Mini-craps

abus, bbus acheminent les opérandes sources des instructions vers l'ual
dbus achemine les données entre les modules
dbusin permet de sélectionner une entrée sur dbus (si plusieurs, court circuit !)
flags : état du résultat dans l'ual N (négatif) Z (zéro) V (débordement sur le bit de signe) C (retenue/emprunt final)
areg, breg, dreg indiquent les n° du registre qu'on veut lire sur les sorties a, b, datain

Schéma d'exécution d'un programme :
PC <- adresse première instruction
Répéter
    lire le code de l'instruction courante // IR <- [PC]
    exécuter cette instruction // peut nécessiter plusieurs étapes
    passer à l'instruction suivante // PC <- PC + 1 ou PC <- adresse de branchement
Jusqu'à Fin du programme

Registres

CRAPS	
r0	constante 0
r1-r19	registres généraux
r28	adresse de return après un call
r29	pointeur de pile (%sp)
r30	adresse de l'instruction en cours (%pc)
r31	instruction en cours (%ir)

mini-craps	
r0	constante 0
r1	constante 1
r2-11	registres généraux
r12-13	registres séquenceur
r14	pc
r15	ir

Doc CRAPS

simm13 must be in [-4096, +4095], *imm5* must be in [0, 31] opérationcc = opération avec modification de flags

add/cc
addcc %rs1, %rs2, %rd / addcc %rs1, simm13, %rd
Add the two operands %rs1 and %rs2 / simm13 and put the result in register %rd. The C flag is not added to the arguments.
Modified flags : N, Z, V, C
Example: addcc %r1, 5, %r1 Add 5 to %r1 and set flags N, Z, V, C

sub/cc
subcc %rs1, %rs2, %rd / subcc %rs1, simm13, %rd
Subtract %rs2 or simm13 from %rs1 and put the result in register %rd.
Modified flags : N, Z, V, C
Example: subcc %r1, 5, %r1 Compute %r1 - 5, put the result into %r1 and set flags N, Z, V, C

umulcc
umulcc %rs1, %rs2, %rd / umulcc %rs1, simm13, %rd
Unsigned multiply. Take only the 16 lowest bits of %rs1 and %rs2 or simm13, perform an unsigned multiplication between them, and put the result in register %rd.
Modified flags : N, Z
Example: umulcc %r1, %r1, %r1 Elevate %r1 at power 2 and set flags N, Z

sethi
sethi imm24, %rd
Set High. This instruction is generally used through the synthetic instruction set. Sets the 24 highest-order bits of %rd with immediate unsigned value imm24.
Modified flags: None

and/cc

andcc %rs1, %rs2, %rd / andcc %rs1, simm13, %rd

Performs a bit-wise AND between the two operands %rs1 and %rs2 / simm13 and put the result in register %rd.

Modified flags : N, Z

Example: andcc %r1, %r2, %r3 Performs a bit-wise AND between %r1 and %r2 and put the result in %r3

or/cc

orcc %rs1, %rs2, %rd / orcc %rs1, simm13, %rd

Performs a bit-wise OR between the two operands %rs1 and %rs2 / simm13 and put the result in register %rd.

Modified flags: N, Z

Example: orcc %r1, 1, %r1 Set bit #0 of %r1 and leave all others unchanged

xor/cc

xorcc %rs1, %rs2, %rd / xorcc %rs1, simm13, %rd

Performs a bit-wise XOR between the two operands %rs1 and %rs2 / simm13 and put the result in register %rd.

