ADD( R15, R15, R8 )

BEQ( R8, ERROR\_CHECK\_1, R15 )   || R8 used to check for error in test cases

JMP( R9 ) || base\_3rror || Illegal Operation, system crash

ADDC( R15, 1000, R9) || R9 used to get to detect error

**|| Error #1:**

|| PCSEL Signal is always 0. Input to the PC register is always PC+4 regardless of the

|| instruction

**ERROR\_CHECK\_1:**

ADD( R15, R15, R1 )       || Set Reg[R1] ← 0

ADD( R15, R15, R0 )       || Set Reg[R0] ← 0

BR( **ERROR\_CHECK\_1\_SUCCESS** ) || Branch if above error is does not exist

JMP( R9 ) || 3rr0r\_CHECK\_1       || Illegal Operation, system crash

**ERROR\_CHECK\_1\_SUCCESS:**

|| **Error #2:**

|| Input 1 of PCSEL mux has a value of 0 instead of PC+4+4\*SEXT(C)

|| Z input to control logic is always 0 instead of the correct value depending on RD1

**ERROR\_CHECK\_2:**

ADD( R15, R15, R0 )     || Set Reg[R0] ← 0

|| BEQ / BR would not execute if above error exists

BEQ( R0, **ERROR\_CHECK\_2\_SUCCESS**, R15 )

JMP( R9 ) || 3rr0r\_CHECK\_2     || Illegal Operation, System Crash

**ERROR\_CHECK\_2\_SUCCESS:**

|| Error #3:

|| BSEL multiplexer control signal 0 instead of as per intended current instruction

|| OPCODE

**ERROR\_CHECK\_3:**

ADD( R15, R15, R0 ) || Set Reg[R0] ← 0

ADDC( R0, 1, R0 ) || Set Reg[R0] ← 1

BNE(R0, **ERROR\_CHECK\_3\_SUCCESS**, R1) || If no error exist then branch

JMP( R9 ) || 3rror\_CHECK\_3              || Illegal Operation, System Crash

**ERROR\_CHECK\_3\_SUCCESS:**

|| Error #4

|| Input 0 of WDSEL mux has a value of 0 instead of PC+4

**ERROR\_CHECK\_4:**

ADD( R15, R15, R0 )   || Set Reg[R0] ← 0

BEQ(R0, **ERROR\_CHECK\_4\_BRANCH**, R1) || Branch and store PC+4 into R1

BR( **ERROR\_CHECK\_4\_SUCCESS** )

**ERROR\_CHECK\_4\_BRANCH:**

ADDC( R15, 4, R8 ) || Set R8 ← used to handle error at instruction addr b0

JMP( R1 ) || Jump to .=0x0 if error else jmp to BR( <SUCCESS> )

**ERROR\_CHECK\_4\_SUCCESS:**

ADD( R15, R15, R8 ) || Clear R8 as we have passed this test

ADD( R15, R15, R0 ) || Clear R0

ADD( R15, R15, R9 ) || Clear R9

ADD( R15, R15, R1 ) || Clear R1

|| ##############################################################################

|| -- PRELOAD CHECK DONE BEYOND THIS POINT (NO DETECTED ERROR IN DATAPATH) --

|| First Instruction of Game

|| Upon calculating valid positions on the board, go to player selection turn

|| If there is a valid position then, find 1st possible spot from top to bottom

**SETUP:**

**||** Set player beginning grid

|| Set for Player 1 (Beginning Pieces):

ADDC(R15, 4104, R11)

SHLC(R11, 24, R11)

|| Set for Player 2 (Beginning Pieces):

ADDC(R15, 2064, R10)

SHLC(R10, 24, R10)

|| GAME STARTS - Setup for 1st Player

ADDC(R15, 1, R4) || Setup Register R4 to indicate Player 1’s turn

|| Branch to function to generate valid grid for this player

BR( **GENERATE\_VALID\_GRID,** R14 )

|| Reset Timer

ADDC(R15, 153, R8) || set both player 1 and player 2 timer to 9 seconds

|| Upon calculating valid positions on the board, go to player selection turn

|| If there is a valid position then, find 1st possible spot from top to bottom

**PLAYER\_TURN:**

**|| Assumption: The Valid Move has already been calculated and stored in R9**

**|| Check if Valid position is 0, if it is 0 move to next turn (no Valid position)**

**|| Make sure there is at least 1 bit that is valid before performing the rest of**

**|| these sets of instruction**

|| Compare with valid move grid, if no valid moves available move to next turn

BEQ(R9, **NEXT\_TURN\_NO\_VALID\_MOVE**, R15)

|| Puts selected bit to most signif Valid bit (in grid)

BR( **FIND\_FIRST\_VALID\_BIT\_GRID,** R14 **)**

|| Light up current selected bit, different colour using Register R6

|| Hardwired on FPGA

|| Wait for player input (Loop for CPU to poll player inputs)

**POLL\_INPUT:**

ADD(R15, R15, R3) || Timer to 0

**POLL\_INPUT\_STAGE:**

ADD(R15, R15, R5) || Reset R5 to all “0”s

**INPUT\_LOOP:**

|| R5 will be wired to buttons

|| 5 possible buttons, taking the 5 LSB of R5

|| Left - 001

|| Right - 010

|| Up - 011

|| Down - 100

|| Confirm - 101

**|| Handle input, poll for player input**

|| ROM with selector bits that corresponds to above button bit sequence

|| ALUFN code to take in user input, use WDSEL = 0b3, to get the user input

INP(R5) || Defined function that stores player input to R5

ANDC(R8, 15, R1) || Get Player 1 timer number

ANDC(R8, 240, R2) || Get Player 2 timer number

|| Timer decrements using hardware.

SUBC(R4, 1, R0) || To check timer during player selection

ADDC(R3, 1, R3) || Increment counter by 1

|| R1: Player 1 Timer, R2: Player 2 Timer

BEQ(R0, **PLAYER\_1\_CHECK\_TIMER**, R15)

|| If Player 2:

**PLAYER\_2\_CHECK\_TIMER:**

|| Create binary for 2Million, b11110100001 0010000000

ADDC(R15, 1953, R0)

SHLC(R0, 10, R0)

ADDC(R0, 128, R0)

CMPLE(R0, R3, R0) || move to next tick after 1 sec

BNE(R0, **PLAYER\_2\_CHECK\_TIMER\_NEXT\_TICK,** R15)

**BEQ(R5, INPUT\_LOOP, R15)** || Keep looping until there is an input

BR( **HANDLE\_INPUT** )

**PLAYER\_2\_CHECK\_TIMER\_NEXT\_TICK:**

BEQ(R2, **PLAYER\_2\_NEXT\_TURN\_TIMER,** R15)

SHRC(R2, 4, R2) || Shift the [7:4] to [3:0]

SUBC(R2, 1, R2) || Subtract 1 from this value

SHLC(R2, 4, R2) || Shift back to [7:4]

BEQ(R2, **PLAYER\_2\_NEXT\_TURN\_TIMER,** R15)

ADD(R15, R15, R3) || Reset counter

ADD(R15, R1, R0) || Mask out only Player 1 timer

ADD(R2, R0, R8) || Add Player 2 timer into R8

**BEQ(R5, INPUT\_LOOP, R15)** || Keep looping until there is an input

BR( **HANDLE\_INPUT** )

**PLAYER\_2\_NEXT\_TURN\_TIMER:**  || If out-of-time, move to next turn

|| If player 2 runs out of time, set player 1’s timer back to 9 seconds, player 2’s timer set to

|| 5 seconds

ADDC(R15, 89, R8) || b01011001, set Player 1 and Player 2 time

||BR( **FLIP\_UPDATE**, R14 )

BR(**NEXT\_TURN**) || Go to next turn

|| To handle player 1 timer check

**PLAYER\_1\_CHECK\_TIMER:**

|| Create binary for 2Million, b11110100001 0010000000

ADDC(R15, 1953, R0)

SHLC(R0, 10, R0)

ADDC(R0, 128, R0)

CMPLE(R0, R3, R0) || move to next tick after 1 sec

BNE(R0, **PLAYER\_1\_CHECK\_TIMER\_NEXT\_TICK,** R15)

**BEQ(R5, INPUT\_LOOP, R15)** || Keep looping until there is an input

BR( **HANDLE\_INPUT** )

**PLAYER\_1\_CHECK\_TIMER\_NEXT\_TICK:**

BEQ(R1, **PLAYER\_1\_NEXT\_TURN\_TIMER,** R15)

SUBC(R1, 1, R1)

BEQ(R1, **PLAYER\_1\_NEXT\_TURN\_TIMER,** R15)

ADD(R15, R15, R3) || Reset counter

ADD(R15, R2, R0) || Mask out only Player 2 timer

ADD(R1, R0, R8) || Add Player 1 timer into R8

**BEQ(R5, INPUT\_LOOP, R15)** || Keep looping until there is an input

BR( **HANDLE\_INPUT** )

**PLAYER\_1\_NEXT\_TURN\_TIMER:**

|| If player 1 runs out of time, set player 2’s timer back to 9 seconds, player 1’s timer set to

|| 5 seconds

ADDC(R15, 149, R8) || b10010101, set Player 1 and Player 2 time

|| BR( **FLIP\_UPDATE**, R14 )

BR(**NEXT\_TURN**) || Go to next turn

**HANDLE\_INPUT:**

|| There was an **input in R5**, store this value in **R0 → Stores Player input value**

ADD(R5, R15, R0) || x8017c000,

|| Do Subtraction to find value

|| Check if Left Button SMASH!!

