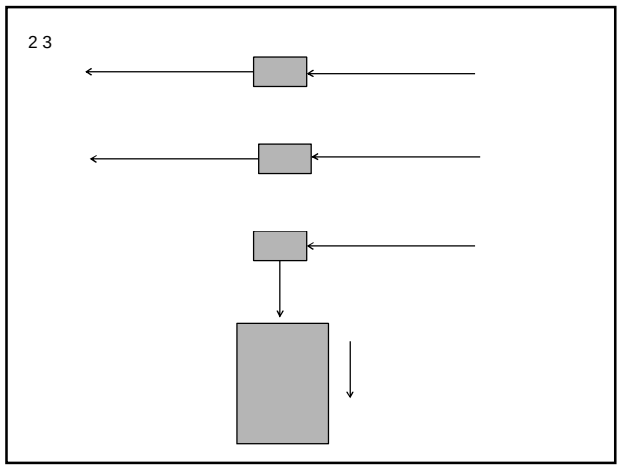
1 3



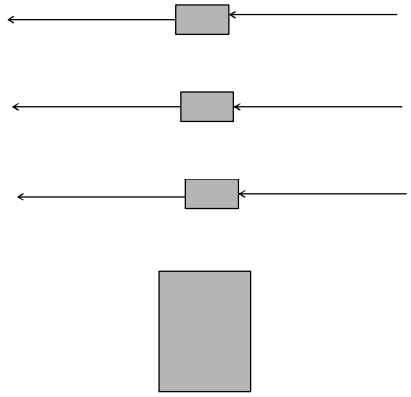
3 2



2 3

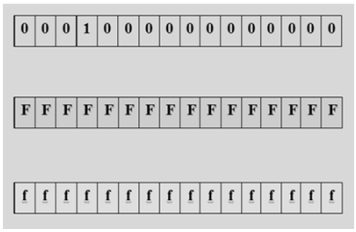


33



Instruction Call

- Idem que BRA, mémorise l'adresse de retour dans l'entity mémoire



Le contrôle de flot

- **Branchement relatif BR**

10 bits pour le déplacement. Ce déplacement est un entier en complément à deux qui s'ajoute à la valeur du compteur ordinal pointant sur cette instruction de branchement

- **Branchement relatif si Zero BZ**

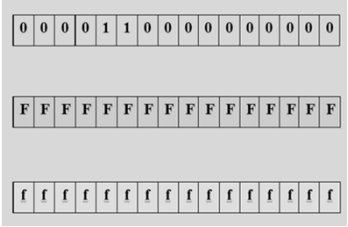
- **Branchement relatif si Non Zero BNZ**

- Return permet le retour d'appel de procédure par restauration du compteur ordinal

- Halt permet d'arreter le processeur dans l'état courant

Branchement absolu

- range la valeur codées sur les deux mots de 16 bits suivants dans le CO



Les litéraux

- LIT permet de manipuler des nombres en complément à deux sur 12 bits.

- Plusieurs LIT de suite avec des decalages et AND logique permettent de construire des mots de plus de 12 bits : il faut instancier l 'IP!!

Gestion du parallélisme

- SPMD déclenche un programme sur tous les Homade connectés à une adresse explicite

  - Adresse de début ' IIIIIIIIIII00' sur 16 (15) bits
- Wait : barrière de synchro entre tous les Homade connectés

La syntaxe de l'assembleur

- Homadeasm :=
  <ID_list>
  Program
  [[<function_decl_list>] begin]
  <instruction_list>
  endprogram
- ID_list:= VARIABLE <id_Function>
- function_decl := <id_Function> FUNCTION
  <instruction_list>
  RETURN

Un programme en ROM

```
13  architecture rom_io of Prom64 is
14  type rom_array is array (NATURAL range <>) of std_logic_vector ( 63 downto 0 ) ;
15  constant rom : rom_array := (
16
17  x"0C00_0000_0010_200F",--0
18  x"A402_8804_8003_1400",--4
19  x"A002_22ff_2fff_C83E",--8
20  x"A401_1400_FFFF_FFFF",--C
21  x"200A_A007_8806_8007",--10
22  x"2008_C82C_200A_A007",--14
23  x"B008_8806_8007_C826",--18
24  x"1000_0000_0008_8806",--1C
25  x"A823_B008_0BF5_A000",--20
26  x"A000_8806_A823_B008",--24
27  x"0BE9_A000_1E00_FFFF"--28
28
29  );
```

Suite

- instruction := <id_Function> CALL !
  <IP_16_BITS> !
  IF < instruction_list>
  [ELSE < instruction_list >]
  ENDIF !
  REPEAT < instruction_list > {AGAIN !UNTIL }!
  DO < instruction_list > LOOP !
  <Hexa_constant> BR !
  <Hexa_constant> BNZ !
  <Hexa_constant> BZ !
  <Hexa_constant> BA !
  HLT !
  NILL !
  NOP !
  <Hexa_constant> LIT

Un assembleur post fixé

- A la Forth
- On utilise Forth pour écrire du Homade
- Installation de GNU Forth
- Un fichier de configuration
- Un fichier de code asm Homade
- Produit le code VHDL pour la ROM

Exemples

```
variable read
program
read function
        f lit
        waitbtnPush
return
begin
repeat
        read call
.....
again
hlt
endprogram
```