Modified flags : N, Z

Example: xorcc %r1, %r2, %r3 Performs a bit-wise XOR between %r1 and %r2 and put the result in %r3

b(cond)

b(cond) label

If condition cond is verified, branch to label; otherwise proceed to next instruction.

instruction	*description*	*test*
ba	always	1
be/bz/beq	equal	Z
bne/bnz	not equal	not Z
bcs	carry set	C
bcc	carry clear	not C
bvs	overflow set	V
bvc	overflow clear	not V
bpos/bnn	positive	not N
bneg/bn	negative	N
bg(u)/bgt	greater	not (Z or (N xor V))
bge(u)	greater or equal	not (N xor V)
bl(u)/blt	less	N xor V
ble(u)	less or equal	Z or (N xor V)

reti

"Return interrupt": this instruction must be used at the end of an interrupt subprogram.

Modified flags: None

ld

ld [%rs1 + %rs2], %rd / ls [%rs1 + simm13], %rd

Load the content of a memory location and copy it into register %rd. The address at which memory is read is obtained by adding %rs1 and %rs2 / simm13.

Modified flags: None

Example: ld [%r1 - 2], %r3 Compute %r1 - 2, read memory at this address and copy memory contents into %r3.

st

st %rd, [%rs1 + %rs2] / st %rd, [%rs1 + simm13]

Store %rd into a memory location, whose address is obtained by adding %rs1 and %rs2 / simm13.

Modified flags: None

Example: ld %r3, [%r1 + %r2] Compute %r1 + %r2 and store at this address the value of %r3.

sll

sll %rs1, %rs2, %rd / sll %rs1, imm5, %rd

Shift Logical Left. Left-shift the bit array of %rs1 of %rs2 (or simm13) positions and put the result in register %rd. Zeros are inserted on the %rs2 / imm5 lowest-order bits of %rd.

Modified flags: None

Example: sll %r1, 7, %r2 Shift left the bits of %r1 and put the result in %r2. Zeros are inserted on the 7 lowest-order bits of %r2.

slr

slr %rs1, %rs2, %rd / slr %rs1, imm5, %rd

Shift Logical Right. Right-shift the bit array of %rs1 of %rs2 (or simm13) positions and put the result in register %rd. Zeros are inserted on the %rs2 / imm5 highest-order bits of %rd.

Modified flags: None

Example: slr %r1, 7, %r2 Right-shift the bits of %r1 and put the result in %r2. Zeros are inserted on the 7 highest-order bits of %r2.

Instructions synthétiques (raccourcis)

instruction	*description*	*implementation*
clr %ri	clear %ri	orcc %r0, %r0, %ri
mov %ri, %rj	copy %ri to %rj	orcc %ri, %r0, %rj
inc %ri	increment %ri, no flag modified	add %ri, 1, %rj
inccc %ri	increment %ri, flags modified	addcc %ri, 1, %rj
dec %ri	decrement %ri, no flag modified	sub %ri, 1, %rj
deccc %ri	decrement %ri, flags modified	subcc %ri, 1, %rj
set imm32, %ri	copy imm32 to %ri	sethi imm32[31..8], %ri ; orcc %ri, imm32[7..0], %ri
setq simm13 %ri	copy simm13 to %ri	orcc %r0, simm13, %ri
cmp %ri, %rj	compare %ri to %rj	subcc %ri, %rj, %r0
cmp %ri, simm13	compare %ri to simm13	subcc %ri, simm13, %r0
tst %ri	test sign and zero of %ri	subcc %ri, %r0, %r0
negcc %ri	replace %ri by its opposite	subcc %r0, %ri, %ri
nop	no operation	sethi 0, %r0
call label	call terminal subprogram	or %r0, %r30, %r28 ; ba label
ret	return from terminal subprogram	orcc %r30, %r0, %r28
push %ri	push %ri into memory stack	sub %r29, 1, %r29 ; st %ri, [%r29+%r0]
pop %ri	pop word from memory stack and put it in %ri	ld [%r29+%r0], %ri ; add %r29, 1, %r29

SWITCHES = 0x90000000	set SWITCHES, %r19
LEDS = 0xB0000000	set LEDS, %r20
PILE = 0x200	set PILE, %sp

Spécification d'un sous-programme :