SUBC(R0, 1, R2) || BYE R2 I dont need you anymore

BEQ(R2, **HANDLE\_LEFT**, R15) || If it is left SMASH, handle left move

|| Check if Right Button SMASH!!

SUBC(R0, 2, R2)

BEQ(R2, **HANDLE\_RIGHT**, R15) || If it is right SMASH, handle right move

|| Check if UP Button SMASH!!

SUBC(R0, 3, R2) || xc4800003

BEQ(R2, **HANDLE\_UP**, R15) || If it is up SMASH, handle up move

|| Check if DOWN Button SMASH!!

SUBC(R0, 4, R2)

BEQ(R2, **HANDLE\_DOWN**, R15) || If it is down SMASH, handle down move

|| Check if CONFIRM Button SMASH!!

SUBC(R0, 5, R2)

BEQ(R2, **HANDLE\_CONFIRM**, R15) || If it is confirm SMASH, handle confirm move

||**##############**

**HANDLE\_LEFT:**

SHLC(R6, 1, R6) || Shift the current selected bit to the left by 1

AND(R6, R7, R0) || Check if current selected bit is still within current row

BNE(R0, **HANDLE\_LEFT\_WITHIN\_ROW**, R15) || If still within row

BR(**FIND\_FIRST\_VALID\_BIT, R14**) || not within row, shift back to original position

BR(**HANDLE\_LEFT\_VALID\_BIT**) || Found a valid bit

**HANDLE\_LEFT\_WITHIN\_ROW:** || Still within the current row

AND(R6, R9, R0) || Check if current selected bit is a valid bit

BEQ(R0, **HANDLE\_LEFT**, R15) || If it is a valid bit, if 0 means not valid, do SHL

BR(**HANDLE\_LEFT\_VALID\_BIT**)

**HANDLE\_LEFT\_VALID\_BIT:** || If left valid

|| TODO: Instruction if we found a valid bit on left.

BR( **POLL\_INPUT\_STAGE** ) || Go back to get more input

||**##############**

**HANDLE\_RIGHT:**

SHRC(R6, 1, R6) || Shift current selected bit to the right by 1

AND(R6, R7, R0) || Check if current selected bit is still within current row

BNE(R0, **HANDLE\_RIGHT\_WITHIN\_ROW**, R15) || If still within row

|| not within row, shift back to original position

BR(**HANDLE\_RIGHT\_FIND\_LAST\_VALID\_BIT**)

**HANDLE\_RIGHT\_FIND\_LAST\_VALID\_BIT:**

BEQ(R0, **FIND\_LAST\_VALID\_BIT**, R14) || If it isnt the last bit, find the last bit

BR(**HANDLE\_RIGHT\_VALID\_BIT**) || Once the last bit has been found, JMP

**HANDLE\_RIGHT\_WITHIN\_ROW:**

AND(R6, R9, R0) || Check if current selected bit is a valid bit

BEQ(R0, **HANDLE\_RIGHT**, R15) || If it is a valid bit, if 0 means not valid, do SHR

BR(**HANDLE\_RIGHT\_VALID\_BIT**)

**HANDLE\_RIGHT\_VALID\_BIT:** || If Right valid

|| TODO: Instruction if we found a valid bit on Right.

BR( **POLL\_INPUT\_STAGE** ) || Go back to get more input

||**##############**

**HANDLE\_UP:**

SHLC(R7, 8, R0) || Store copy of R7 >> 8 into R0 to check if out of bounds if move down

BEQ(R0, **HANDLE\_UP\_OUT\_OF\_ROW**, R15) || If out of bounds, immediately branch

|| If in the bounds of the columns (Down not out of bounds)

AND(R0, R9, R1) || AND Mask to check if there is valid bits on this row

BNE(R1, **HANDLE\_UP\_WITHIN\_ROW,** R15) || Branch if this row has valid bits

BR(**HANDLE\_UP\_LOOP**) || Loop and shift to next row if no valid bits

**HANDLE\_UP\_LOOP:**

SHLC(R0, 8, R0)

BEQ(R0, **HANDLE\_UP\_OUT\_OF\_ROW**, R15) || If out of bounds, immediately branch

|| If in the bounds of the columns (Down not out of bounds)

AND(R0, R9, R1) || AND Mask to check if there is valid bits on this row

BNE(R1, **HANDLE\_UP\_WITHIN\_ROW,** R15) || Branch if this row has valid bits

BR(**HANDLE\_UP\_LOOP**) || Loop and shift to next row if no valid bits

**HANDLE\_UP\_OUT\_OF\_ROW:** || If up movement lead to out-of-bounds

BR( **POLL\_INPUT\_STAGE** ) || GO back to poll input

**HANDLE\_UP\_WITHIN\_ROW:** || If up movement is valid

ADD(R15, R0, R7) || Store temp row to actual selected row

BR(**FIND\_FIRST\_VALID\_BIT**, R14) || Find first valid bit to display, x77bc004c

|| R7 is positioned correctly at the last row

BR( **POLL\_INPUT\_STAGE** ) || Go back to get more input

||**##############**

**HANDLE\_DOWN:**

SHRC(R7, 8, R0) || Store copy of R7 >> 8 into R0 to check if out of bounds if move down

BEQ(R0, **HANDLE\_DOWN\_OUT\_OF\_ROW**, R15) || If out of bounds, immediately branch

|| If in the bounds of the columns (Down not out of bounds)

AND(R0, R9, R1) || AND Mask to check if there is valid bits on this row

BNE(R1, **HANDLE\_DOWN\_WITHIN\_ROW,** R15) || Branch if this row has valid bits

BR(**HANDLE\_DOWN\_LOOP**) || Loop and shift to next row if no valid bits

**HANDLE\_DOWN\_LOOP:**

SHRC(R0, 8, R0)

BEQ(R0, **HANDLE\_DOWN\_OUT\_OF\_ROW**, R15) || If out of bounds, immediately branch

|| If in the bounds of the columns (Down not out of bounds)

AND(R0, R9, R1) || AND Mask to check if there is valid bits on this row

BNE(R1, **HANDLE\_DOWN\_WITHIN\_ROW,** R15) || Branch if this row has valid bits

BR(**HANDLE\_DOWN\_LOOP**) || Loop and shift to next row if no valid bits

**HANDLE\_DOWN\_OUT\_OF\_ROW:** || If down movement lead to out-of-bounds

BR( **POLL\_INPUT\_STAGE** ) || GO back to poll input

**HANDLE\_DOWN\_WITHIN\_ROW:** || If down movement is valid

ADD(R15, R0, R7) || Store temp row to actual selected row

BR(**FIND\_FIRST\_VALID\_BIT**, R14) || Find first valid bit to display, x77bc004c

|| R7 is positioned correctly at the last row

BR( **POLL\_INPUT\_STAGE** ) || Go back to get more input

||**##############**

**HANDLE\_CONFIRM:**  
BR( **FLIP\_UPDATE**, R14 )

|| ################  
|| Reset Player timer based on current player  
SUBC(R4, 1, R3) || To check timer during player selection

ANDC(R8, 15, R1) || Get Player 1 timer number

ANDC(R8, 240, R2) || Get Player 2 timer number

SHRC(R2, 4, R2) || Shift Player 2’s timer to [3:0]

ADD(R15, R15, R8) || Clear R8 to all 0s

BEQ(R3, **PLAYER1\_TIMER\_CONFIRM,** R15) || Branch to handle Player 1

BR( **PLAYER2\_TIMER\_CONFIRM** ) || Branch to handle Player 2

**PLAYER1\_TIMER\_CONFIRM:**

|| If Player 1 confirms on the exact same time as Player 2’s last confirm time

|| Player 1 gets 9 sec while Player 2 gets 5 sec

CMPEQ(R1, R2, R0) || R0 contains if Reg[R1] == Reg[R2]

BNE(R0, **PLAYER1\_TIMER\_CONFIRM\_EQUAL**, R15)

ADD(R15, R1, R8) || Add Player 1’s current time (save this time)

