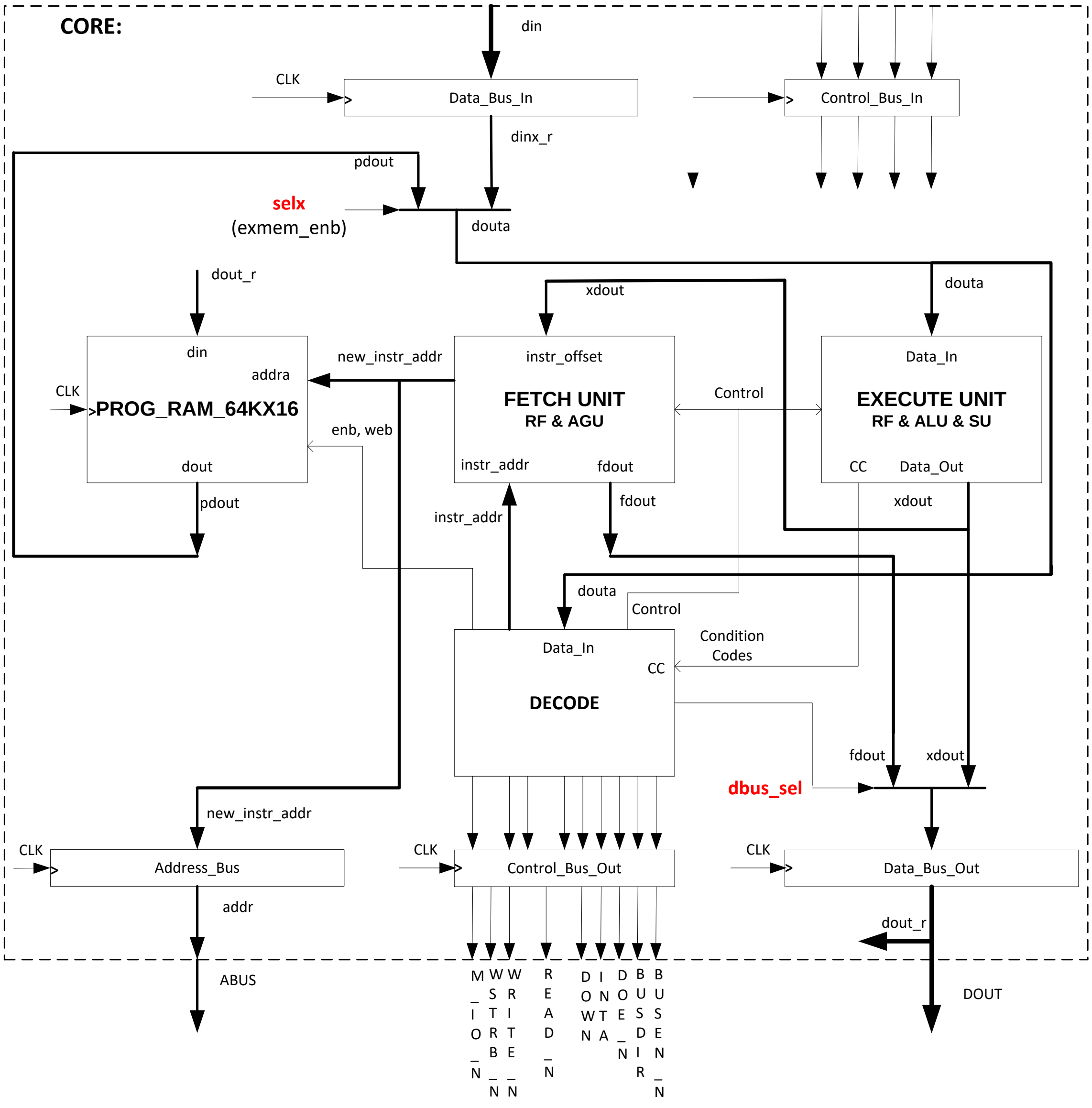
OPUS16

16 BIT CENTRAL PROCESSING UNIT



Version 1.0 Rev.1.0

Guillermo H. ROMERO / Copyright 2002..2022

GiLL Romero : Romero@IEEE.org

OPUS1

======

Opus16 16 bits integer processor unit

========================

ISA: ARCHITECTURE DEFINITION

===========================

AUTHOR: Guillermo H. Romero (GiLL)

EMAIL: Romero@ieee.org
PHONE: (617)-905-0508

FIRST ISSUE: 08/01/2002

HISTORY:

08/01/2002
First version of an 8 bit data/ 12 bit address processor.

08/01/2004
Second version of a 16 bits data/address processor.

08/01/2021
Final version of the Opus16 custom processor.

GENERAL DESCRIPTION

INTRODUCTION

The Opus16 custom processor is a 16 bits integer processor, with about 80 instructions and 4 interrupts, instructions are very simple but very useful.
The main object of this design is to program in Assembler language easy, fast and it does not require a software environment but some careful process that I’ll explain later.

1. *ISA: Instruction Set Architecture*
2. *Opus16 pin out*
3. *Opus16 architecture*
4. *Hardware interrupt control*
5. *Opus16 Internal Structure*
6. *Examples*

The following documentation provides an architectural definition of this custom processor. It should be noted that this is not a fully 100% tested CPU, as much as I simulated with different standar simulators and also thoroughly tested in hardware, there is a possibility that one particular instruction sequence can cause undesirable results. Said that, it is not complicated to fix the problem if exists.

Instruction structure:

rd is the destination register-------------------------- rd
rs is the source register-------------------------------- rs
rg is either source or destination -------------------- rg
pd is the destination pointer-------------------------- pd
ps is the source pointer-------------------------------- ps
Semicolon is the line terminator---------------------- ;
Double forward slash is a comment------------------ //
Preceding ‘h means an hexadecimal number------- ‘h1234
A parenthesis means content of --------------------- (p1)
P1 = address 1000 hex, content of address 1000 is 55
(p1) = 55 hex

For example :

```
Add        r1,r2;        // r1 = r1+r2
Addv       r3,’h0234;    // r3 = r3 + 0234 (hexadecimal)
Addv       r4,123;       // r4 = r4 + 0123 (decimal)
Addrp      r5,p2;        // r5 = r5 + p2
```