// rôle : …

// IN paramètre1 : …… dans r1

// …

// OUT résultat : ..... dans r3

↗	RAM	reg	pile
RAM	x	ld	x
reg	st	mov	push
pile	x	pop	x
const	x	set	x
0	x	clr	x

Programmes

PGCD de A et B stockés en mémoire

```
pgcd :  set NbA, %r2
        ld [%r2], %r2
        set NbB, %r3
        ld [%r3], %r3
tq :    cmp %r2, %r3
        beq ftq
        bleu sinon
        sub %r2, %r3, %r2
        ba fsi
sinon : sub %r3, %r2, %r3
fsi :   ba tq
ftq :   set pgcd, %r4
        st %r2, [%r4]
stop :  ba stop
NbA :   .word 90
NbB :   .word 175
pgcd :  .word 0
```

Addition de 2 variables

```
debut : set var1, %r2    // r2 <- adresse de var1
        set var2, %r3    // r3 <- adresse de var2
        set res, %r4     // r4 <- adresse de res
        ld [%r2], %r5    // r5 <- valeur mémoire de var1
        ld [%r3], %r6    // r6 <- valeur mémoire de var2
        add %r5, %r6, %r7        // %r7 <- %r5 + %r6
        st %r7, [%r4]    // st = store : [Résultat] <- r7
Stop :  ba Stop
var1 :  .word 123 // réservation d'un mot mémoire initialisé
var2 :  .word 654
res :   .word 0
```

Factorielle TantQue

```
fact :  set 1, %r2
        set N, %r3
        ld [%r3], %r3
tq :    cmp %r3, 1
        bleu ftq
        umulcc %r2, %r3, %r2
        sub %r3, 1, %r3
        ba tq
ftq :   set res, %r4
        st %r2, [%r4]
stop :  ba stop
N :     .word 6
res :   .word 0
```

Factorielle Répéter

```
fact :  set 1, %r2
        set N, %r3
        ld [%r3], %r3
        cmp %r3
        bleu fin
repet : umulcc %r2, %r3, %r2
        sub %r3, 1, %r3
        cmp %r3, 1
        bgu repet
fin :   set res, %r4
        st %r2, [%r4]
stop :  ba stop
N :     .word 6
res :   .word 0
```

Factorielle récursive

```
        set N, %r1
        call fact
stop :  ba stop
fact :  cmp %r2, 0
        beq init
        cmp %r1, 0
        beq retour
        umulcc %r1, %r2, %r2
        dec %r1
        push %r28
        call fact
        pop %r28
retour :ret
init :  set 1, %r2
        ba fact
```

Factorielle TantQue 2

```
        set N, %r1
        call fact
stop :  ba stop
fact :  push %r1
        set 1, %r2
tq :    cmp %r1, 1
        bleu retour
        umulcc %r1,
              %r2, %r2
        dec %r1
        ba tq
retour :pop %r1
        ret
```

Copier tab1 dans tab2

```
        N = 10
        set Tab1, %r1
        set Tab2, %r2
        clr %r3
Tq :    cmp %r3, N
        bgeu Stop
        ld [%r1+%r3], %r4 // %r2+%r3=adresse de tab(r3)
        st %r4, [%r2+%r3]
        add %r3, 1, %r3 // index <- index + 1
        ba Tq
Stop :  ba Stop
Tab1 :  .word 10, 9, 8, 7, 1, 6, 5, 4, 3, 2, 1
Tab2 :  .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

Inverser tab1 dans tab2

```
        N = 10
        set Tab1, %r1
        set Tab2, %r2
        set N, %r5
        clr %r3
Tq :    cmp %r3, N
        bgeu Stop
        ld [%r1+%r3], %r4
        st %r4, [%r2+%r5]
        sub %r5, 1, %r5
        add %r3, 1, %r3
        ba Tq
Stop :  ba Stop
```

Copier les éléments de Chaîne dans la pile