ADDC(R8, 144, R8) || Reset Player 2’s timer back to 9 seconds

BR( **NEXT\_TURN** ) || Go to next turn

|| Player 1 manages to confirm on same time as last player’s 2 timer

**PLAYER1\_TIMER\_CONFIRM\_EQUAL:**

ADDC(R15, 89, R8) || Set b01011001

BR( **NEXT\_TURN** ) || Go to next turn  
  
**PLAYER2\_TIMER\_CONFIRM:**  
|| If Player 2 confirms on the exact same time as Player 1’s last confirm time  
|| Player 2 gets 9 sec while Player 1 gets 5 sec  
CMPEQ(R1, R2, R0) || R0 contains if Reg[R1] == Reg[R2]  
BNE(R0, **PLAYER2\_TIMER\_CONFIRM\_EQUAL**, R15)

SHLC(R2, 4, R2) || Shift the R2 back to [7:4]  
ADD(R15, R2, R8) || Add Player 2’s current time (save this time)

ADDC(R8, 9, R8) || Reset Player 1’s timer back to 9 seconds

BR( **NEXT\_TURN** ) || Go to next turn

|| Player 2 manages to confirm on same time as last player’s 1 timer

**PLAYER2\_TIMER\_CONFIRM\_EQUAL:**

ADDC(R15, 149, R8) || Set b10010101

BR( **NEXT\_TURN** ) || Go to next turn

||**#############################**||**######## UTILITY #########**||**#############################**

||**####################################################**

||**####################################################**

|| To go to next player turn

**NEXT\_TURN:**

**|| Find out which is current player**

SUBC( R4, 1, R0 ) || If Player 1 then R0 will have a value of 0

BEQ( R0, **NEXT\_TURN\_PLAYER1**, R15) || If player 1 then branch

|| If Player 2:

ADDC(R15, 1, R4) || Change to Player 1

BR( **NEXT\_TURN\_GENERATE\_VALID\_GRID** )

|| If Player 1:

**NEXT\_TURN\_PLAYER1:**

ADDC( R15, 2, R4 ) || Change to Player 2

BR( **NEXT\_TURN\_GENERATE\_VALID\_GRID** )

**NEXT\_TURN\_GENERATE\_VALID\_GRID:**

|| Branch to function to generate valid grid for this player

BR( **GENERATE\_VALID\_GRID**, R14 )

BR( **PLAYER\_TURN** )

||**####################################################**

||**####################################################**

|| To go to next player turn (After prev player’s turn has no valid moves)  
|| This function does 1 extra step to check if the next player has valid moves  
|| If the next player does not then the game is over

**NEXT\_TURN\_NO\_VALID\_MOVE:**

**|| Find out which is current player**

SUBC( R4, 1, R0 ) || If Player 1 then R0 will have a value of 0

BEQ( R0, **NEXT\_TURN\_NVM\_PLAYER1**, R15) || If player 1 then branch

|| If Player 2:

ADDC(R15, 1, R4) || Change to Player 1

BR( **NEXT\_TURN\_NVM\_GENERATE\_VALID\_GRID** )

|| If Player 1:

**NEXT\_TURN\_NVM\_PLAYER1:**

ADDC( R15, 2, R4 ) || Change to Player 2

BR( **NEXT\_TURN\_NVM\_GENERATE\_VALID\_GRID** )

**NEXT\_TURN\_NVM\_GENERATE\_VALID\_GRID:**

|| Branch to function to generate valid grid for this player

BR( **GENERATE\_VALID\_GRID**, R14 )

BEQ( R9, **GAME\_OVER**, R15 ) || If next player has no valid moves too then game is over

BR( **PLAYER\_TURN** ) || Else, move to the next turn

||**####################################################**

||**####################################################**

|| This function counts the score and announces the winner  
|| by flashing the whole led matrix with the color of the winner

**GAME\_OVER:**

|| **Strategy: Count each space and increment R2 (P1) and R3(P2) based on player**

**|| with piece on that space**

|| Using R6 to run through whole grid

ADDC( R15, 1, R6 ) || R6 ← 0b1, setup R6

|| Set R2 and R3 (the counters) to 0

ADDC( R15, 0, R2 )

ADDC( R15, 0, R3 )

BR( **GAME\_OVER\_P1** ) || For first check, dont go to shift bit loop yet

**GAME\_OVER\_SHIFT\_BIT:**

|| To shift the bit to the next bit of the grid to check

SHLC( R6, 1, R6 )

|| If shifter bit, the one selecting which bit to check is out of grid then indicate who won!!

BEQ( R6, **GAME\_OVER\_INDICATE**, R15 )

|| If not out of grid then continue checking if Player 1 or Player 2

BR( **GAME\_OVER\_P1** )

|| Check whether space contains Player 1, Player 2, or is empty

**GAME\_OVER\_P1:**

AND(R6, R11, R0) || Check if Player 1

BEQ( R0, **GAME\_OVER\_P2,** R15) || Not Player 1

ADDC(R2, 1, R2) || If Player 1 then increment R2

BR( **GAME\_OVER\_SHIFT\_BIT** ) || Shift to next bit to check

**GAME\_OVER\_P2:**

AND(R6, R10, R0) || Check if Player 2

BEQ( R0, **GAME\_OVER\_SHIFT\_BIT,** R15 ) || Not Player 2

ADDC(R3, 1, R3) || If Player 2 then increment R3

BR( **GAME\_OVER\_SHIFT\_BIT** ) || Shift to next bit to check

**GAME\_OVER\_INDICATE:**

|| Compare between R2 and R3 see who is bigger

CMPEQ( R2, R3, R0 ) || IT IS A DRAW! (if is TRUE)

BNE( R0, **GAME\_OVER\_DISPLAY\_DRAW**, R15 )

CMPLT( R2, R3, R0 ) || Player 2 WON, Player 1 LOST!! (if its TRUE)

BNE( R0, **GAME\_OVER\_DISPLAY\_P2WIN,** R15 )

BR( **GAME\_OVER\_DISPLAY\_P1WIN** )

**GAME\_OVER\_DISPLAY\_DRAW:**

|| Set ONLY the VALID grid to be all ‘1’s…

ADDC( R15, 0, R11 )

ADDC( R15, 0, R10 )

ADDC( R15, 1, R9 )

SHLC( R9, 63, R9 ) || MSB of valid grid is 1

SRA( R9, 63, R9 ) || All of valid grid are 1s

BR( **GAME\_OVER\_LATCH )**

**GAME\_OVER\_DISPLAY\_P1WIN:**

|| Set ONLY the Player 1 grid to be all ‘1’s…

ADDC( R15, 0, R10 )

ADDC( R15, 0, R9 )

ADDC( R15, 1, R11 )

SHLC( R11, 63, R11 ) || MSB of Player 1’s grid is 1

SRA( R11, 63, R11 ) || All of Player 1’s grid are 1s

BR( **GAME\_OVER\_LATCH** )

**GAME\_OVER\_DISPLAY\_P2WIN:**

|| Set ONLY the Player 2 grid to be all ‘1’s…

ADDC( R15, 0, R11 )

ADDC( R15, 0, R9 )

ADDC( R15, 1, R10 )

SHLC( R10, 63, R10 ) || MSB of Player 2’s grid is 1

SRA( R10, 63, R10 ) || All of Player 2’s grid are 1s

BR( **GAME\_OVER\_LATCH** )

**GAME\_OVER\_LATCH:**

**||** Latch indicator until Reset Button (Hardwired to go back to 1st instruction) is pressed

BEQ( R15, **GAME\_OVER\_LATCH**, R15) || #77FFFFFF

||**####################################################**

||**####################################################**

**||** To update R6 to the MSB valid bit of the grid

|| To update current selected row to meet the MSB valid bit of the grid

|| Assumption: Valid grid has been formed for current player

|| Assumption: There is at least 1 valid bit

**FIND\_FIRST\_VALID\_BIT\_GRID:**

**||** Grid-wise first valid bit

|| Makes use of valid grid to find left-most (MSB) valid bit.

|| Set current selected bit (Default)

ADDC(R15, 1, R6) || R6 ← 0b1

SHLC(R6, 63, R6) || R6 ← Most Significant Bit of the grid

|| Check if MSB is a valid bit

AND(R6, R9, R2) || Mask to check if bit is in valid grid

BEQ(R2, **FIND\_FIRST\_VALID\_BIT\_RECURSIVE**, R15) || Branch to loop to find valid bit

|| If we found a valid bit for the MSB, then job done! (DID NOT branch in line above)

BR( **FIND\_FIRST\_VALID\_BIT\_CORRECT\_SELECTED\_ROW** )

**FIND\_FIRST\_VALID\_BIT\_RECURSIVE:**

SHRC(R6, 1, R6) || Shift right by 1 to get next bit

AND(R6, R9, R2) || Mask to check if bit is in valid grid

BEQ(R2, **FIND\_FIRST\_VALID\_BIT\_RECURSIVE**, R15) || Branch to loop to find valid bit

BR( **FIND\_FIRST\_VALID\_BIT\_CORRECT\_SELECTED\_ROW** )

