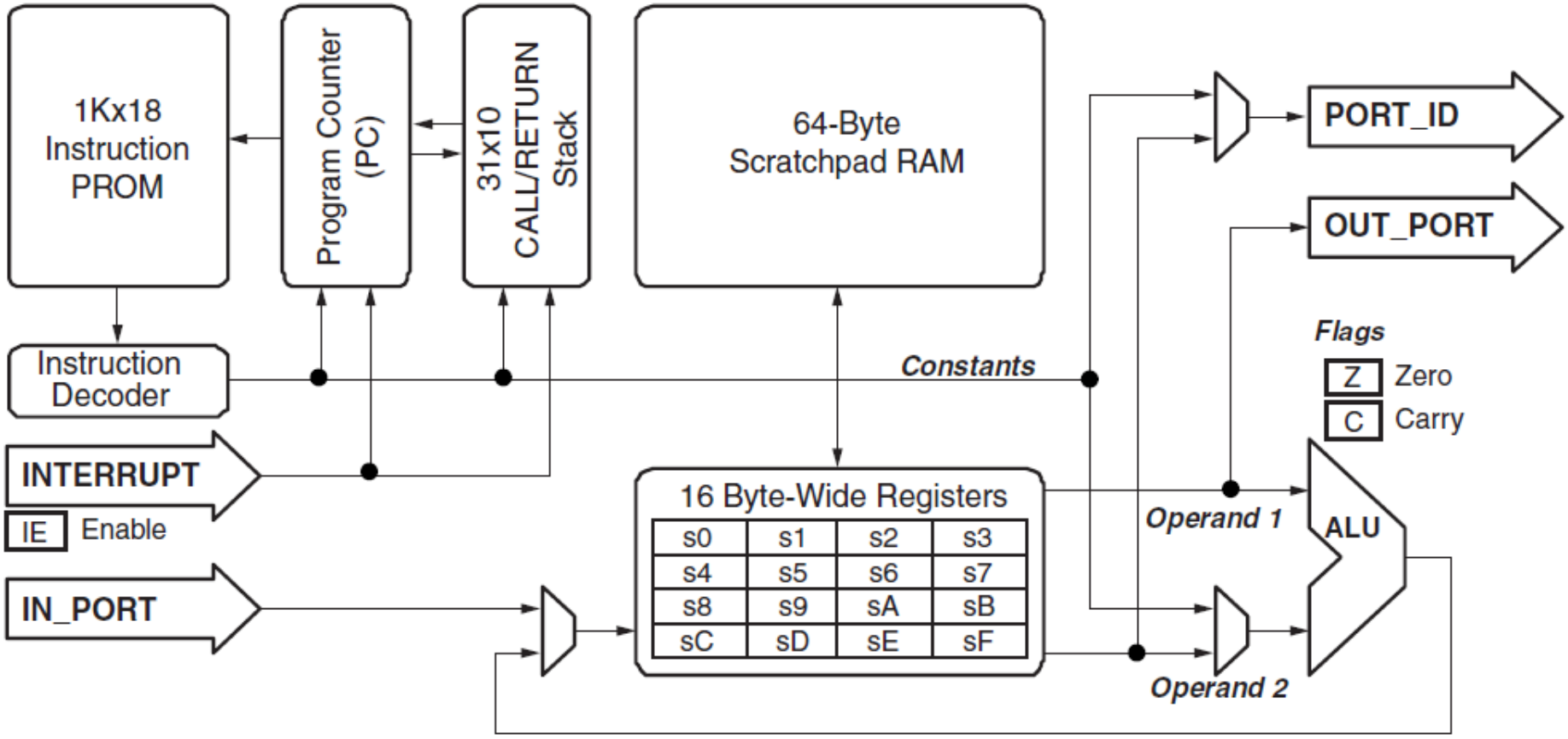
PicoBlaze 8-bit Embedded Microcontroller

Source: “PicoBlaze 8-bit Embedded Microcontroller User Guide *for Spartan-3, Spartan-6, Virtex-5,and Virtex-6 FPGAs,” UG129 (v2.0), Jan 28, 2010.*