```
        PILE = 0x200 // fond de pile à l'adresse 0x200
        set PILE, %sp // initialisation du pointeur de pile
        set Chaine, %r1
        clr %r2 // %r2 <- 0 : nombre d'éléments
Repet : ld [%r1], %r3
        cmp %r3, %r0 // r3 ? 0
        beq Stop
        push %r3 // %r3 -> sommet de pile
        inc %r2 // add %r2, 1, %r2
        inc %r1 // adresse du prochain élément
        ba Repet
Stop : ba Stop
Chaine : .word 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0
```

Tri sélection

```
        N = 10
        set Tab1, %r1
        set N, %r7
        call Tri_selection
Stop :  ba Stop
Tri :   clr %r2 // i
        clr %r3  // j
        clr %r5
        clr %r6 // indice du min
Tq1 :   set 2, %r8
        subcc %r7, %r8, %r8
        cmp %r2, %r8
        bgu Fin
        ld [%r1+%r2], %r5 // minimum_courant <- Tab[i]
        mov %r2, %r6 // attribution valeur i à indice du min
        mov %r2, %r3 // attribution valeur i à j
        inc %r3
Tq2 :   cmp %r3, %r7
        bgeu Minimum_trouve
        ld [%r1+%r3], %r4
        cmp %r4, %r5
        bge Min
        mov %r4, %r5
        mov %r3, %r6
Min :   add %r3, 1, %r3 // index <- index + 1
        ba Tq2
Minimum_trouve:
        ld [%r1+%r2], %r9 // tab[i]
        st %r5, [%r1+%r2]
        st %r9, [%r1+%r6]
        inc %r2
        ba Tq1
Fin :   ret
```

Trouver le min d'un tableau

```
        N = 11
        set Tab1, %r1
        ld [%r1], %r5
Tq :    cmp %r3, N
        bgeu Stop
        ld [%r1+%r3], %r4
        add %r3, 1, %r3
        cmp %r5, %r4
        bgeu Rmin // Remplacer Min
        ba Tq
Stop :  ba Stop
Rmin :  mov %r4, %r5
        mov %r3, %r6
        ba Tq
Tab1 :  .word 10, 9, 8, 7, 1, 6, 5, 4, 3, 2, 1
```

Test du tri sélection

```
Test :  set Tab_attendu, %r2
        clr %r3
Test_corps :
        set 2, %r8
        subcc %r7, %r8, %r8
        cmp %r2, %r8
        bgu Test_vrai
        ld [%r1+%r4], %r4
        ld [%r2+%r3], %r5
        ld [%r1+1], %r11
        …
        ld [%r1+9], %r20
        cmp %r4, %r5
        bne Test_faux
        inc %r3
        ba Test_corps
Test_faux :
        set 0, %r9
        ret
Test_vrai :
        set 1, %r9
        ret
Tab1 :  .word 8, 9, 1, 7, 5, 4, 3, 2, 10, 6
Tab_attendu :
        .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Palindrome binaire (méthode invversion)

```
palindrome_bin_inv:
        push %r3 // sauvegarde des registres modifiés
        push %r4
        push %r5
        push %r6
        set 1, %r2 // résultat <- 1
        set 32, %r3 // nombre de bits
        mov %r1, %r4 // copie du nombre
        clr %r5 // nombre inverse
pbi_bcle:
        andcc %r2, %r4, %r6 // isoler bit0
        sll %r5, 1, %r5 // r5 *2
        or %r5, %r6, %r5 // + bit0
        slr %r4, 1, %r4 // décaler d'une position -< droite
        deccc %r3
        bne pbi_bcle
        cmp %r1, %r5 // nombre intial ?= nombre inverse
        beq pbi_ret // palindrome : r2 est déjà =1
        clr %r2 // non palindrome
pbi_ret:
        pop %r6 // restauration des registres
        pop %r5
        pop %r4
        pop %r3
        ret
```