**FIND\_FIRST\_VALID\_BIT\_CORRECT\_SELECTED\_ROW:**

|| Move selected row bits to the correct row now that we have found a valid bit

ADDC(R15, 255, R7) || Append all ‘1’s to the last row of the grid

AND(R7, R6, R2) || Check if selected row is where current selected bit is

|| R2 is 0 if we have not found the right selected row

BEQ(R2, **FIND\_FIRST\_VALID\_BIT\_CORRECT\_SELECTED\_ROW\_RECURSIVE**, R15)

|| If we have found the correct selected row, give back control

JMP( **R14** )

**FIND\_FIRST\_VALID\_BIT\_CORRECT\_SELECTED\_ROW\_RECURSIVE:**

SHLC(R7, 8, R7) || Move to next row (upwards in grid)

AND(R7, R6, R2) || Check if selected row is where current selected bit is

|| R2 is 0 if we have not found the right selected row

BEQ(R2, **FIND\_FIRST\_VALID\_BIT\_CORRECT\_SELECTED\_ROW\_RECURSIVE**, R15)

|| If we have found the correct selected row, give back control

JMP( **R14** )

||**####################################################**

||**####################################################**

**||** To update R6 to the MSB valid bit of current selected row

|| Assumption: Valid grid has been formed for current player

*|| Assumption: There is at least 1 valid bit in the current selected row*

**FIND\_FIRST\_VALID\_BIT:**

|| Set Selected bit to 0b1…

ADDC(R15, 1, R6)

|| Check if current selected bit is in the selected row

AND(R6, R7, R2) || Masking to check if current selected bit is in selected row

BEQ(R2, **FVB\_ROW\_RECURSIVE**, R15) || Selected bit is not in the selected row

BR( **FVB\_GET\_BIT** )

**FVB\_ROW\_RECURSIVE:**

SHLC(R6, 8, R6) || Shift to next row (Upwards in grid)

AND(R6, R7, R2) || Masking to check if current selected bit is in selected row

BEQ(R2, **FVB\_ROW\_RECURSIVE**, R15) || Selected bit is not in the selected row

BR( **FVB\_GET\_BIT** )

**FVB\_GET\_BIT:**

|| Purpose: To shift selected bit to the rightmost valid bit of the selected row

SHLC( R6, 7, R6 ) || Shift selected bit to get the MSB of the current row

AND( R6, R9, R2 ) || Mask with valid grid to see if selected bit is on a valid bit

BEQ( R2, **FVB\_GET\_BIT\_RECURSIVE,** R15 ) || Selected bit not in valid grid

|| If did not branch: Selected bit is a valid bit

JMP( **R14** )

**FVB\_GET\_BIT\_RECURSIVE:**

SHRC(R6, 1, R6) || Shift selected bit to the right (on the row) to get next bit

AND( R6, R9, R2 ) || Mask with valid grid to see if selected bit is on a valid bit

BEQ( R2, **FVB\_GET\_BIT\_RECURSIVE,** R15 ) || Selected bit not in valid grid

JMP( **R14** )

||**####################################################**

||**####################################################**

**||** To update R6 to the LSB valid bit of current selected row

|| Assumption: Valid grid has been formed for current player

*|| Assumption: There is at least 1 valid bit in the current selected row*

**FIND\_LAST\_VALID\_BIT:**

ADDC(R15, 1, R6) || Set selected bit to 0b1

|| Check if current selected bit is in the selected row

AND(R6, R7, R2) || Masking to check if current selected bit is in selected row

BEQ(R2, **LVB\_ROW\_RECURSIVE**, R15) || Selected bit is not in the selected row

BR( **LVB\_GET\_BIT** )

**LVB\_ROW\_RECURSIVE:**

SHLC(R6, 8, R6) || Shift to next row (Upwards in grid)

AND(R6, R7, R2) || Masking to check if current selected bit is in selected row

BEQ(R2, **LVB\_ROW\_RECURSIVE**, R15) || Selected bit is not in the selected row

BR( **LVB\_GET\_BIT** )

**LVB\_GET\_BIT:**

|| Purpose: To shift selected bit to the rightmost valid bit of the selected row

AND( R6, R9, R2 ) || Mask with valid grid to see if selected bit is on a valid bit

BEQ( R2, **LVB\_GET\_BIT\_RECURSIVE,** R15 ) || Selected bit not in valid grid

|| If did not branch: Selected bit is a valid bit

JMP( **R14** )

**LVB\_GET\_BIT\_RECURSIVE:**

SHLC(R6, 1, R6) || Shift selected bit to the left (on the row) to get next bit

AND( R6, R9, R2 ) || Mask with valid grid to see if selected bit is on a valid bit

BEQ( R2, **LVB\_GET\_BIT\_RECURSIVE,** R15 ) || Selected bit not in valid grid

JMP( **R14** )

||**####################################################**

||**####################################################**

**FLIP\_UPDATE:**

**|| This function** finds the valid pieces that can be flipped for that particular valid bit.

|| Firstly, check which player is currently having his/her turn

SUBC(R4, 1, R0) || If Player 1 then R0 will be 0

BEQ(R0, **FLIP\_UPDATE\_PLAYER1,** R15)

|| if Player 2:

**FLIP\_UPDATE\_PLAYER2:**

ADD(R15, R10, R13) || Update Own Grid Register

ADD(R15, R11, R12) || Update Opponent Grid Register

BR( **CHECK\_LEFT\_FLIP, R3** )

ADD( R15, R13, R10 ) || Update Player 2 Grid Register

ADD( R15, R12, R11 ) || Update Player 1 Grid Register

JMP( **R14** ) || return control

|| if Player 1:

**FLIP\_UPDATE\_PLAYER1:**

ADD(R15, R11, R13) || Update Own Grid Register

ADD(R15, R10, R12) || Update Opponent Grid Register

BR( **CHECK\_LEFT\_FLIP, R3** )

ADD( R15, R13, R11 ) || Update Player 1 Grid Register

ADD( R15, R12, R10 ) || Update Player 2 Grid Register

JMP( **R14** ) || return control

**CHECK\_LEFT\_FLIP:**

SHLC(R6, 1, R1) || Shift current selection mask to left by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row

|| Check if we are out of bounds for left bound

BEQ(R0, **CHECK\_LEFT\_UP\_FLIP**, R15)

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent in line-of-sight

BEQ(R0, **CHECK\_LEFT\_UP\_FLIP**, R15) || No opponent in line-of-sight

BR( **CHECK\_LEFT\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find own piece

**CHECK\_LEFT\_INNERLOOP1:**

SHLC(R1, 1, R1) || Shift current selection mask to left by 1

AND(R1, R7, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for left bound (No left-directed own piece)

BEQ(R0, **CHECK\_LEFT\_UP\_FLIP**, R15)

|| Check if this piece is opponent’s piece

AND(R1, R12, R0) || Masking current pointer with opponent grid

BNE(R0, **CHECK\_LEFT\_INNERLOOP1**, R15) || if is opponent piece then SHL to next bit

|| Check if this piece is own piece

AND(R1, R13, R0) || Masking current pointer with own grid

|| If this piece is not P1 then we branch (Empty space)

BEQ(R0, **CHECK\_LEFT\_UP\_FLIP**, R15)

BR( **CHECK\_LEFT\_BACKWARD** )

|| Else: From this point on we found own piece on left now move back flipping pieces

|| till we found the own piece again

**CHECK\_LEFT\_BACKWARD:**

SHRC( R1, 1, R1 ) || Shift R1 current pointer to right by 1 bit

AND( R1, R12, R0 ) || Check if the current pointed piece is Opponent by masking

BNE( R0, **CHECK\_LEFT\_INNERLOOP2**, R15 ) || If Opponent then continue

**CHECK\_LEFT\_INNERLOOP2:**

XOR( R1, R12, R12 ) || Flip Opponent piece ( Flip to Opponent piece )

OR( R1, R13, R13 ) || Flip Own piece ( Turn to own piece )

SHRC( R1, 1, R1 ) || Shift the current pointer right since we have not found own piece

AND( R1, R12, R0 ) || Mask with Opponent piece

BNE( R0, **CHECK\_LEFT\_INNERLOOP2**, R15 ) || If Opponent then continue

BR( **CHECK\_LEFT\_UP\_FLIP** ) || Flipped all and have reached Own piece

**CHECK\_LEFT\_UP\_FLIP:**

SHLC(R6, 1, R1) || Shift current selection mask to left by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row, hA005C000

|| Check if we are out of bounds for left bound

BEQ(R0, **CHECK\_UP\_FLIP**, R15)

SHLC(R1, 8, R1) || Shift current selection mask to left by 8 (Initialise Pointer)

SHLC(R7, 8, R2) || Shift current row selection mask to left (Initialise Pointer)

|| R2 → Pointer for row selection, R1 → Pointer for current selection

BEQ(R1, **CHECK\_UP\_FLIP**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **CHECK\_UP\_FLIP**, R15) || No opponent in line-of-sight