1) ISA: Instruction Set Architecture

Mnemonic	**opcode**	**words**	**order**	**example;**	**// comment**	
General Group						
NOP	00	1	0	nop;	// no operation	
Arithmeticgroup						
ADD	01	1	1	add rd,rs;	// rd = rd + rs	
ADC	02	1	1	adc rd,rs;	// rd = rd + rs + cy	
ADDV	03	2	1	addv rd,val;	// rd = rd + val	affects R0
ADDRP	04	1	1	addrp rd,ps;	// rd = rd + ps	affects R0
SUB	05	1	1	sub rd,rs;	// rd = rd - rs	
SBB	06	1	1	sbb rd,rs;	// rd = rd - rs - ~cy	
SUBV	07	2	1	subv rd,val;	// rd = rd – val	affects R0
SUBRP	08	1	1	subrp rd,ps;	// rd = rd – ps	affects R0
Logic group						
NOT	09	1	2	not [rs,]rd;	// rd = ~rs	
AND	0a	1	1	and rd,rs;	// rd = rd & rs	
ANDV	0b	2	1	andv rd,val;	// rd = rd & val	affects R0
OR	0c	1	1	or rd,rs;	// rd = rd | rs	
ORV	0d	2	1	orv rd,val;	// rd = rd | val	affects R0
XOR	0e	1	1	xor rd,rs;	// rd = rd ^ rs	
XORV	0f	2	1	xorv rd,val;	// rd = rd ^ val	affects R0
INC	10	1	1	inc rd;	// rd = rd + 1	
DEC	11	1	1	dec rd;	// rd = rd - 1	
CMPR	12	1	1	cmpr rd,rs;	// rd - rs	
CMPRV	13	2	2	cmprv rg,val;	// rg – val	affects R0
CMPRP	14	1	1	cmprp rg,pt;	// rg – pt	affects R0
SHL	15	1	1	shl type,rg;	// rd = rd << 1	
SHR	16	1	1	shr type,rg;	// rd = rd >> 1	
SHL4	17	1	1	shl4 rg;	// rd = rd << 4	
SHR4	18	1	1	shr4 rg;	// rd = rd >> 4	
Move Group						
MVRR	19	1	0	mvrr rs,rd;	// rd = rs	
MVRP	1a	1	0	mvrp rs,pd;	// pd = rs	
MVPR	1b	1	0	mvpr ps,rd;	// rd = ps	
MVPP	1c	1	0	mvpp ps,pd;	// pd = ps	
SWAP	1d	1	0	swap rg,rg;	// rg <--> rg	affects R0
SWAPP	1e	1	0	swap rg,pt;	// rg <--> pt	
Load Group						
LDCV	1f	2	1	ldcv cn,val;	// c[0/1/2/3] = val	
LDC	43	2	1	ldc cn,rs;	// c[0/1/2/3] = reg	
LDR	20	2	1	ldr rd,addr;	// rd = (addr)	
LDRV	21	2	1	ldrv rd,val;	// rd = val	

LDRP	22	1	1	ldrp	rd,ps;	// rd = (ps)
LDRX	23	2	1	ldrx	rd,ps,val;	// rd = (ps + val)
LDRPI	24	1	1	ldrpi	rd,ps;	// rd = (ps) => ps++
LDRPD	25	1	1	ldrpd	rd,ps;	// --ps, rd = (ps)
PULL	25	1	1	ldrpd	rd,ps;	// --ps, rd = (ps)
POP	25	1	1	ldrpd	rd,ps;	// --ps, rd = (ps)
LDMAM	26	1	0	ldmam	mn;	// m[0/1/2/3] access mode
LDPAGV	27	2	1	ldpagv	val;	// page address = value
LDPAG	42	1	0	ldpag	rs;	// page address = reg

Store Group

STR	28	2	0	str	rs,addr;	// (addr) = rs
STRP	29	1	0	strp	rs,pd;	// (pd) = rs
STRX	2a	2	0	strx	rs,pd,val;	// (pd + val) = rs
STRPI	2b	1	0	strpi	rs,pd;	// (pd) = rs => pd++
PUSH	2b	1	0	strpi	rs,pd;	// (pd) = rs => pd++
STRPD	2c	1	0	strpd	rs,pd;	// --pd, (pd) = rs

Pointer Group

LDP	2d	2	1	ldp	pd,addr;	// pd = (addr)
LDPV	2e	2	1	ldpv	pd,val;	// pd = val
LDPY	44	1	1	ldpy	pd,rs;	// pd = (pd+rs)
STP	2f	2	0	stp	ps,addr;	// (addr) = ps
INCP	30	1	1	incp	pt;	// pd = pd + 1
DECP	31	1	1	decp	pt;	// pd = pd - 1

I/O Group

INP	32	2	1	inp	rd,port;	// rd = (port) => i/o
INPP	33	1	1	inpp	rd,ps;	// rd = (ps) => i/o
OUTP	34	2	0	outp	rs,port;	// (port) = rs => i/o
OUTPP	35	1	0	outpp	rs,pd;	// (pd) = rs => i/o
//	36					// see DCJZ

Control Transfer : Branch Group, two's complements -2^15-1 to +2^15

BRA	37	2	0	bra	cond,addr;	// pc = pc + offset

Jump Group: full address range

DCJZ	36	2	1	dcjz	cn,addr;	// pc = pc + offset,cn=0
DCJNZ	38	2	1	dcjnz	cn,addr;	// pc = pc + offset,cn~=0
JMP	39	2	5	jmp	[t/f]#,addr	// pc = addr
JMPP	3a	1	1	jmpp	pt;	// pc = pt
JSR	3b	2	0	jsr	addr;	// pc = addr => --sp
RTS	3c	1	0	rts		// pc = (sp) => sp++
RTI	3d	1	0	rti		// pc = (sp) => sp++

Bit control

UFLAG	3e	1	5	uflag	[t/f]#;	// uflag bit n clr/set
UPORT	3f	1	5	uport	[t/f]#;	// oport bit n clr/set

// ***Test bits***

BIT	40	1	1	bit	rd,rs;	// rd = rd & rs	
BITV	41	2	1	bitv	rd,val;	// rd = rd & val	affects R0
//	42	// see LDPAG					
//	43	// see LDC					
//	44	// see LDPY					
SEI	45	1	1	sei;		// set interrupt, enable	
CLI	46	1	1	cli;		// clear interrupt, disable	
SWI	47	2	1	swi addr;		// pc = (sp) => sp++	

Special Instructions

HWI	5a	1	0	hwi ;		// hw interrupt, do not use
STOP	ff	1	0	stop;		// hw signal,

Notes:

Mnemonic

Lower case instruction

Words:

1 = one 16 bit opcode

2 = one 16 bit opcode and one address/data value, total of two words

Order:

Register order for src and dst fields: used by PERL compiler

0: src,0
1: 0,dst
2: src,src
3: src,src
4: src,src
5: special

Full opcode:

```
_________________________________________________________________
| 1 | 1 | 1 | 1 | 1 | 1 |   |   |   |   |   |   |   |   |   |   |
| 5 | 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|___|___|___|___|___|___|___|___|___|___|___|___|___|___|___|___|
|  s r c        |   d s t       |        opcode                 |
|_______________|_______________|_______________________________|
|  mode/type    |   d s t       |        opcode                 |
|_______________|_______________|_______________________________|
|      e x t e n d e d          |        opcode                 |
|_______________________________|_______________________________|
```

// Conditional and mofifier fields for branchs.