Palindrome binaire (méthode symétrie)

```
palindrome_bin_sym:
        push %r3 // compteur 0-15
        push %r4 // compteur 31-16
        push %r5 // premiers bits
        push %r6 // derniers bits
        push %r7 // 1
        set 1, %r7
        set 31, %r4
pbs_bcle:
        slr %r1, %r3, %r5
        andcc %r7, %r5, %r5
        slr %r1, %r4, %r6
        andcc %r7, %r6, %r6
        cmp %r5, %r6
        bne pbs_ret
        inccc %r3
        deccc %r4
        cmp %r3, 16
        bne pbs_bcle
        set 1, %r2
pbs_ret:
        pop …. // restauration des registres
        ret
```

Crible d'Eratosthène

```
        N = 110
        PILE = 0x200
        set PILE, %sp
        set N, %r2
        set Tab_premiers, %r1
        call Initialiser_tab_premiers
        call eratosthene
Stop :  ba Stop
Initialiser_tab_premiers:
        clr %r3
        st %r3, [%r1]        // Tab_premiers[0] = 0
        inc %r3
        st %r3, [%r1+4]      // Tab_premiers[1] = 0
        set 2, %r3
Init_boucle:
        cmp %r3, %r2
        bgeu Init_fin
        st %r3, [%r1+%r3]    // Tab_premiers[i] = I
        inc %r3
        ba Init_boucle
Init_fin:
        ret
```

```
Eliminer_multiples: mov %r3, %r4
Elim_boucle:
        add %r3, %r4, %r4     // k = k + i
        cmp %r4, %r2
        bgeu Elim_fin
        clr %r5
        st %r5, [%r1+%r4]    // Tab_premiers[k] = 0
        ba Elim_boucle
Elim_fin: ret
eratosthene: set 2, %r3
era_boucle:
        umulcc %r3, %r3, %r4
        cmp %r4, %r2
        bgeu era_fin
        ld [%r1+%r3], %r4 // Charger Tab[i] dans %r4
        cmp %r4, 0
        be era_suivant
        push %r28
        call Eliminer_multiples
        pop %r28
era_suivant:
        inc %r3
        ba era_boucle
era_fin: ret
Tab_premiers: .word (N+1)
```

Lire switches sur les leds

```
boucle: ld [%r19], %r1
        st %r1, [%r20]
        ba boucle
```

Afficher 8bits sur les leds poids faible

```
        set N, %r1
afficher_leds_7_0:
        and %r1, 0x0FF, %r1
        set 0xB0000000, %r20
        st %r1, [%r20]
        ret
```

Afficher 8bits sur les leds poids fort

```
        set N, %r1
afficher_leds_15_8:
        and %r1, 0x0FF, %r1
        sll %r1, 8, %r1
        set 0xB0000000, %r20
        st %r1, [%r20]
        ret
```

Compter le nb d'interruptions

```
        ba progp // instruction à l'@ 0
handler_IT:
        push %r1 // handler d'IT à l'@ 1
        push %r2
        push %r20
        set LEDS, %r20
        set compteur_IT, %r1
        ld [%r1], %r2
        inc %r2
        st %r2, [%r20]
        st %r2, [%r1]
        pop %r20
        pop %r2
        pop %r1
        reti // retour dans le prog interrompu
// forcément après le handler d'IT
progp:
        set PILE, %sp
        set compteur_IT, %r1
        st %r0, [%r1]
pbcle:
        nop
        ba pbcle
compteur_IT:
        .word 0
```