PicoBlaze Overview

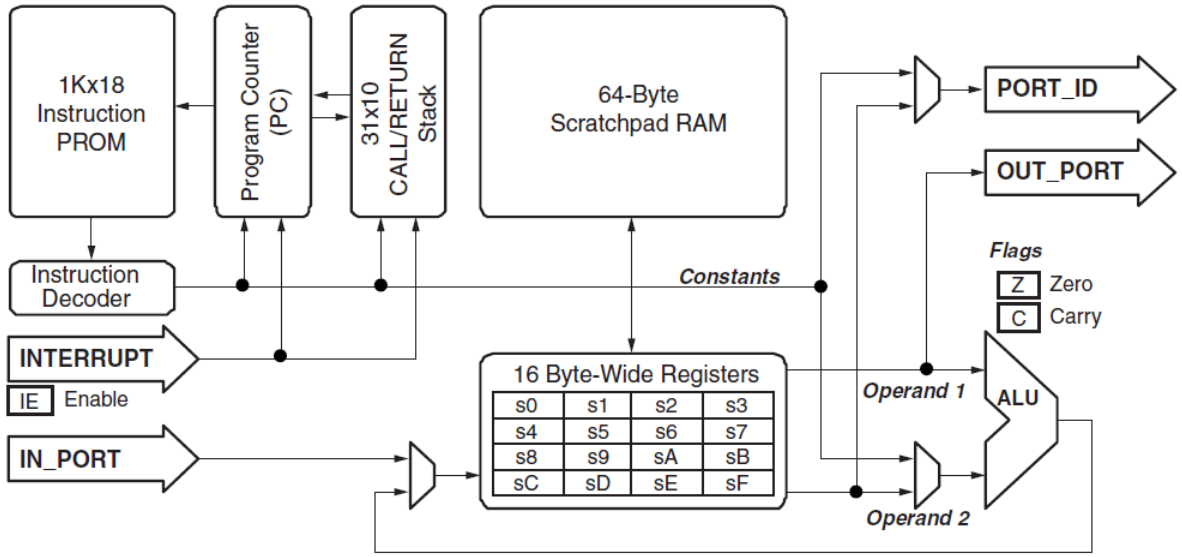
- PicoBlaze is a simple 8-bit microcontroller
- Compact in size
  - Occupies 96 slices in a Spartan-3 (5% of an XC3S200)
- Performance is respectable to approximately 43 to 66 MIPs depending upon device and speed grade
- Over 49 instructions
  - All instructions under all conditions will execute over 2 clock cycles
- Supports a program up to a length of 1024 instructions using one Block RAM

PicoBlaze Architecture



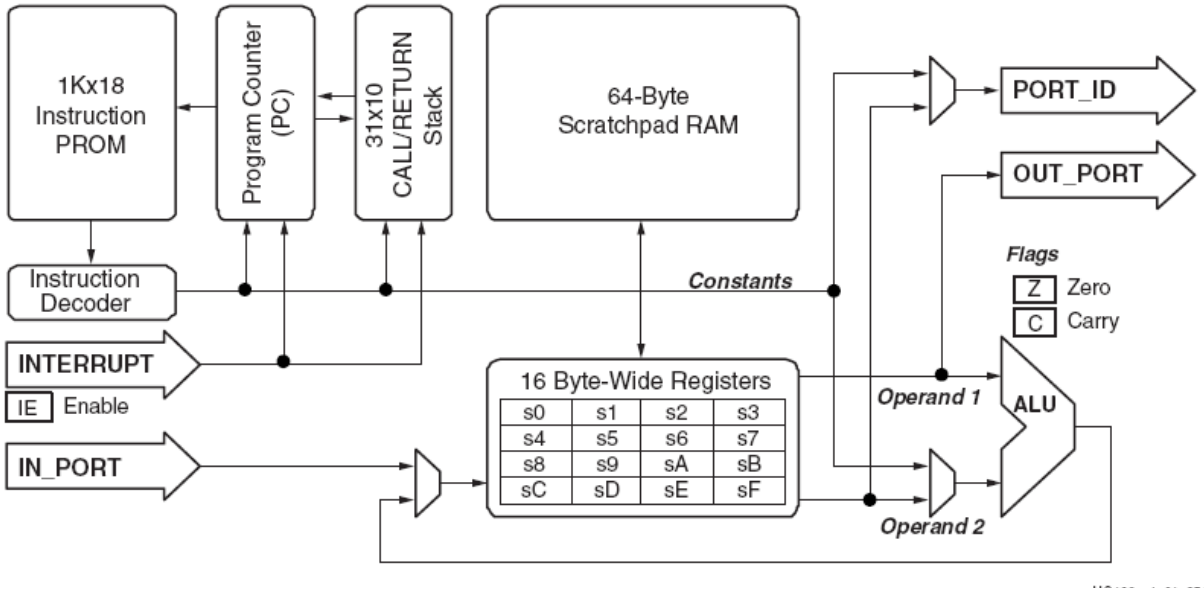
PicoBlaze Architecture

- **General-Purpose Registers**
  - 16 8-bit regs: s0 – sF
  - No register reserved for special task
  - No dedicated accumulator
- **1,024-Instruction Program Store**
  - Up to 1,024 (1K) instructions
  - Instruction is 18 bits wide
- **Arithmetic Logic Unit (ALU)**
  - Basic arithmetic operations
  - Bitwise logic operations
  - Arithmetic compare and bitwise test operations
  - Shift and rotate operations
- **Flags**
  - ZERO, CARRY, INTERRUPT_ENABLE
  - ALU operations affect the ZERO & CARRY Flags