Cond	Mode	Instruction	opcode	condition	
none	0	bra label;	0037	always	
z	1	bra z,label;	1037	zero	
nz	9	bra nz,label;	9037	non zero	
e	1	bra e,label;	1037	equal	
ne	9	bra ne,label;	9037	non equal	
c	2	bra c,label;	2037	carry	
nc	a	bra nc,label;	a037	no carry	
lo	2	bra lo,label;	2037	lower	(unsigned)
hs	a	bra hs,label;	a037	higher or same	(unsigned)
s	3	bra s,label;	3037	sign	(negative)
ns	b	bra ns,label;	b037	no sign	(positive)
o	4	bra o,label;	4037	overflow	
no	c	bra no,label;	c037	no overflow	
ge	5	bra ge,label;	5037	greater than or equal	
lt	d	bra lt,label;	d037	less than	
gt	6	bra gt,label;	6037	greater than	
le	e	bra le,label;	e037	less than or equal	
hi	7	bra hi,label;	7037	higher	(unsigned)
ls	f	bra ls,label;	f037	lower or same	(unsigned)

// type for shifts

l	0	shl l,r2;	0215	shift left, link bit = bit15
k	1	shl k,r3;	1315	shift left, bit0 = link bit
r	3	shl r,r5;	3515	rotate left or right, link bit = msb or lsb
l	0	shr l,ra;	0a16	shift right, link bit = bit0
k	1	shr k,rb;	1b16	shift right, bit15 = link bit
a	2	shr a,r4;	2416	shift right arithmetic
r	3	shr r,rc;	3c16	rotate left or right, link bit = msb or lsb

// types for ldmam : load memory access mode, internal RAM vs external RAM

0	ldmam m0;	0026	internal read, internal write
1	ldmam m1;	1026	external read. internal write
2	ldmam m2;	2026	internal read, external write
3	ldmam m3;	3026	external read, external write

SPECIAL NOTES:

1) Pointer 0 (PC) is read only access.

```
Valid instructions are:
  stp   p0,address;
  any other store instruction involves p0
```

2) Register 0 (r0) is modified by the following instructions

Arithmetic Group

```
ADDV
ADDRP
SUBV
SUBRP
```

Logic Group

```
ANDV
ORV
XORV
CMPRV
CMPRP
BITV
```

Move Group

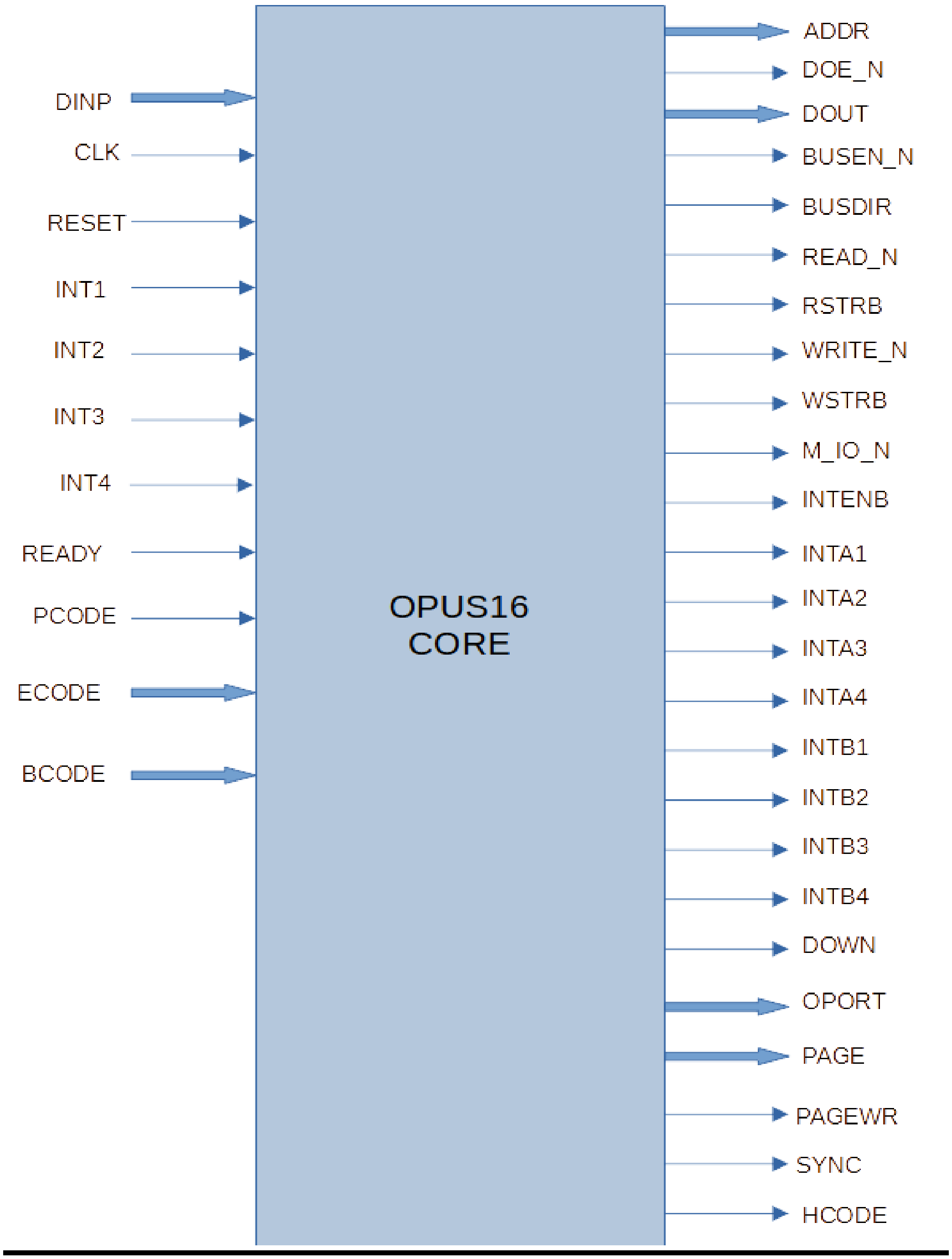
```
SWAP
```

3) LDMAM : load memory access mode

```
LDMAM m0 internal read, internal write 0  0 IRW
LDMAM m1 external read. internal write 0  1 XRD
LDMAM m2 internal read, external write 1  0 XWR
LDMAM m3 external read, external write 1  1 XRW
```