Commutateur de tâches à 16 programmes

```
        PILE0 = 0x200
        …
        PILE15 = 0x1700
        NB_PROGS = 16
        ba start
handler_IT:
        // Save ts les r de travail dans la pile du processus suspendu
        push %r1
        …
        push %r10
        push %r28
         // Save pointeur pile dans emplacement qui lui est associé
        st %sp, [%r19 + %r17]
        // Incrémenter le numéro du processus courant
        inc %r17
        cmp %r17, %r18
        bl skip
        clr %r17
skip :
        // Récupérer le pointeur de pile du processus élu
        ld [%r19 + %r17], %sp
        // Récupérer tous les registres de travail du processus élu
        pop %r28
        …
        pop %r1
        // Reprendre l'exécution du processus élu
        reti
start:
        set Tab_sp, %r19
        set Tab_progs, %r16
        set NB_PROG, %r18
        set 1, %r17
boucle_init:
        ld [%r19 + %r17], %sp // load la PILE i
        ld [%r16 + %r17], %r11 // load le PC i
        push %r11 // push du PC
        push %r0 // push de NZVC
        push %r0 …x10 // push r1-r10
        push %r0 // push de r28
        st %sp, [%r19 + %r17]
        inc %r17
        cmp %r17, %r18 // si i < NB_PROGS,
        bl boucle_init // on boucle
        // Initialise au prog 0
        clr %r17
        set PILE0, %sp
        prog0 : … prog1 : … prog15 : …

        Tab_sp :
                .word PILE0, …, PILE15
        Tab_progs:
                .word prog0, …, prog15
```

TP mini-craps

module **registres** (rst, clk, areg[3..0], breg[3..0], dreg[3..0], datain[31..0] : a[31..0], b[31..0], ir[31..0], pc[31..0])

decoder4to16 (areg[3..0] : asel[15..0])
decoder4to16 (breg[3..0] : bsel[15..0])
decoder4to16 (dreg[3..0] : dsel[15..0])

// L'entrée 'dreg' indique le n° du registre dans lequel on souhaite écrire l'entrée 'datain'
reg32_T (rst, clk, dsel[2], datain[31..0] : r2[31..0])
reg32_T (rst, clk, dsel[3], datain[31..0] : r3[31..0])
reg32_T (rst, clk, dsel[4], datain[31..0] : r4[31..0])
reg32_T (rst, clk, dsel[5], datain[31..0] : r5[31..0])
reg32_T (rst, clk, dsel[6], datain[31..0] : r6[31..0])
reg32_T (rst, clk, dsel[7], datain[31..0] : r7[31..0])
reg32_T (rst, clk, dsel[12], datain[31..0] : r12[31..0])
reg32_T (rst, clk, dsel[13], datain[31..0] : r13[31..0])
reg32_T (rst, clk, dsel[14], datain[31..0] : r14[31..0])
reg32_T (rst, clk, dsel[15], datain[31..0] : ir[31..0])

// L'entrée 'areg' indique le n° du registre qu'on souhaite lire sur la sortie 'a'
a[31..0] = asel[0] * "00000000000000000000000000000000" + asel[1] * "00000000000000000000000000000001" + asel[2] * r2[31..0] + asel[3] * r3[31..0] + asel[4] * r4[31..0] + asel[5] * r5[31..0] + asel[6] * r6[31..0] + asel[7] * r7[31..0] + asel[12] * r12[31..0] + asel[13] * r13[31..0] + asel[14] * r14[31..0] + asel[15] * ir[31..0]
// L'entrée 'areg' indique le n° du registre qu'on souhaite lire sur la sortie 'a'
b[31..0] = bsel[0] * "00000000000000000000000000000000" + bsel[1] * "00000000000000000000000000000001" + bsel[2] * r2[31..0] + bsel[3] * r3[31..0] + bsel[4] * r4[31..0] + bsel[5] * r5[31..0] + bsel[6] * r6[31..0] + bsel[7] * r7[31..0] + bsel[12] * r12[31..0] + bsel[13] * r13[31..0] + bsel[14] * r14[31..0] + bsel[15] * ir[31..0]
// On lit r14 sur la sortie 'pc'
pc[31..0] = r14[31..0]