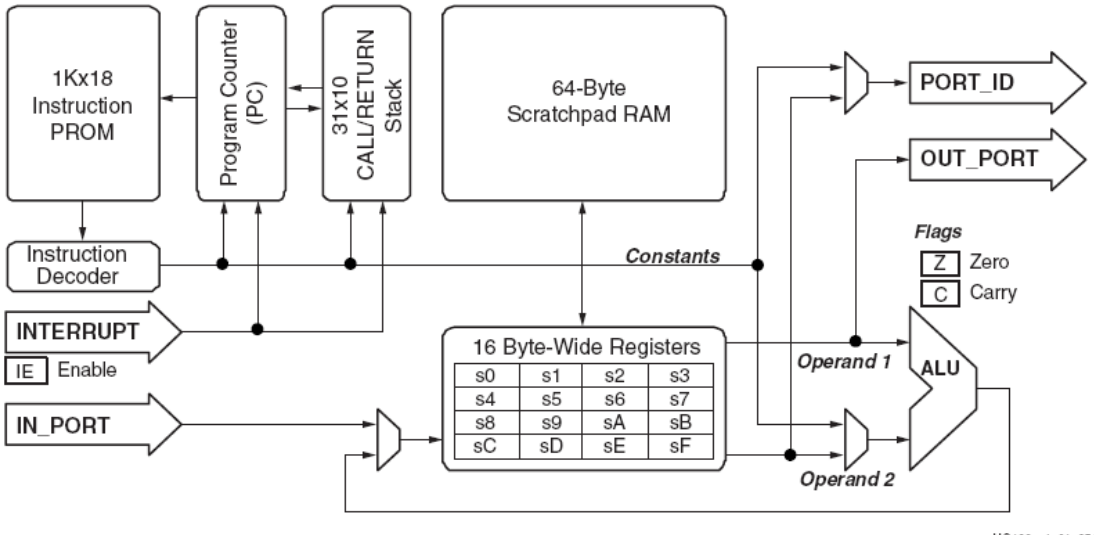
PicoBlaze Architecture

- **64-Byte Scratchpad RAM**
  - Directly or indirectly addressable from the register file
  - STORE/FETCH instructions
- **Input/Output**
  - Can connect custom peripheral
  - Upto 256 input ports
  - Upto 256 output ports
  - Or a mix of input/ouputs
- **Program Counter (PC)**
  - JUMP, CALL, RETURN, and RETURNI, Interrupt, and Reset modify default operation
  - Maximum Code space of 1,024 instructions (000 – 3FF hex)
  - Rolls over if 3FF hex is reached



PicoBlaze Architecture

- **Program Flow Control**
  - Conditional and non-conditional program flow
  - JUMP absolute address
  - CALL start address is an absolute address
  - Return address is automatically stored on STACK
- **CALL/RETURN Stack**
  - Up to 31 instn. addresses
  - Stack is used during interrupt
  - Implemented as Cyclic buffer
  - No explicit Stack instructions
  - No program memory required for Stack
- **Interrupts**
  - INTERRUPT input
  - Handle async. External inputs
  - Response – 5 clock cycles
- **Reset**
  - PC=0; Flags Cleared
  - Interrupts disabled
  - CALL/RETURN Stack reset