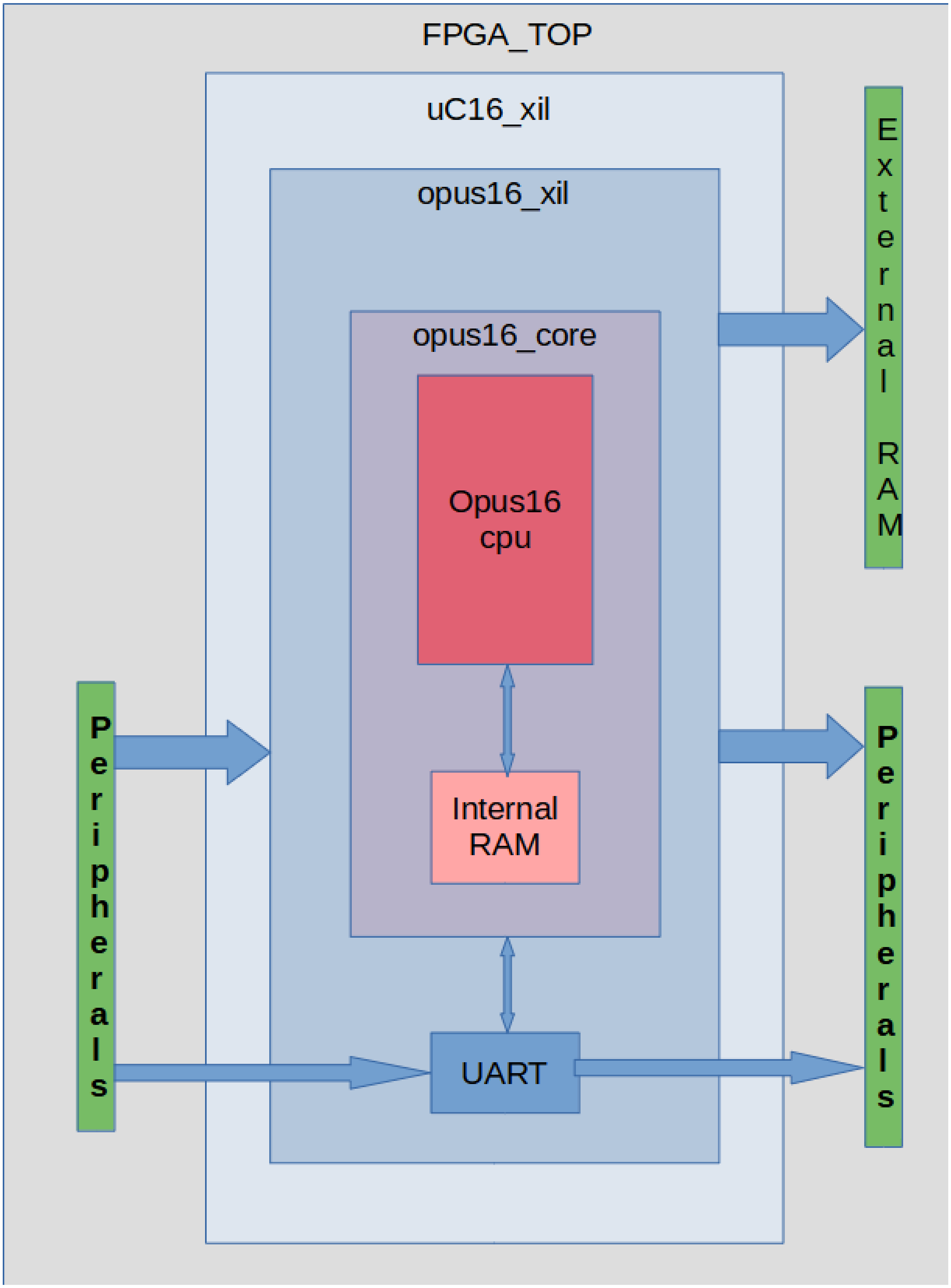
```
affected instructions:
     LDR         STR
     LDRP        STRP
     LDRX        STRX
     LDRPI       STRPI PUSH
     LDRPD STRPD PULL   POP
     LDP         STP
```

