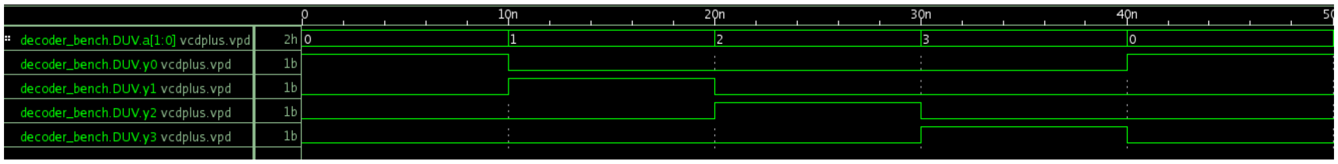
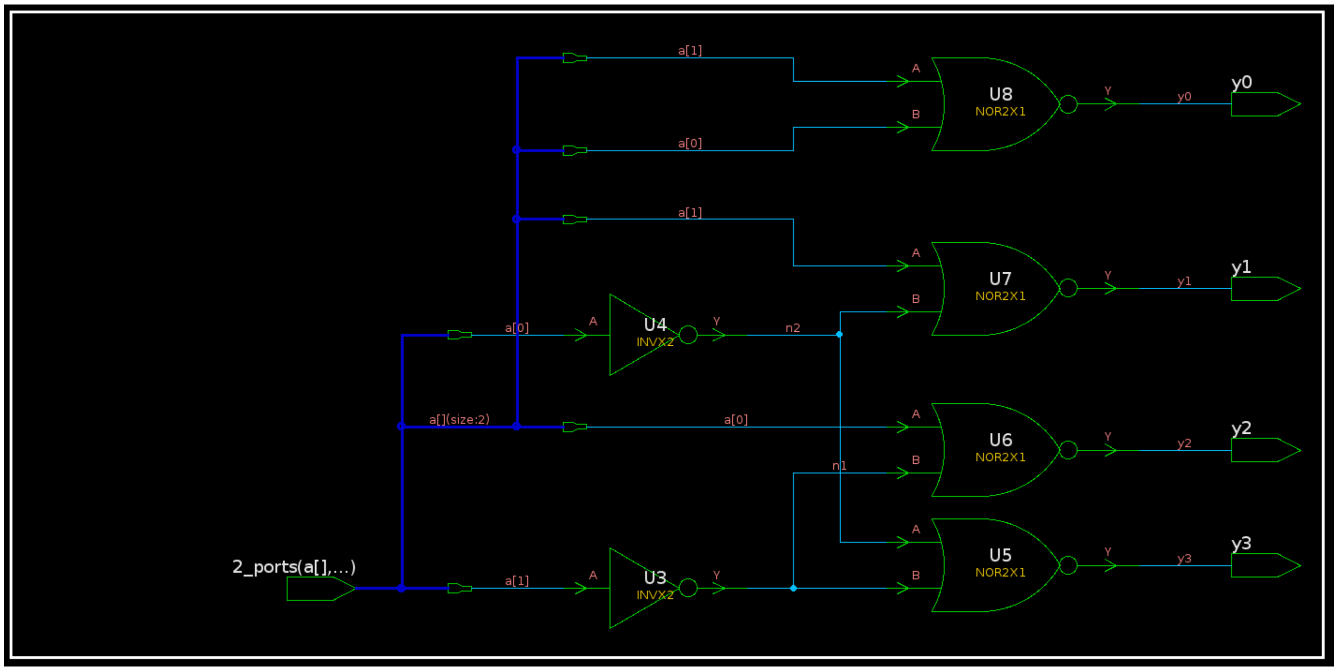
Adam Ness and Marty Townley
2/27/17
ECE424