end module

module **ual** (a[31..0], b[31..0], cmd[3..0] : s[31..0], N, Z, V, C)

ucmp4 (cmd[3..0], "0000" : sup1, addition)
ucmp4 (cmd[3..0], "0001" : sup2, soustraction)
ucmp4 (cmd[3..0], "1100" : sup3, extend)

addsub32(a[31..0], b[31..0], soustraction : addsub[31..0], C, V)

sig_extend24[22..0] = a[22..0]
sig_extend24[30..23] = /a[23]*"00000000" + a[23]*"11111111"
sig_extend24[31] = a[23]

s[31..0] = addsub[31..0]*(addition + soustraction) + sig_extend24[31..0]*extend
ucmp32 (s[31..0], "00000000000000000000000000000000" : sup4, Z)
N = s[31]

end module

Exam corrigé

1. Implanter « 0 cop(3) | rdest(4) | rs1(4) | rad2(4) | 1 … 15 bits libres … »

Il faut commencer par lire l'opérande 2 en mémoire et le mettre dans un registre car on ne peut faire d'op arithmétique qu'entre 2 registres !

transition	condition	action
fetch > decode	1	IR <- [PC]
decode > load_op	/IR[31]	r12 <- [rad2]
load_op > pcplus1	1	rdest <- rs1 op r12
pcplus1 > fetch	1	PC <- PC + 1

	areg	breg	dreg	ualcmd	dbusin	write
fetch > decode	1110	0000	1111	0000	10	0
decode > load_op	IR[19..16]	0000	1100	0000	10	0
load_op > pcplus1	IR[23..20]	1100	IR[27..24]	IR[31..28]	01	0

1bis. Implanter « 0 cop(3) | rdest(4) | rs1(4) | rs2(4) | … 16 bits libres … »