2) Opus16 Pinout

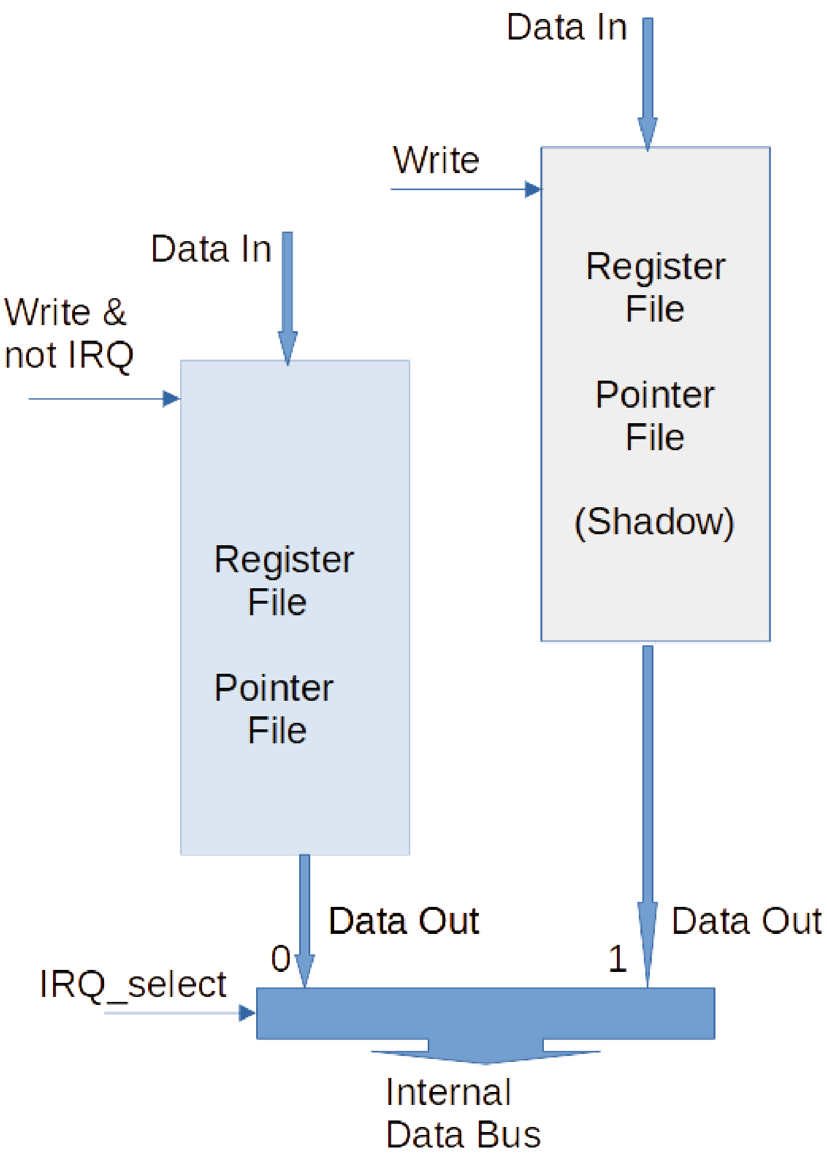


```
        --OUTPUTS
OPUS1_ADDR     : out std_logic_vector(ABUS_WIDTH-1 downto 0); -- ADDRESS BUS
OPUS1_BUSEN_N  : out std_logic; -- BUS ENABLE, ACTIVE LOW
OPUS1_BUSDIR   : out std_logic; -- BUS DIRECTION
OPUS1_READ_N   : out std_logic; -- READ CONTROL LINE, ACTIVE LOW
OPUS1_RSTRB    : out std_logic; -- READ  STROBE
OPUS1_WRITE_N  : out std_logic; -- WRITE CONTROL LINE, ACTIVE LOW
OPUS1_WSTRB    : out std_logic; -- WRITE STROBE
OPUS1_M_IO_N   : out std_logic; -- MEMORY OR IO SELECT
OPUS1_INTENB   : out std_logic; -- INTERRUPT ENABLE
OPUS1_INTA1    : out std_logic; -- INTERRUPT ACKNOWLEDGE
OPUS1_INTA2    : out std_logic; -- INTERRUPT ACKNOWLEDGE
OPUS1_INTA3    : out std_logic; -- INTERRUPT ACKNOWLEDGE
OPUS1_INTA4    : out std_logic; -- INTERRUPT ACKNOWLEDGE
OPUS1_INTB1    : out std_logic; -- INTERRUPT BUSY
OPUS1_INTB2    : out std_logic; -- INTERRUPT BUSY
OPUS1_INTB3    : out std_logic; -- INTERRUPT BUSY
OPUS1_INTB4    : out std_logic; -- INTERRUPT BUSY
OPUS1_DOWN     : out std_logic; -- POWER DOWN
OPUS1_OPORT    : out std_logic_vector(OBUS_WIDTH-1 downto 0); -- USER PORT
OPUS1_PAGE     : out std_logic_vector(ABUS_WIDTH-1 downto 0); -- address page.
OPUS1_PAGEWR   : out std_logic; -- address page write pulse.
OPUS1_SYNC     : out std_logic; -- SYNC
OPUS1_HCODE    : out std_logic; -- PROTECTION OVERWRITE HIT

        --INPUTS
OPUS1_CLK      : in  std_logic; -- SYSTEM CLOCK
OPUS1_RESET    : in  std_logic; -- GLOBAL RESET
OPUS1_INT1     : in  std_logic; -- INTERRUPT #1
OPUS1_INT2     : in  std_logic; -- INTERRUPT #2
OPUS1_INT3     : in  std_logic; -- INTERRUPT #3
OPUS1_INT4     : in  std_logic; -- INTERRUPT #4
OPUS1_READY    : in  std_logic; -- READY, for external slow memories/devices
OPUS1_PCODE    : in  std_logic; -- PROTECTION OVERWRITE
OPUS1_ECODE    : in  std_logic_vector(ABUS_WIDTH-1 downto 0);-- CODE PROTECTION
OPUS1_BCODE    : in  std_logic_vector(ABUS_WIDTH-1 downto 0);-- BOOT CODE

        --INOUT
OPUS1_DOUT     : out std_logic_vector(DBUS_WIDTH-1 downto 0); -- DATA BUS OUT
OPUS1_DINP     : in  std_logic_vector(DBUS_WIDTH-1 downto 0); -- DATA BUS INPUT
OPUS1_DOE_N    : out std_logic  -- DATA BUS OUTPUT ENABLE


Defines:
        ABUS_WIDTH    : integer := 16; -- address bus width
        DBUS_WIDTH    : integer := 16; -- data bus width
        OBUS_WIDTH    : integer := 16; -- ouput port single bit access
        PBUS_WIDTH    : integer := 12 ;-- internal program bus RAM width
```