Lab 5 Report

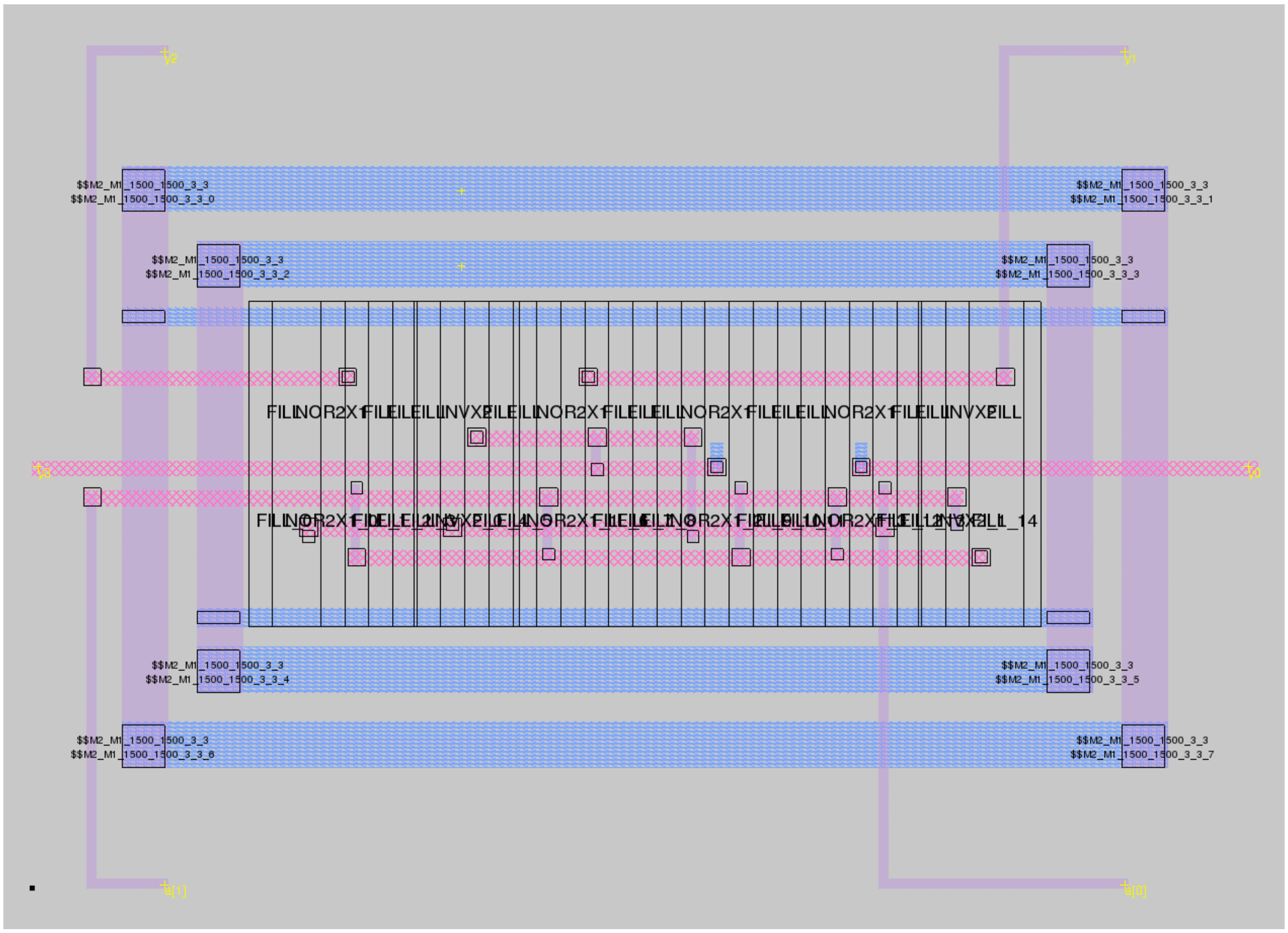
decoder_bench.v Simulation Waveform:



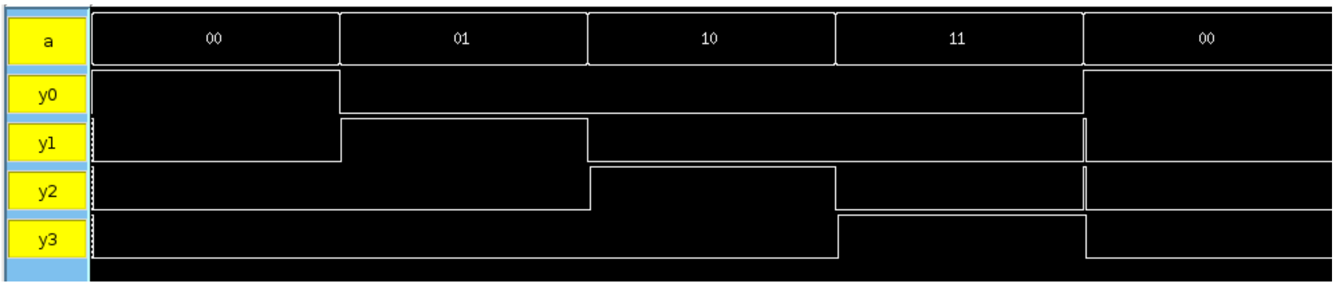
decoder Schematic:



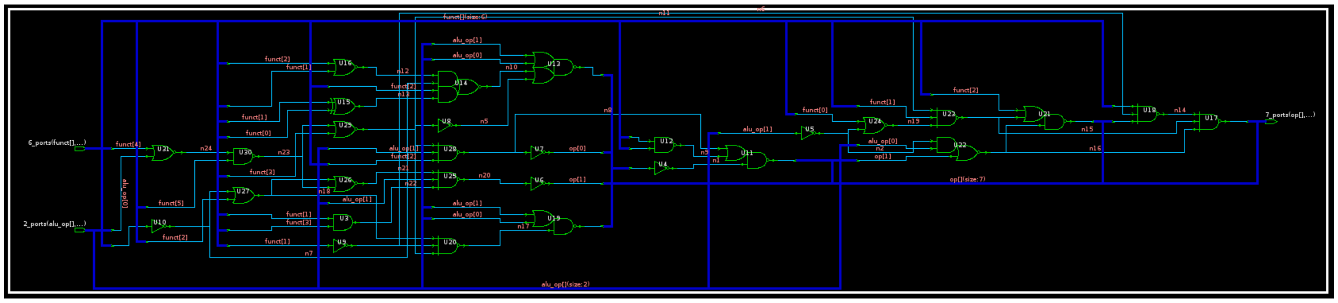
decoder Magic Layout:


y2
y1
y3
y0
a[1]
a[0]
FILL NOR2X1 FILL INVX2 FILL NOR2X1 FILL NOR2X1 FILL NOR2X1 FILL INVX2 FILL
FILL_14
$$M2_M1_1500_1500_3_3

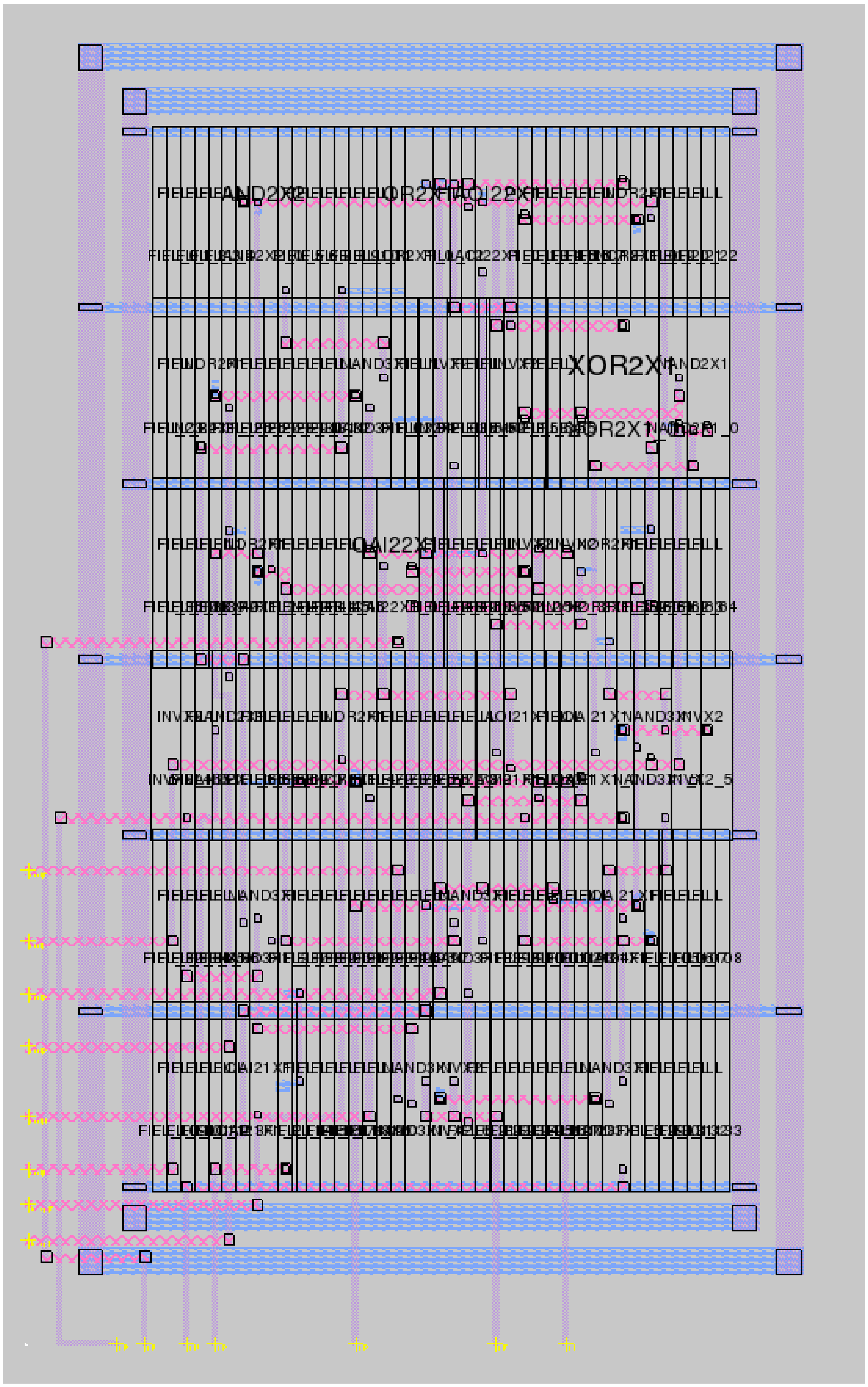
decoder IRSIM Waveform:



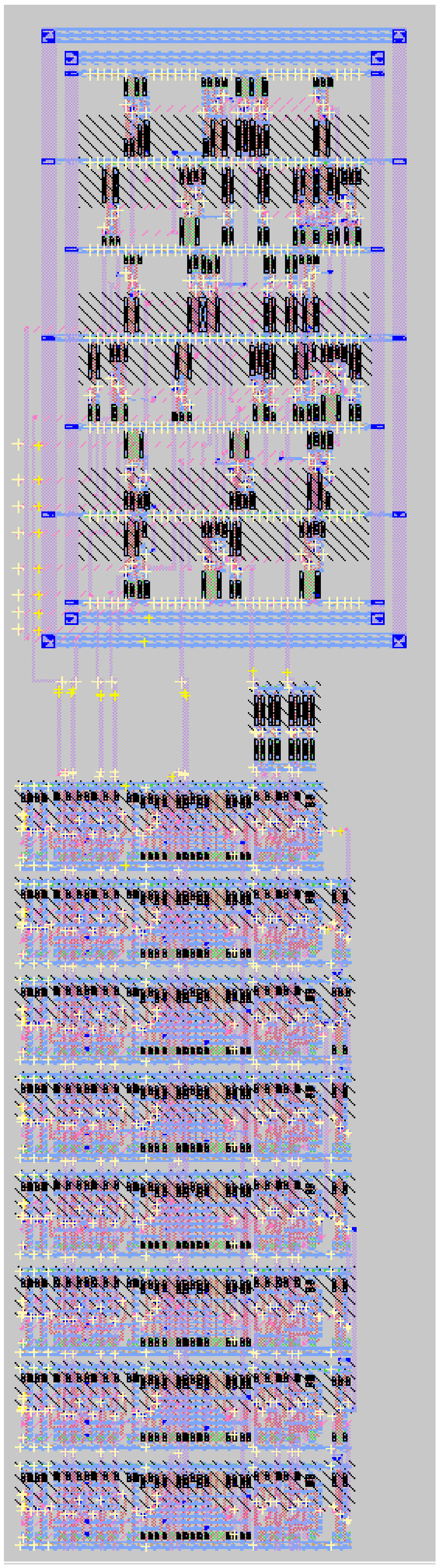
alu_ctl Schematic:



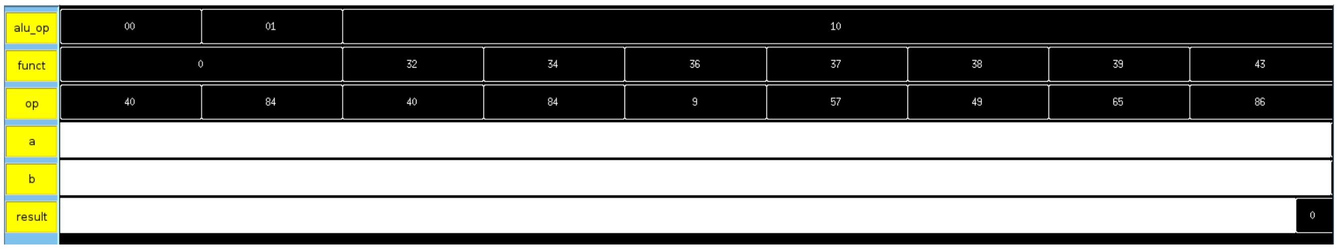
alu_ctl Magic Layout:



alu_decoder Magic Layout (alu_ctl + alt_alu):



alt_decoder IRSIM Waveform:



Notes:

In the tarball:

Magic Layout that has <u>only</u> alu_ctl: alu_ctl.mag
Magic Layout with alu_ctl <u>and</u> alt_alu_8: alu_decode.mag

TCL command file for decoder: decoder_test.cmd
TCL command file for alu_decode: alu_decode.cmd

All tests were passed for both simulations.