BR( **CHECK\_LEFT\_UP\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_LEFT\_UP\_INNERLOOP1:**

SHLC(R1, 1, R1) || Shift current selection mask to left by 1

AND(R1, R2, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for left bound (No left-directed own piece)

BEQ(R0, **CHECK\_UP\_FLIP**, R15)

SHLC(R1, 8, R1) || Shift current selection mask to left by 8 (next row)

SHLC(R2, 8, R2) || Shift current row selection mask to left by 8 (next row)

BEQ(R1, **CHECK\_UP\_FLIP**, R15) || Check if Out-Of-Bounds (Up-Down)

|| Check if this piece is opponent’s piece

AND(R1, R12, R0) || Masking current pointer with opponent’s grid

BNE(R0, **CHECK\_LEFT\_UP\_INNERLOOP1**, R15) || If opponent piece, then get next pcs

|| Check if this piece is own piece

AND(R1, R13, R0) || Masking current pointer with own grid

|| If this piece is not own then we branch (Empty space)

BEQ(R0, **CHECK\_UP\_FLIP**, R15)

BR( **CHECK\_LEFT\_UP\_BACKWARD** )

|| Else: From this point on we found P1 piece on top-left, now move back flipping pieces

|| till we found the P1 piece again

**CHECK\_LEFT\_UP\_BACKWARD:**

SHRC( R1, 8, R1 ) || Move down one row

SHRC( R2, 8, R2 ) || Shift current row selection mask to right (prev row)

SHRC( R1, 1, R1 ) || Shift R1 current pointer to right by 1 bit

AND( R1, R12, R0 ) || Check if the current pointed piece is Opponent by masking

BNE( R0, **CHECK\_LEFT\_UP\_INNERLOOP2**, R15 ) || If Opponent then continue

**CHECK\_LEFT\_UP\_INNERLOOP2:**

XOR( R1, R12, R12 ) || Flip Opponent piece ( Flip to Opponent piece )

OR( R1, R13, R13 ) || Flip Own piece ( Turn to own piece )

SHRC( R1, 8, R1 ) || Shift the current pointer right since we have not found own piece

SHRC( R2, 8, R2 ) || Shift current row selection mask to right (prev row)

SHRC( R1, 1, R1 ) || Shift the current pointer right since we have not found own piece

AND( R1, R12, R0 ) || Mask with Opponent piece

BNE( R0, **CHECK\_LEFT\_UP\_INNERLOOP2**, R15 ) || If Opponent then continue

BR( **CHECK\_UP\_FLIP** ) || Flipped all and have reached Own piece

**CHECK\_UP\_FLIP:**

SHLC(R6, 8, R1) || Shift current selection mask to left by 8 (Initialise Pointer)

BEQ(R1, **CHECK\_RIGHT\_UP\_FLIP**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **CHECK\_RIGHT\_UP\_FLIP**, R15) || No opponent in line-of-sight

BR( **CHECK\_UP\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_UP\_INNERLOOP1:**

SHLC(R1, 8, R1) || Shift current selection mask to left by 8 (next row)

BEQ(R1, **CHECK\_RIGHT\_UP\_FLIP**, R15) || Check if Out-Of-Bounds

|| Check if this piece is P2’s piece

AND(R1, R12, R0) || Masking current pointer with P2 grid

BNE(R0, **CHECK\_UP\_INNERLOOP1**, R15) || if it is P2 piece then move up 1 row

|| Check if this piece is P1’s piece

AND(R1, R13, R0) || Masking current pointer with P1 grid

|| If this piece is not P1 then we branch (Empty space)

BEQ(R0, **CHECK\_RIGHT\_UP\_FLIP**, R15)

BR( **CHECK\_UP\_BACKWARD** )

|| Else: From this point on we found P1 piece on top, now move back flipping pieces

|| till we found the P1 piece again

**CHECK\_UP\_BACKWARD:**

SHRC( R1, 8, R1 ) || Move down one row

AND( R1, R12, R0 ) || Check if the current pointed piece is Opponent by masking

BNE( R0, **CHECK\_UP\_INNERLOOP2**, R15 ) || If Opponent then continue

**CHECK\_UP\_INNERLOOP2:**

XOR( R1, R12, R12 ) || Flip Opponent piece ( Flip to Opponent piece )

OR( R1, R13, R13 ) || Flip Own piece ( Turn to own piece )

SHRC( R1, 8, R1 ) || Shift the current pointer right since we have not found own piece

AND( R1, R12, R0 ) || Mask with Opponent piece

BNE( R0, **CHECK\_UP\_INNERLOOP2**, R15 ) || If Opponent then continue

BR( **CHECK\_RIGHT\_UP\_FLIP** ) || Flipped all and have reached Own piece

**CHECK\_RIGHT\_UP\_FLIP:**

SHRC(R6, 1, R1) || Shift current selection mask to right by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row

|| Check if we are out of bounds for right bound

BEQ(R0, **CHECK\_RIGHT\_FLIP**, R15)

SHLC(R1, 8, R1) || Shift current selection mask to left by 8 (Initialise Pointer)

SHLC(R7, 8, R2) || Shift current row selection mask to left (Initialise Pointer)

|| R2 → Pointer for row selection, R1 → Pointer for current selection

BEQ(R1, **CHECK\_RIGHT\_FLIP**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **CHECK\_RIGHT\_FLIP**, R15) || No opponent in line-of-sight

BR( **CHECK\_RIGHT\_UP\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_RIGHT\_UP\_INNERLOOP1:**

SHRC(R1, 1, R1) || Shift current selection mask to right by 1

AND(R1, R2, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for right bound (No right-directed own piece)

BEQ(R0, **CHECK\_RIGHT\_FLIP**, R15)

SHLC(R1, 8, R1) || Shift current selection mask to right by 8 (next row)

SHLC(R2, 8, R2) || Shift current row selection mask to right by 8 (next row)

BEQ(R1, **CHECK\_RIGHT\_FLIP**, R15) || Check if Out-Of-Bounds (Up-Down)

|| Check if this piece is P2’s piece

AND(R1, R12, R0) || Masking current pointer with P2 grid

BNE(R0, **CHECK\_RIGHT\_UP\_INNERLOOP1**, R15) || If P2 piece, then get next pcs

|| Check if this piece is P1’s piece

AND(R1, R13, R0) || Masking current pointer with P1 grid

|| If this piece is not P1 then we branch (Empty space)

BEQ(R0, **CHECK\_RIGHT\_FLIP**, R15)

BR( **CHECK\_RIGHT\_UP\_BACKWARD** )

|| Else: From this point on we found P1 piece on top-right, now move back flipping pieces

|| till we found the P1 piece again

**CHECK\_RIGHT\_UP\_BACKWARD:**

SHRC( R1, 8, R1 ) || Move down one row

SHRC( R2, 8, R2 ) || Shift current row selection mask to right (prev row)

SHLC( R1, 1, R1 ) || Shift R1 current pointer to left by 1 bit

AND( R1, R12, R0 ) || Check if the current pointed piece is Opponent by masking

BNE( R0, **CHECK\_RIGHT\_UP\_INNERLOOP2**, R15 ) || If Opponent then continue

**CHECK\_RIGHT\_UP\_INNERLOOP2:**

XOR( R1, R12, R12 ) || Flip Opponent piece ( Flip to Opponent piece )

OR( R1, R13, R13 ) || Flip Own piece ( Turn to own piece )

SHRC( R1, 8, R1 ) || Shift the current pointer left since we have not found own piece

SHRC( R2, 8, R2 ) || Shift current row selection mask to right (prev row)

SHLC( R1, 1, R1 ) || Shift current pointer left (row) since we have not found own piece

AND( R1, R12, R0 ) || Mask with Opponent piece

BNE( R0, **CHECK\_RIGHT\_UP\_INNERLOOP2**, R15 ) || If Opponent then continue

BR( **CHECK\_RIGHT\_FLIP** ) || Flipped all and have reached Own piece

**CHECK\_RIGHT\_FLIP:**

SHRC(R6, 1, R1) || Shift current selection mask to right by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row

|| Check if we are out of bounds for right bound

BEQ(R0, **CHECK\_RIGHT\_DOWN\_FLIP**, R15)

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent in line-of-sight

BEQ(R0, **CHECK\_RIGHT\_DOWN\_FLIP**, R15) || No opponent in line-of-sight

BR( **CHECK\_RIGHT\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_RIGHT\_INNERLOOP1:**

SHRC(R1, 1, R1) || Shift current selection mask to right by 1

AND(R1, R7, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for right bound (No right-directed own piece)

BEQ(R0, **CHECK\_RIGHT\_DOWN\_FLIP**, R15)

|| Check if this piece is P2’s piece

AND(R1, R12, R0) || Masking current pointer with P2 grid

BNE(R0, **CHECK\_RIGHT\_INNERLOOP1**, R15) || if is P2 piece then SHR to next bit

|| Check if this piece is P1’s piece

AND(R1, R13, R0) || Masking current pointer with P1 grid

|| If this piece is not P1 then we branch (Empty space)