3)Opus16 Architecture

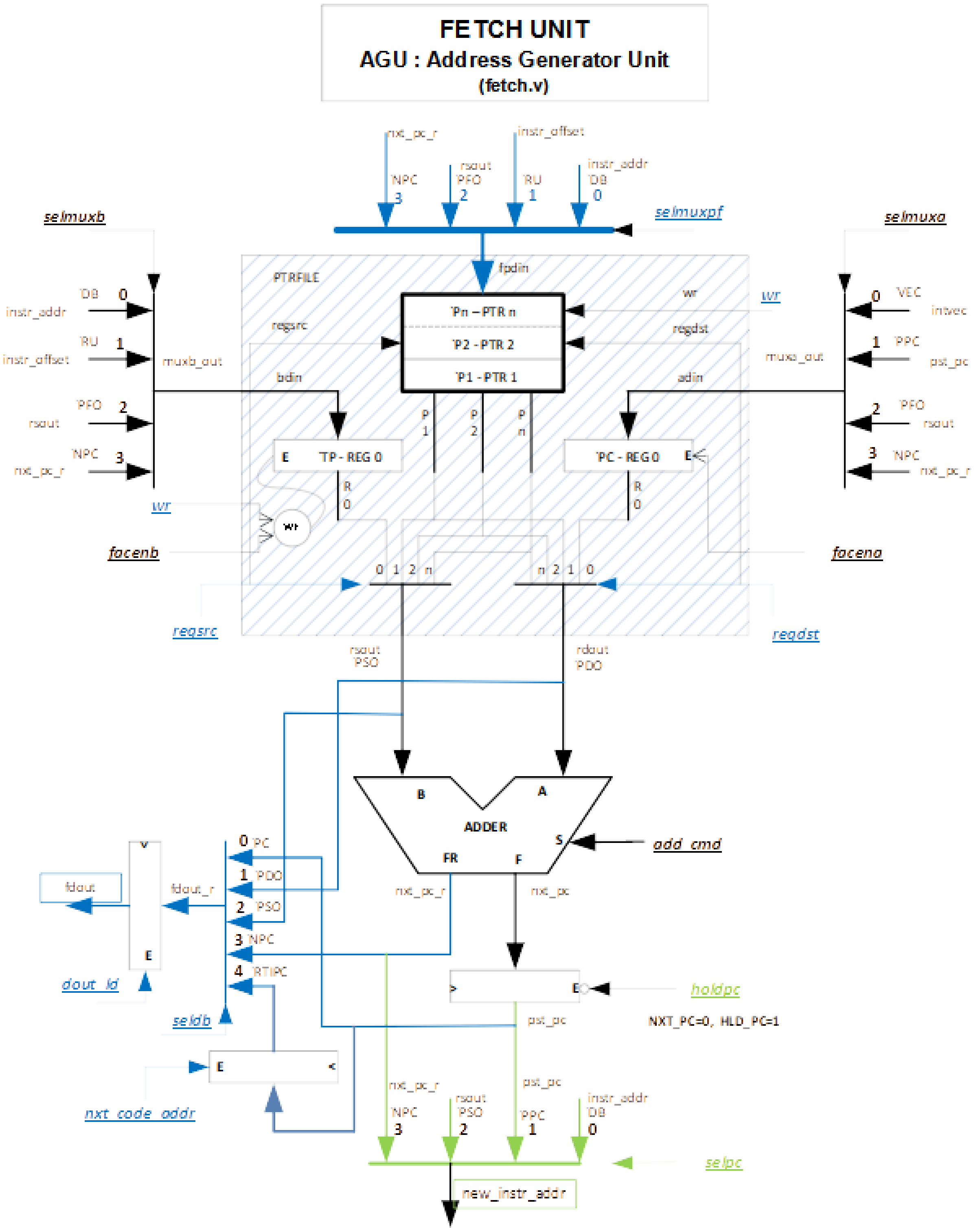


4) Hardware interrupt control



```
     The  interrupt  logic,  in  normal  operation   both  the  register
file  and  the  pointer  file  are  writen  at  the  same  time,  when  one  of
the  4  Opus16  interrupts  occur  the  logic  switch  and  the  shadow  file
is selected.
     This means that in normal mode the File and the Shadow File are
writen  at  the  same  time,  when  an  interrupt  occurs  only  the  shadow
register file is writen, so there is no need to save in stack any of
the 16 registers or pointers.
Also  you  can  pass  arguments  through  registers  to  the  interrupt
routine  but  not  back  unless  you  save  into  memory.  Another  way  is
through the Stack, the normal stack pointer operation is frozen when
an  interrupt  occurred  ,  so  the  last  stack  pointer  is  restored  after
return  from  interrupt,  you  can  use  this  feature  to  pass  values
through the stack.
```

5)Opus16 Internal Structure

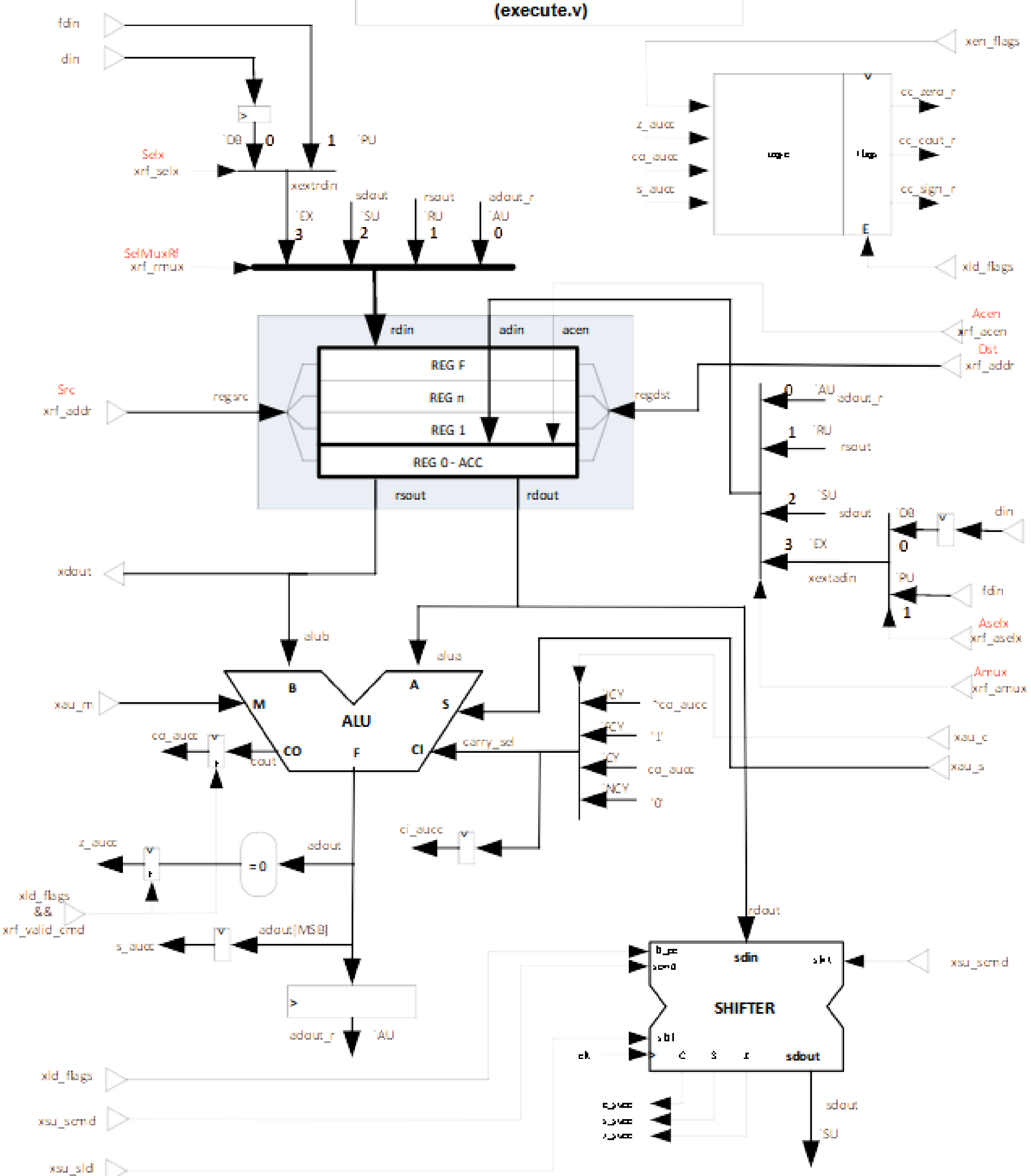


fetch_rf_cmd(regsrc,regdst,wr,selmuxpf,seldb,dout_ld)

fetch_adder_cmd(selmuxa,acena,selmuxb,acenb,add_cmd)

fetch_pc_cmd(selpc,holdpc)