transition	condition	action
decode > pcplus1	/IR[31]	rdest <- rs1 op rs2

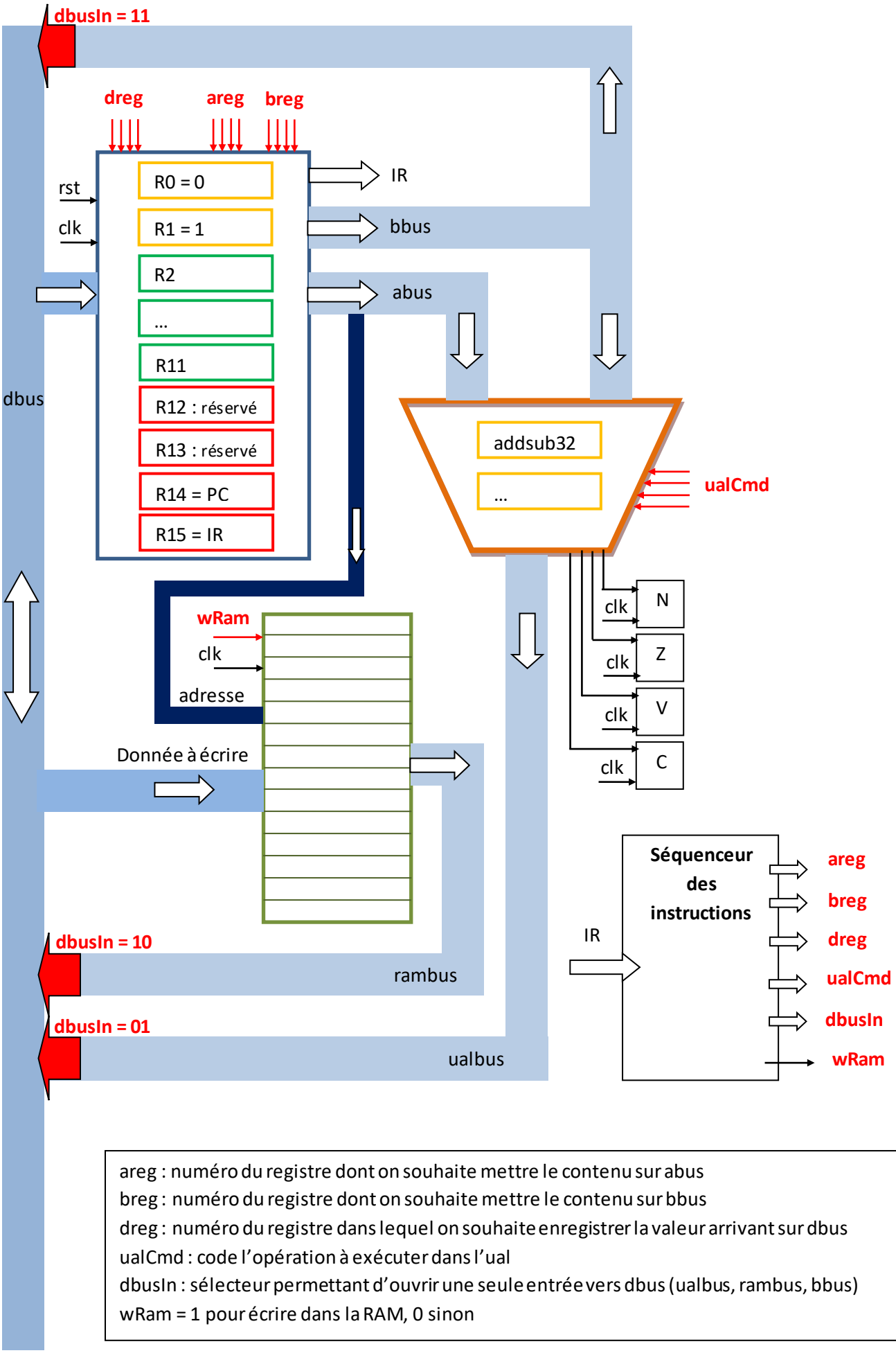
	areg	breg	dreg	ualcmd	dbusin	write
decode > pcplus1	IR[23..20]	IR[19..16]	IR[27..24]	IR[31..28]	01	0

2. Écrire un s-p comptant le nb d'occurrences d'une valeur dans un tableau, et un programme de test

```
        PILE = 0x100
        M = 10
        set PILE, %sp
        set tab, %r1
        set M, %r2
        set 2, %r3
        call nb_occurrences
stop:   ba stop
tab:    .word 3, 2, 7, 5, 2, 11, 6, 9, 4, 2
nb_occurrences:
        push %r5
        push %r6
        clr %r4 // nombre d'occurrences
        clr %r5 // index
loop:   cmp %r5, %r2
        bgeu end_loop
        ld [%r1+%r5], %r6
        cmp %r6, %r3
        bne suite
        inc %r4
suite:  inc %r5
        ba loop
end_loop:
        pop %r6
        pop %r5
        ret
```

3. Écrire un programme qui incrément modulo x et visuale la valeur courante sur les leds lors d'une IT

```
        X = 6
        PILE = 0x100
        LEDS = 0xB0000000
        ba progp
        handler :
                push %r20
                set LEDS, %r20
                st %r1, [%r20]
                pop %r20
                reti
        progp :
                set PILE, %sp
                clr %r1
        boucle :
                cmp %r1, X-1
                bne incrementer
                clr %r1
                ba boucle
        incrementer :
                inc %r1
                ba boucle
```



areg : numéro du registre dont on souhaite mettre le contenu sur abus
breg : numéro du registre dont on souhaite mettre le contenu sur bbus
dreg : numéro du registre dans lequel on souhaite enregistrer la valeur arrivant sur dbus
ualCmd : code l'opération à exécuter dans l'ual
dbusIn : sélecteur permettant d'ouvrir une seule entrée vers dbus (ualbus, rambus, bbus)
wRam = 1 pour écrire dans la RAM, 0 sinon