BEQ(R0, **CHECK\_RIGHT\_DOWN\_FLIP**, R15)

BR( **CHECK\_RIGHT\_BACKWARD** )

|| Else: From this point on we found P1 piece on right now move back flipping pieces

|| till we found the P1 piece again

**CHECK\_RIGHT\_BACKWARD:**

SHLC( R1, 1, R1 ) || Shift R1 current pointer to left by 1 bit

AND( R1, R12, R0 ) || Check if the current pointed piece is Opponent by masking

BNE( R0, **CHECK\_RIGHT\_INNERLOOP2**, R15 ) || If Opponent then continue

**CHECK\_RIGHT\_INNERLOOP2:**

XOR( R1, R12, R12 ) || Flip Opponent piece ( Flip to Opponent piece )

OR( R1, R13, R13 ) || Flip Own piece ( Turn to own piece )

SHLC( R1, 1, R1 ) || Shift the current pointer left since we have not found own piece

AND( R1, R12, R0 ) || Mask with Opponent piece

BNE( R0, **CHECK\_RIGHT\_INNERLOOP2**, R15 ) || If Opponent then continue

BR( **CHECK\_RIGHT\_DOWN\_FLIP** ) || Flipped all and have reached Own piece

**CHECK\_RIGHT\_DOWN\_FLIP:**

SHRC(R6, 1, R1) || Shift current selection mask to right by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row

|| Check if we are out of bounds for right bound

BEQ(R0, **CHECK\_DOWN\_FLIP**, R15)

SHRC(R1, 8, R1) || Shift current selection mask to right by 8 (Initialise Pointer)

SHRC(R7, 8, R2) || Shift current row selection mask to right (Initialise Pointer)

|| R2 → Pointer for row selection, R1 → Pointer for current selection

BEQ(R1, **CHECK\_DOWN\_FLIP**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **CHECK\_DOWN\_FLIP**, R15) || No opponent in line-of-sight

BR( **CHECK\_RIGHT\_DOWN\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_RIGHT\_DOWN\_INNERLOOP1:**

SHRC(R1, 1, R1) || Shift current selection mask to right by 1

AND(R1, R2, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for right bound (No right-directed own piece)

BEQ(R0, **CHECK\_DOWN\_FLIP**, R15)

SHRC(R1, 8, R1) || Shift current selection mask to right by 8 (next row)

SHRC(R2, 8, R2) || Shift current row selection mask to right by 8 (next row)

BEQ(R1, **CHECK\_DOWN\_FLIP**, R15) || Check if Out-Of-Bounds (Up-Down)

|| Check if this piece is P2’s piece

AND(R1, R12, R0) || Masking current pointer with P2 grid

|| If P2 piece, then get next pcs

BNE(R0, **CHECK\_RIGHT\_DOWN\_INNERLOOP1**, R15)

|| Check if this piece is P1’s piece

AND(R1, R13, R0) || Masking current pointer with P1 grid

|| If this piece is not P1 then we branch (Empty space)

BEQ(R0, **CHECK\_DOWN\_FLIP**, R15)

BR( **CHECK\_RIGHT\_DOWN\_BACKWARD** )

|| Else: From this point on we found P1 piece on bot-right, now move back flipping pieces

|| till we found the P1 piece again

**CHECK\_RIGHT\_DOWN\_BACKWARD:**

SHLC( R1, 8, R1 ) || Move down one row

SHLC( R2, 8, R2 ) || Shift current row selection mask to left (prev row)

SHLC( R1, 1, R1 ) || Shift R1 current pointer to left by 1 bit

AND( R1, R12, R0 ) || Check if the current pointed piece is Opponent by masking

BNE( R0, **CHECK\_RIGHT\_DOWN\_INNERLOOP2**, R15 ) || If Opponent then continue

**CHECK\_RIGHT\_DOWN\_INNERLOOP2:**

XOR( R1, R12, R12 ) || Flip Opponent piece ( Flip to Opponent piece )

OR( R1, R13, R13 ) || Flip Own piece ( Turn to own piece )

SHLC( R1, 8, R1 ) || Shift the current pointer left since we have not found own piece

SHLC( R2, 8, R2 ) || Shift current row selection mask to left (prev row)

SHLC( R1, 1, R1 ) || Shift current pointer left (row) since we have not found own piece

AND( R1, R12, R0 ) || Mask with Opponent piece

BNE( R0, **CHECK\_RIGHT\_DOWN\_INNERLOOP2**, R15 ) || If Opponent then continue

BR( **CHECK\_DOWN\_FLIP** ) || Flipped all and have reached Own piece

**CHECK\_DOWN\_FLIP:**

SHRC(R6, 8, R1) || Shift current selection mask to right by 8 (Initialise Pointer)

BEQ(R1, **CHECK\_LEFT\_DOWN\_FLIP**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **CHECK\_LEFT\_DOWN\_FLIP**, R15) || No opponent in line-of-sight

BR( **CHECK\_DOWN\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_DOWN\_INNERLOOP1:**

SHRC(R1, 8, R1) || Shift current selection mask to right by 8 (next row)

BEQ(R1, **CHECK\_LEFT\_DOWN\_FLIP**, R15) || Check if Out-Of-Bounds

|| Check if this piece is P2’s piece

AND(R1, R12, R0) || Masking current pointer with P2 grid

|| if is P2 piece then move down 1 row

BNE(R0, **CHECK\_DOWN\_INNERLOOP1**, R15)

|| Check if this piece is P1’s piece

AND(R1, R13, R0) || Masking current pointer with P1 grid

|| If this piece is not P1 then we branch (Empty space)

BEQ(R0, **CHECK\_LEFT\_DOWN\_FLIP**, R15)

BR( **CHECK\_DOWN\_BACKWARD** )

|| Else: From this point on we found P1 piece on bottom, now move back flipping pieces

|| till we found the P1 piece again

**CHECK\_DOWN\_BACKWARD:**

SHLC( R1, 8, R1 ) || Move down one row

AND( R1, R12, R0 ) || Check if the current pointed piece is Opponent by masking

BNE( R0, **CHECK\_DOWN\_INNERLOOP2**, R15 ) || If Opponent then continue

**CHECK\_DOWN\_INNERLOOP2:**

XOR( R1, R12, R12 ) || Flip Opponent piece ( Flip to Opponent piece )

OR( R1, R13, R13 ) || Flip Own piece ( Turn to own piece )

SHLC( R1, 8, R1 ) || Shift the current pointer left since we have not found own piece

AND( R1, R12, R0 ) || Mask with Opponent piece

BNE( R0, **CHECK\_DOWN\_INNERLOOP2**, R15 ) || If Opponent then continue

BR( **CHECK\_LEFT\_DOWN\_FLIP** ) || Flipped all and have reached Own piece

**CHECK\_LEFT\_DOWN\_FLIP:**

SHLC(R6, 1, R1) || Shift current selection mask to left by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row

|| Check if we are out of bounds for left bound return control to main program

BEQ(R0, **CHECK\_FINAL**, R15)

SHRC(R1, 8, R1) || Shift current selection mask to right by 8 (Initialise Pointer)

SHRC(R7, 8, R2) || Shift current row selection mask to right (Initialise Pointer)

|| R2 → Pointer for row selection, R1 → Pointer for current selection

BEQ(R1, **CHECK\_FINAL**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **CHECK\_FINAL**, R15) || No opponent in line-of-sight

BR( **CHECK\_LEFT\_DOWN\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_LEFT\_DOWN\_INNERLOOP1:**

SHLC(R1, 1, R1) || Shift current selection mask to left by 1

AND(R1, R2, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for left bound (No right-directed own piece)

BEQ(R0, **CHECK\_FINAL**, R15)

SHRC(R1, 8, R1) || Shift current selection mask to right by 8 (next row)

SHRC(R2, 8, R2) || Shift current row selection mask to right by 8 (next row)

BEQ(R1, **CHECK\_FINAL**, R15) || Check if Out-Of-Bounds (Up-Down)

|| Check if this piece is opponent’s piece

AND(R1, R12, R0) || Masking current pointer with opponent’s grid

|| If opponent piece, then get next pcs

BNE(R0, **CHECK\_LEFT\_DOWN\_INNERLOOP1**, R15)

|| Check if this piece is own piece

AND(R1, R13, R0) || Masking current pointer with own grid

|| If this piece is not P1 then we branch back to main program (Empty space)

BEQ(R0, **CHECK\_FINAL**, R15)

BR( **CHECK\_LEFT\_DOWN\_BACKWARD** )

|| Else: From this point on we found P1 piece on bot-left, now move back flipping pieces

|| till we found the P1 piece again

**CHECK\_LEFT\_DOWN\_BACKWARD:**

SHLC( R1, 8, R1 ) || Move up one row

SHLC( R2, 8, R2 ) || Shift current row selection mask to right (prev row)