EXEC UNIT
RALU : Register & ALU
(execute.v)



5)Opus16 program example

Assembly Source Code

Whole set of instructions in order, to show the opcodes.

```
define CR 13   // Return (decimal)
define LF ‘h0a // Line feed (hexadecimal)

@'h0000
main:             nop                     ; // no operation

                  // ARITHMETIC
lb_arith:         add         r1,r2       ; // r1 = r1 + r2
                  adc         r3,r4       ; // r3 = r3 + r4 + CY
                  addv        r5,'h1234   ; // r5 = 1234h
                  addrp       r6,p0       ; // r6 = (p0)
                  sub         r7,r8       ; // r7 = r7 - r8
                  sbb         r9,ra       ; // r9 = r9 - ra - CY
                  subv        rB,'h0001   ; // rB = rB - 1
                  subrp       rC,p1       ; // rC = rC - p1

                  // Logic
lb_logic:         not         rD          ; // rD = ~rD
                  not         rD,r1       ; // r1 = ~rD
                  and         rE,rF       ; // rE = rE and rF
                  andv        r1,'hffff   ; // r1 = r1 and 'hffff
                  or          r2,r1       ; // r2 = r2 or r1
                  orv         r3,'h1010   ; // r3 = r3 or 10
                  xor         r4,r5       ; // r4 = r4 xor r5
                  xorv        r6,'h1010   ; // r6 = r6 xor 10
                  inc         r7          ; // r7++
                  dec         r8          ; // r8--
                  cmpr        r9,rA       ; // r9-rA, set status flags
                  cmprv       rb,'h0000   ; // rB-0, set status flags
                  cmprp       rc,p2       ; // rC-p2,set status flags
                  shl         r1          ; // r1<<1
                  shr         r2          ; // r2>>1
                  shl4        r3          ; // r3<<4
                  shr4        r3          ; // r3>>4

                  // MOVE
lb_move:          mvrr        r3,r4       ; // r4 - r3
                  mvrp        r4,p4       ; // p4 = r4
                  mvpr        p4,r7       ; // r7 = p4
                  mvpp        p4,p5       ; // p5 = p4
                  swap        r3,r4       ; // r4=r3, r3=r4
                  swapp       r4,p3       ; // p3=r4, r4=p3

                  // LOAD
lb_load:          ldcv        c0,3        ; // counter #0 = 3
                  ldc         c1,r9       ; // counter #1 = 9
                  ldr         r1,temp0    ; // r1 = (temp0)
                  ldrv        r2,'h0003   ; // r2 = 3
                  ldrp        r3,p1       ; // r3 = (p1)
                  ldrx        r4,p1,2     ; // r4 = (p1+2)
                  ldrpi       r5,p2       ; // r5 = (p2), p2++
                  ldrpd       r6,p3       ; // r6 = (p3),--p3
                  pull        r7,p4       ; // r7 = (p4),--p4
                  pop         r8,p5       ; // r8 = (p5),--p5
                  ldmam       m2          ; // internal
                  ldpagv      'haba       ; // page = 'h0aba
                  ldpag       r9          ; // page = r9

                  // store group
lb_store:         str         r6,temp0    ; // (temp0) = r6
                  strp        r1,p4       ; // (p4) = r1
                  strx        r2,p4,2     ; // (p4+2) = r2
                  strpi       r3,p5       ; // (p5) = r3, p5++
                  push        ra,p8       ; // (p8) = ra, p8++
                  strpd       rB,p9       ; // (p9) = rb, --p9

                  // pointer group
lb_pointer:       ldp         pa,tempa    ; // pa = (tempa)
                  ldpv        pb,temp0    ; // pB = &temp0
                  ldpy        pc,rb       ; // pc = (pc+rb)
                  stp         pd,tempa    ; // (tempa) = pd
                  incp        pE          ; // pE++
                  decp        pf          ; // pF--

                  // I/O group
lb_io:            inp         r1,port1    ; // r1 = (port1)
                  inpp        r2,pc       ; // r2 = (pC)
                  outp        r3,port2    ; // (port2) = r3
                  outpp       r4,pd       ; // (pD) = r4

                  // control transfer
main1:            bra         main1       ; // branch to main

                  // Jump Group, full address range

main2:            dcjz        c0,main2    ; // if c0=0 jump main2
main3:            dcjnz       c1,main3    ; // if c1<>0 jump main3
                  jmp         main4       ; // always jump main4
main4:            jmp         f1,main5    ; // if uflag 1 = 0 jump main5
main5:            jmp         tA,main6    ; // if uflag a = 1 jump main6
main6:            jmpp        pe          ; // jump to (pe)
main7:            jsr         subr1       ; // jump subr1
subr1:            rts                     ; // return from subr1
                  rti                     ; // return from interrupt

                  // User flags and single bit output port.
lb_bits:          uflag       tE          ; // user flag E = 1
                  uflag       ff          ; // user flag F = 0
                  uport       ta          ; // oport a (10dec) = 1
                  uport       fB          ; // oport b (11dec) = 0

                  // Bit test
                  bit         r1,r2       ; // r1 and r2, cc affected
                  bitv        r3,'h0011   ; // r3 and  3, cc affected

                  // Interrupts
lb_irq:           cli                     ; // disable interrupts
                  sei                     ; // enable interruts
swia:             swi         swia        ; // software interrupt / trap

                  // Special intructions
                  hwi                     ; // do not use it
                  stop                    ; // output stop signal

                  // temporary data storage
temp0:            dw    33          ; // one word and initialize with 33 decimal
tempa:            ds    5           ; // reserve 5 words, uninitialized
temp_ascii:       dw    CR          ; // Carry Return ASCII 0d(hex)
port1:            dw    'h0100      ; // address port1
port2:            dw    'h0108      ; // address port2
```