<u>Appendix:</u>

decoder_test.cmd:

```
h Vdd!
l Gnd!

##### VECTOR DECLARATIONS #####
vector a {a[1]} {a[0]}

stepsize 100

analyzer a y0 y1 y2 y3


##### EXECUTED CODE #####

setvector a 00
s

setvector a 01
s

setvector a 10
s

setvector a 11
s
```

```
setvector a 00
s
```

alu_decode.cmd:

```
h Vdd!
l Gnd!

############# VECTOR DECLARATIONS #####################

vector alu_op alu_op1 alu_op0
vector funct  funct5 funct4 funct3 funct2 funct1 funct0
vector op     op6 op5 op4 op3 op2 op1 op0

vector a a7 a6 a5 a4 a3 a2 a1 a0
vector b b7 b6 b5 b4 b3 b2 b1 b0
vector result result7 result6 result5 result4 result3 result2 result1 result0

stepsize 250

analyzer alu_op funct op a b result zero
w alu_op funct op a b result zero

################## PROCESSES ############################

proc dec2bin {i {width {}}} {
    #returns the binary representation of $i
    # width determines the length of the returned string (left truncated or added
left 0)
    # use of width allows concatenation of bits sub-fields

    set res {}
    if {$i<0} {
        set sign -
        set i [expr {abs($i)}]
    } else {
        set sign {}
    }
    while {$i>0} {
        set res [expr {$i%2}]$res
        set i [expr {$i/2}]
    }
    if {$res eq {}} {set res 0}

    if {$width ne {}} {
        append d [string repeat 0 $width] $res
        set res [string range $d [string length $res] end]
    }
    return $sign$res
}

proc cycleInput {} {
for {set i 0} {$i < 256} {incr i} {
    set adec [dec2bin $i 8]
    #puts "adec is: $adec"
    setvector a $adec

    set j 1
            while {$j < 256}  {
                    set bdec [dec2bin $j 8]
              #puts "bdec is: $bdec"
              setvector b $bdec
              set j [expr $j << 1]
              s
              checkAnswer
            }
}
}

proc checkAnswer {} {
     set expectedNum 0

     if { [query op] == 9 } {
      #and
      set expectedNum [expr [query a] & [query b] ]
     } elseif { [query op] ==  57} {
       #or
       set expectedNum [expr [query a] | [query b] ]
     } elseif { [query op] ==  65} {
       #nor
       set expectedNum [expr [query a] | [query b] ]
       set expectedNum [expr ~$expectedNum]
       set expectedNum [expr 256 + $expectedNum]
     } elseif { [query op] ==  49} {
       #xor
       set expectedNum [expr [query a] ^ [query b] ]
     } elseif { [query op] ==  40} {
       #add
       set expectedNum [expr [query a] + [query b] ]
     } elseif { [query op] ==  84} {
       #sub
       set expectedNum [expr [query a] - [query b] ]
       if { $expectedNum < 0} {
              set expectedNum [expr $expectedNum + 256]
       }
     } elseif { [query op] ==  86} {
       #slt
       set expectedNum [expr [query a] < [query b] ]
     }

     set decex $expectedNum
     set binex [dec2bin $expectedNum 8]
     set decre [query result]
     set binre [dec2bin $decre 8]

     puts "a is [query a]; b is [query b]"
     puts "dec expected: $decex"
     #puts "bin expected: $binex"
     puts "dec result: $decre"
     #puts "bin result: $binre"

     #if { $expectedNum < 240} {
      assert result [dec2bin $expectedNum 8]
     #}

}

###################### EXECUTED CODE: #######################################

#lw, sw, ...
setvector alu_op 0x0
setvector funct  0x00
cycleInput

#beq, bne, ...
setvector alu_op 0x1
setvector funct  0x00
cycleInput

#add
setvector alu_op 0x2
setvector funct  0x20
cycleInput

#sub
setvector alu_op 0x2
setvector funct  0x22
cycleInput

#and
setvector alu_op 0x2
setvector funct  0x24
cycleInput

#or
setvector alu_op 0x2
setvector funct  0x25
cycleInput

#xor
setvector alu_op 0x2
setvector funct  0x26
cycleInput

#nor
setvector alu_op 0x2
setvector funct  0x27
cycleInput

#sltu
setvector alu_op 0x2
setvector funct  0x2B
cycleInput
```