SHRC( R1, 1, R1 ) || Shift R1 current pointer to right by 1 bit

AND( R1, R12, R0 ) || Check if the current pointed piece is Opponent by masking

BNE( R0, **CHECK\_LEFT\_DOWN\_INNERLOOP2**, R15 ) || If Opponent then continue

**CHECK\_LEFT\_DOWN\_INNERLOOP2:**

XOR( R1, R12, R12 ) || Flip Opponent piece ( Flip to Opponent piece )

OR( R1, R13, R13 ) || Flip Own piece ( Turn to own piece )

SHLC( R1, 8, R1 ) || Shift the current pointer right since we have not found own piece

SHLC( R2, 8, R2 ) || Shift current row selection mask to right (prev row)

SHRC( R1, 1, R1 ) || Shift current pointer right (row) since we have not found own piece

AND( R1, R12, R0 ) || Mask with Opponent piece

BNE( R0, **CHECK\_LEFT\_DOWN\_INNERLOOP2**, R15 ) || If Opponent then continue

OR( R1, R13, R13) || Flip Empty Valid space

BR( CHECK\_FINAL )

**CHECK\_FINAL:**

OR( R6, R13, R13) || Flip empty valid space (Selected Spot)

JMP( **R3** ) || Flipped all and have reached Own piece

||**####################################################**

||**####################################################**

**GENERATE\_VALID\_GRID:**

|| 2 Apr 2023

|| Generate the valid moves grid based on the current player

|| Firstly, clean the valid grid

ADDC(R15, 0, R9)

|| Secondly, check which player is currently having his/her turn

SUBC(R4, 1, R0) || If Player 1 then R0 will be 0

BEQ(R0, **GVG\_SETUP\_PLAYER1,** R15)

|| Player 2 Setup:

**GVG\_SETUP\_PLAYER2:**

ADD(R15, R10, R13) || Update Own Grid Register

ADD(R15, R11, R12) || Update Opponent Grid Register

BR( **GVG\_CONT** )

|| Player 1 Setup:

**GVG\_SETUP\_PLAYER1:**

ADD(R15, R11, R13) || Update Own Grid Register

ADD(R15, R10, R12) || Update Opponent Grid Register

BR( **GVG\_CONT** )

|| Temporarily use Selection Bit R6, since new turn  
**GVG\_CONT:**  
ADDC(R15, 1, R6) || Setup R6 ← 0b1

ADDC(R15, 255, R7) || Setup R7 🡨 0b11111111 to select last row for selected bit

**|| Strategy:** Shift left selection bit, check if selection bit is on an empty grid, if it is then do the 8 directional check to see if this bit is valid for current player, if valid then set valid grid.

AND(R6, R13, R0) || Initial Check to see if bit is on empty space

AND(R6, R12, R1) || Cont. of check above (opponent grid)

OR( R0, R1, R0 ) || See if either own or opponent grid

BEQ(R0, **CHECK\_IF\_VALID**, R15) || Empty space => execute loop to check if it’s valid

|| If not empty then move left (on grid) to get the next bit

SHLC(R6, 1, R6)

BEQ( R6, **GVG\_DONE**, R15 ) || To check if we have checked all bits, out of grid

AND( R6, R13, R0 ) || Check if current bit is on empty space

AND( R6, R12, R1 ) || Cont. of check above (opponent grid)

OR( R0, R1, R0 ) || See if either own or opponent grid

BEQ( R0, **CHECK\_IF\_VALID**, R15) || If is own bit then execute loop to check if it’s valid

BNE( R0, **GVG\_BIT\_SHIFT**, R15) || If not, then shift bit to left and loop until out of grid

**GVG\_BIT\_SHIFT:**

SHLC( R6, 1, R6 )

BEQ( R6, **GVG\_DONE**, R15 ) || To check if we have checked all bits, out of grid

AND( R6, R7, R0 ) || Check if selected row is still corresponds to check bit

BNE( R0, **GVG\_BIT\_SHIFT\_CONT**, R15 ) || If correct selected row then branch

SHLC( R7, 8, R7 ) || (not correct row) Shift current selected row to next row (Upwards)

**GVG\_BIT\_SHIFT\_CONT:**

AND( R6, R13, R0 ) || Check if current bit is on empty space

AND( R6, R12, R1 ) || Cont. of check above (opponent grid)

OR( R0, R1, R0 ) || See if either own or opponent grid

BEQ( R0, **CHECK\_IF\_VALID**, R15) || If is own bit then execute loop to check if it’s valid

BNE( R0, **GVG\_BIT\_SHIFT**, R15) || If not, then shift bit to left and loop until out of grid

**GVG\_DONE:**

JMP( **R14** ) || Jump back to main function

**CHECK\_IF\_VALID:**

BR( **CHECK\_LEFT\_VALID** ) || Go to first check function

**CHECK\_LEFT\_VALID:**

SHLC(R6, 1, R1) || Shift current selection mask to left by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row

|| Check if we are out of bounds for left bound

BEQ(R0, **CHECK\_LEFT\_UP\_VALID**, R15)

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent in line-of-sight

BEQ(R0, **CHECK\_LEFT\_UP\_VALID**, R15) || No opponent in line-of-sight

BR( **CHECK\_LEFT\_VALID\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent

**CHECK\_LEFT\_VALID\_INNERLOOP1:**

SHLC(R1, 1, R1) || Shift current selection mask to left by 1

AND(R1, R7, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for left bound (No left-directed own piece)

BEQ(R0, **CHECK\_LEFT\_UP\_VALID**, R15)

|| Check if this piece is opponent’s piece

AND(R1, R12, R0) || Masking current pointer with opponent grid

|| if is opponent piece then SHL to next bit

BNE(R0, **CHECK\_LEFT\_VALID\_INNERLOOP1**, R15)

|| Check if this piece is own piece

AND(R1, R13, R0) || Masking current pointer with own grid

|| If this piece is not P1 then we branch (Empty space)

BEQ(R0, **CHECK\_LEFT\_UP\_VALID**, R15)

BR( **MARK\_VALID** )

**CHECK\_LEFT\_UP\_VALID:**

SHLC(R6, 1, R1) || Shift current selection mask to left by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row

|| Check if we are out of bounds for left bound

BEQ(R0, **CHECK\_UP\_VALID**, R15)

SHLC(R1, 8, R1) || Shift current selection mask to left by 8 (Initialise Pointer)

SHLC(R7, 8, R2) || Shift current row selection mask to left (Initialise Pointer)

|| R2 → Pointer for row selection, R1 → Pointer for current selection

BEQ(R1, **CHECK\_UP\_VALID**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **CHECK\_UP\_VALID**, R15) || No opponent in line-of-sight

BR( **CHECK\_LEFT\_UP\_VALID\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_LEFT\_UP\_VALID\_INNERLOOP1:**

SHLC(R1, 1, R1) || Shift current selection mask to left by 1

AND(R1, R2, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for left bound (No left-directed own piece)

BEQ(R0, **CHECK\_UP\_VALID**, R15)

SHLC(R1, 8, R1) || Shift current selection mask to left by 8 (next row)

SHLC(R2, 8, R2) || Shift current row selection mask to left by 8 (next row)

BEQ(R1, **CHECK\_UP\_VALID**, R15) || Check if Out-Of-Bounds (Up-Down)

|| Check if this piece is opponent’s piece

AND(R1, R12, R0) || Masking current pointer with opponent grid

|| If opponent piece, then get next pcs

BNE(R0, **CHECK\_LEFT\_UP\_VALID\_INNERLOOP1**, R15)

|| Check if this piece is own piece

AND(R1, R13, R0) || Masking current pointer with own grid

|| If this piece is not P1 then we branch (Empty space)

BEQ(R0, **CHECK\_UP\_VALID**, R15)

BR( **MARK\_VALID** )

**CHECK\_UP\_VALID:**

SHLC(R6, 8, R1) || Shift current selection mask to left by 8 (Initialise Pointer)

BEQ(R1, **CHECK\_RIGHT\_UP\_VALID**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **CHECK\_RIGHT\_UP\_VALID**, R15) || No opponent in line-of-sight

BR( **CHECK\_UP\_VALID\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_UP\_VALID\_INNERLOOP1:**

SHLC(R1, 8, R1) || Shift current selection mask to left by 8 (next row)

BEQ(R1, **CHECK\_RIGHT\_UP\_VALID**, R15) || Check if Out-Of-Bounds

|| Check if this piece is opponent’s piece

AND(R1, R12, R0) || Masking current pointer with opponent grid

|| if it is opponent piece then move up 1 row

BNE(R0, **CHECK\_UP\_VALID\_INNERLOOP1**, R15)

|| Check if this piece is own piece

AND(R1, R13, R0) || Masking current pointer with own grid

|| If this piece is not P1 then we branch (Empty space)