Assembly Code List

```
*****************************************
Assembler and Date    : version 6.1: (11-09-21)
Program size          : 181
----------
Asm  file name        : sim_isa_order.asm
List file name        : sim_isa_order.lst
----------
       Hex  file name : sim_isa_order.hex
       COE  file name : sim_isa_order.coe
       MEM  file name : sim_isa_order.mem
Altera Mif  file name : sim_isa_order_alt.mif
Xilinx Mif  file name : sim_isa_order_xil.mif
*****************************************

Labels and Defines:
       lb_arith: 0041
        lb_bits: 00a0
          lb_io: 0089
         lb_irq: 00a7
        lb_load: 0066
       lb_logic: 004b
        lb_move: 0060
     lb_pointer: 0080
       lb_store: 0078
           main: 0000
          main1: 008f
          main2: 0091
          main3: 0093
          main4: 0097
          main5: 0099
          main6: 009b
          main7: 009c
          port1: 00b3
          port2: 00b4
          subr1: 009e
           swia: 00a9
          temp0: 00ad
          tempa: 00ae
*****************************************
Opcode listing.

0000   0000   main:nop ;
0001   0021   ldrv r0,'h1000 ;
0002   1000
0003   0121   ldrv r1,'h1001 ;
0004   1001
0005   0221   ldrv r2,'h1002 ;
0006   1002
0007   0321   ldrv r3,'h1003 ;
0008   1003
0009   0421   ldrv r4,'h1004 ;
000a   1004
000b   0521   ldrv r5,'h1005 ;
000c   1005
000d   0621   ldrv r6,'h1006 ;
000e   1006
000f   0721   ldrv r7,'h1007 ;
0010   1007
0011   0821   ldrv r8,'h1008 ;
0012   1008
0013   0921   ldrv r9,'h1009 ;
```

```
0014   1009
0015   0a21   ldrv ra,'h100a ;
0016   100a
0017   0b21   ldrv rb,'h100b ;
0018   100b
0019   0c21   ldrv rc,'h100c ;
001a   100c
001b   0d21   ldrv rd,'h100d ;
001c   100d
001d   0e21   ldrv re,'h100e ;
001e   100e
001f   0f21   ldrv rf,'h100f ;
0020   100f
0021   002e   ldpv r0,'h2000 ;
0022   2000
0023   012e   ldpv r1,'h2001 ;
0024   2001
0025   022e   ldpv r2,'h2002 ;
0026   2002
0027   032e   ldpv r3,'h2003 ;
0028   2003
0029   042e   ldpv r4,'h2004 ;
002a   2004
002b   052e   ldpv r5,'h2005 ;
002c   2005
002d   062e   ldpv r6,'h2006 ;
002e   2006
002f   072e   ldpv r7,'h2007 ;
0030   2007
0031   082e   ldpv r8,'h2008 ;
0032   2008
0033   092e   ldpv r9,'h2009 ;
0034   2009
0035   0a2e   ldpv ra,'h200a ;
0036   200a
0037   0b2e   ldpv rb,'h200b ;
0038   200b
0039   0c2e   ldpv rc,'h200c ;
003a   200c
003b   0d2e   ldpv rd,'h200d ;
003c   200d
003d   0e2e   ldpv re,'h200e ;
003e   200e
003f   0f2e   ldpv rf,'h200f ;
0040   200f
0041   2101   lb_arith: add r1,r2 ;
0042   4302   adc r3,r4 ;
0043   0503   addv r5,'h1234 ;
0044   1234
0045   0604   addrp r6,p0 ;
0046   8705   sub r7,r8 ;
0047   a906   sbb r9,ra ;
0048   0b07   subv rB,'h0001 ;
0049   0001
004a   1c08   subrp rC,p1 ;
004b   dd09   lb_logic: not rD ;
004c   d109   not rD,r1 ;
004d   fe0a   and rE,rF ;
004e   010b   andv r1,'hffff ;
004f   ffff
0050   120c   or r2,r1 ;
0051   030d   orv r3,'h1010 ;
0052   1010
0053   540e   xor r4,r5 ;
0054   060f   xorv r6,'h1010 ;
0055   1010
```

```
0056   0710   inc r7 ;
0057   0811   dec r8 ;
0058   a912   cmpr r9,rA ;
0059   bb13   cmprv rb,'h0000 ;
005a   0000
005b   2c14   cmprp rc,p2 ;
005c   0115   shl r1 ;
005d   0216   shr r2 ;
005e   0317   shl4 r3 ;
005f   0318   shr4 r3 ;
0060   3419   lb_move: mvrr r3,r4 ;
0061   441a   mvrp r4,p4 ;
0062   471b   mvpr p4,r7 ;
0063   451c   mvpp p4,p5 ;
0064   341d   swap r3,r4 ;
0065   431e   swapp r4,p3 ;
0066   001f   lb_load: ldcv c0,3 ;
0067   0003
0068   9143   ldc c1,r9 ;
0069   0120   ldr r1,temp0 ;
006a   00ad
006b   0221   ldrv r2,'h0003 ;
006c   0003
006d   1322   ldrp r3,p1 ;
006e   1423   ldrx r4,p1,2 ;
006f   0002
0070   2524   ldrpi r5,p2 ;
0071   3625   ldrpd r6,p3 ;
0072   4725   pull r7,p4 ;
0073   5825   pop r8,p5 ;
0074   2026   ldmam m2 ;
0075   0027   ldpagv 'haba ;
0076   0aba
0077   9042   ldpag r9 ;
0078   6028   lb_store: str r6,temp0 ;
0079   00ad
007a   1429   strp r1,p4 ;
007b   242a   strx r2,p4,2 ;
007c   0002
007d   352b   strpi r3,p5 ;
007e   a82b   push ra,p8 ;
007f   b92c   strpd rB,p9 ;
0080   0a2d   lb_pointer: ldp pa,tempa ;
0081   00ae
0082   0b2e   ldpv pb,temp0 ;
0083   00ad
0084   bc44   ldpy pc,rb ;
0085   d02f   stp pd,tempa ;
0086   00ae
0087   0e30   incp pE ;
0088   0f31   decp pf ;
0089   0132   lb_io: inp r1,port1 ;
008a   00b3
008b   c233   inpp r2,pc ;
008c   3034   outp r3,port2 ;
008d   00b4
008e   4d35   outpp r4,pd ;
008f   0037   main1: bra main1 ;
0090   ffff
0091   0036   main2: dcjz c0,main2 ;
0092   0091
0093   0138   main3: dcjnz c1,main3 ;
0094   0093
0095   0039   jmp main4 ;
0096   0097
0097   0139   main4: jmp f1,main5 ;
```

```
0098   0099
0099   8a39   main5: jmp tA,main6 ;
009a   009b
009b   0e3a   main6: jmpp pe ;
009c   003b   main7: jsr subr1 ;
009d   009e
009e   003c   subr1: rts ;
009f   003d   rti ;
00a0   8e3e   lb_bits: uflag tE ;
00a1   0f3e   uflag ff ;
00a2   8a3f   uport ta ;
00a3   0b3f   uport fB ;
00a4   2140   bit r1,r2 ;
00a5   0341   bitv r3,'h0011 ;
00a6   0011
00a7   0046   lb_irq: cli ;
00a8   0045   sei ;
00a9   0047   swia: swi swia ;
00aa   00a9
00ab   005a   hwi ;
00ac   00ff   stop ;
00ad   0021   temp0: dw 33 ;
00b3   0100   port1: dw 'h0100 ;
00b4   0108   port2: dw 'h0108 ;
```