BEQ(R0, **CHECK\_RIGHT\_UP\_VALID**, R15)

BR( **MARK\_VALID** )

**CHECK\_RIGHT\_UP\_VALID:**

SHRC(R6, 1, R1) || Shift current selection mask to right by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row

|| Check if we are out of bounds for right bound

BEQ(R0, **CHECK\_RIGHT\_VALID**, R15)

SHLC(R1, 8, R1) || Shift current selection mask to left by 8 (Initialise Pointer)

SHLC(R7, 8, R2) || Shift current row selection mask to left (Initialise Pointer)

|| R2 → Pointer for row selection, R1 → Pointer for current selection

BEQ(R1, **CHECK\_RIGHT\_VALID**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **CHECK\_RIGHT\_VALID**, R15) || No opponent in line-of-sight

BR( **CHECK\_RIGHT\_UP\_VALID\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_RIGHT\_UP\_VALID\_INNERLOOP1:**

SHRC(R1, 1, R1) || Shift current selection mask to right by 1

AND(R1, R2, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for right bound (No right-directed own piece)

BEQ(R0, **CHECK\_RIGHT\_VALID**, R15)

SHLC(R1, 8, R1) || Shift current selection mask to right by 8 (next row)

SHLC(R2, 8, R2) || Shift current row selection mask to right by 8 (next row)

BEQ(R1, **CHECK\_RIGHT\_VALID**, R15) || Check if Out-Of-Bounds (Up-Down)

|| Check if this piece is OPPONENT’s piece

AND(R1, R12, R0) || Masking current pointer with opponent’s grid

|| If opponent piece, then get next pcs

BNE(R0, **CHECK\_RIGHT\_UP\_VALID\_INNERLOOP1**, R15)

|| Check if this piece is own piece

AND(R1, R13, R0) || Masking current pointer with own grid

|| If this piece is not own then we branch (Empty space)

BEQ(R0, **CHECK\_RIGHT\_VALID**, R15)

BR( **MARK\_VALID** )

**CHECK\_RIGHT\_VALID:**

SHRC(R6, 1, R1) || Shift current selection mask to right by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row

|| Check if we are out of bounds for right bound

BEQ(R0, **CHECK\_RIGHT\_DOWN\_VALID**, R15)

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent in line-of-sight

BEQ(R0, **CHECK\_RIGHT\_DOWN\_VALID**, R15) || No opponent in line-of-sight

BR( **CHECK\_RIGHT\_VALID\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_RIGHT\_VALID\_INNERLOOP1:**

SHRC(R1, 1, R1) || Shift current selection mask to right by 1

AND(R1, R7, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for right bound (No right-directed own piece)

BEQ(R0, **CHECK\_RIGHT\_DOWN\_VALID**, R15)

|| Check if this piece is opponent’s piece

AND(R1, R12, R0) || Masking current pointer with opponent’s grid

|| if is opponent’s piece then SHR to next bit

BNE(R0, **CHECK\_RIGHT\_VALID\_INNERLOOP1**, R15)

|| Check if this piece is own piece

AND(R1, R13, R0) || Masking current pointer with own grid

|| If this piece is not own then we branch (Empty space)

BEQ(R0, **CHECK\_RIGHT\_DOWN\_VALID**, R15)

BR( **MARK\_VALID** )

**CHECK\_RIGHT\_DOWN\_VALID:**

SHRC(R6, 1, R1) || Shift current selection mask to right by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row

|| Check if we are out of bounds for right bound

BEQ(R0, **CHECK\_DOWN\_VALID**, R15)

SHRC(R1, 8, R1) || Shift current selection mask to right by 8 (Initialise Pointer)

SHRC(R7, 8, R2) || Shift current row selection mask to right (Initialise Pointer)

|| R2 → Pointer for row selection, R1 → Pointer for current selection

BEQ(R1, **CHECK\_DOWN\_VALID**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **CHECK\_DOWN\_VALID**, R15) || No opponent in line-of-sight

BR( **CHECK\_RIGHT\_DOWN\_VALID\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_RIGHT\_DOWN\_VALID\_INNERLOOP1:**

SHRC(R1, 1, R1) || Shift current selection mask to right by 1

AND(R1, R2, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for right bound (No right-directed own piece)

BEQ(R0, **CHECK\_DOWN\_VALID**, R15)

SHRC(R1, 8, R1) || Shift current selection mask to right by 8 (next row)

SHRC(R2, 8, R2) || Shift current row selection mask to right by 8 (next row)

BEQ(R1, **CHECK\_DOWN\_VALID**, R15) || Check if Out-Of-Bounds (Up-Down)

|| Check if this piece is opponent’s piece

AND(R1, R12, R0) || Masking current pointer with opponent grid

|| If opponent’s piece, then get next pcs

BNE(R0, **CHECK\_RIGHT\_DOWN\_VALID\_INNERLOOP1**, R15)

|| Check if this piece is own piece

AND(R1, R13, R0) || Masking current pointer with own grid

|| If this piece is not own then we branch (Empty space)

BEQ(R0, **CHECK\_DOWN\_VALID**, R15)

BR( **MARK\_VALID** )

**CHECK\_DOWN\_VALID:**

SHRC(R6, 8, R1) || Shift current selection mask to right by 8 (Initialise Pointer)

BEQ(R1, **CHECK\_LEFT\_DOWN\_VALID**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **CHECK\_LEFT\_DOWN\_VALID**, R15) || No opponent in line-of-sight

BR( **CHECK\_DOWN\_VALID\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_DOWN\_VALID\_INNERLOOP1:**

SHRC(R1, 8, R1) || Shift current selection mask to right by 8 (next row)

BEQ(R1, **CHECK\_LEFT\_DOWN\_VALID**, R15) || Check if Out-Of-Bounds

|| Check if this piece is opponent’s piece

AND(R1, R12, R0) || Masking current pointer with opponent’s grid

|| if is opponent’s piece then move down 1 row

BNE(R0, **CHECK\_DOWN\_VALID\_INNERLOOP1**, R15)

|| Check if this piece is own piece

AND(R1, R13, R0) || Masking current pointer with own grid

|| If this piece is not own then we branch (Empty space)

BEQ(R0, **CHECK\_LEFT\_DOWN\_VALID**, R15)

BR( **MARK\_VALID** )

**CHECK\_LEFT\_DOWN\_VALID:**

SHLC(R6, 1, R1) || Shift current selection mask to left by 1 (Initialise Pointer)

AND(R1, R7, R0) || Mask the current selected bit with the selected row

|| Check if we are out of bounds for left bound return control to main program

BEQ(R0, **GVG\_BIT\_SHIFT**, R15)

SHRC(R1, 8, R1) || Shift current selection mask to right by 8 (Initialise Pointer)

SHRC(R7, 8, R2) || Shift current row selection mask to right (Initialise Pointer)

|| R2 → Pointer for row selection, R1 → Pointer for current selection

BEQ(R1, **GVG\_BIT\_SHIFT**, R15) || If 0 means that we are out-of-bounds

|| Need to have at least 1 opponent in line-of-sight

AND(R1, R12, R0) || Check if we have 1 opponent (P2) in line-of-sight

BEQ(R0, **GVG\_BIT\_SHIFT**, R15) || No opponent in line-of-sight

BR( **CHECK\_LEFT\_DOWN\_VALID\_INNERLOOP1** )

|| Here onwards, we have at least 1 opponent, now find Own piece

**CHECK\_LEFT\_DOWN\_VALID\_INNERLOOP1:**

SHLC(R1, 1, R1) || Shift current selection mask to left by 1

AND(R1, R2, R0) || Mask current selected bit with selected row

|| Check if we are out of bounds for left bound (No right-directed own piece)

BEQ(R0, **GVG\_BIT\_SHIFT**, R15)

SHRC(R1, 8, R1) || Shift current selection mask to right by 8 (next row)

SHRC(R2, 8, R2) || Shift current row selection mask to right by 8 (next row)

BEQ(R1, **GVG\_BIT\_SHIFT**, R15) || Check if Out-Of-Bounds (Up-Down)

|| Check if this piece is opponent’s piece

AND(R1, R12, R0) || Masking current pointer with opponent’s grid

|| If opponent’s piece, then get next pcs

BNE(R0, **CHECK\_LEFT\_DOWN\_VALID\_INNERLOOP1**, R15)

|| Check if this piece is own piece

AND(R1, R13, R0) || Masking current pointer with own grid

|| If this piece is not own then we branch back to main program (Empty space)

BEQ(R0, **GVG\_BIT\_SHIFT**, R15)

BR( **MARK\_VALID** )

**MARK\_VALID:**

|| Purpose: This function is used to mark current selected bit to be valid in the valid grid

OR( R9, R6, R9 ) || Make current empty space as valid space

BR( **GVG\_BIT\_SHIFT** ) || Branch back to the get next bit of grid to check