Port Interface

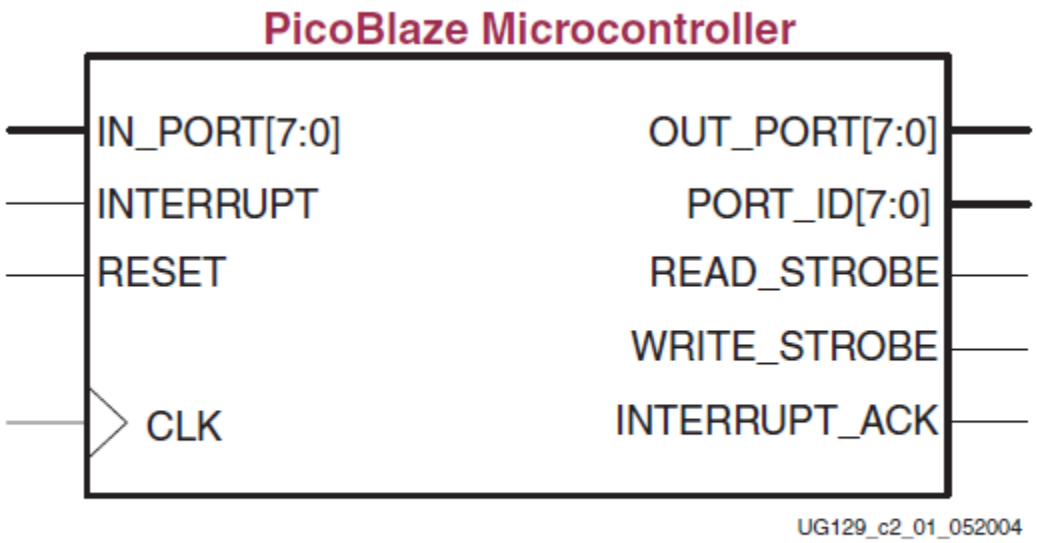


Figure 2-1: **PicoBlaze Interface Connections**

Input Ports

Table 2-1: **PicoBlaze Interface Signal Descriptions**

Signal	Direction	Description
IN_PORT[7:0]	Input	**Input Data Port:** Present valid input data on this port during an INPUT instruction. The data is captured on the rising edge of CLK.
INTERRUPT	Input	**Interrupt Input:** If the INTERRUPT_ENABLE flag is set by the application code, generate an INTERRUPT Event by asserting this input High for at least two CLK cycles. If the INTERRUPT_ENABLE flag is cleared, this input is ignored.
RESET	Input	**Reset Input:** To reset the PicoBlaze microcontroller and to generate a RESET Event, assert this input High for at least one CLK cycle. A Reset Event is automatically generated immediately following FPGA configuration.
CLK	Input	**Clock Input:** The frequency may range from DC to the maximum operating frequency reported by the Xilinx ISE® development software. All PicoBlaze synchronous elements are clocked from the rising clock edge. There are no clock duty-cycle requirements beyond the minimum pulse width requirements of the FPGA.
OUT_PORT[7:0]	Output	**Output Data Port:** Output data appears on this port for two CLK cycles during an OUTPUT instruction. Capture output data within the FPGA at the rising CLK edge when WRITE_STROBE is High.
PORT_ID[7:0]	Output	**Port Address:** The I/O port address appears on this port for two CLK cycles during an INPUT or OUTPUT instruction.

Output Ports

Table 2-1: **PicoBlaze Interface Signal Descriptions *(Cont'd)***

Signal	Direction	Description
READ_STROBE	Output	**Read Strobe:** When asserted High, this signal indicates that input data on the IN_PORT[7:0] port was captured to the specified data register during an INPUT instruction. This signal is asserted on the second CLK cycle of the two-cycle INPUT instruction. This signal is typically used to acknowledge read operations from FIFOs.
WRITE_STROBE	Output	**Write Strobe:** When asserted High, this signal validates the output data on the OUT_PORT[7:0] port during an OUTPUT instruction. This signal is asserted on the second CLK cycle of the two-cycle OUTPUT instruction. Capture output data within the FPGA on the rising CLK edge when WRITE_STROBE is High.
INTERRUPT_ACK	Output	**Interrupt Acknowledge:** When asserted High, this signal acknowledges that an INTERRUPT Event occurred. This signal is asserted during the second CLK cycle of the two-cycle INTERRUPT Event. This signal is optionally used to clear the source of the INTERRUPT input.

Logic Instructions -- Examples

```
toggle_bit:
;   XOR sX, <bit_mask>

    XOR s0, 01  ; toggle the least-significant bit in register sX
```

Figure 3-3: **Inverting an Individual Bit Location**

```
    XOR sX, sX  ; clear register sX, set ZERO flag
```

Figure 3-4: **Clearing a Register and Setting the ZERO Flag**

```
    LOAD sX,00  ; clear register sX, ZERO flag unaffected
```

Figure 3-5: **Clearing a Register without Modifying the ZERO Flag**

```
set_bit:
;   OR sX, <bit_mask>

    OR s0, 01 ; set bit 0 of register s0
```

Figure 3-6: **16-Setting a Bit Location**

Arithmetic Instructions - Examples

```
ADD16:
        NAMEREG s0, a_lsb   ; rename register s0 as "a_lsb"
        NAMEREG s1, a_msb   ; rename register s1 as "a_msb"
        NAMEREG s2, b_lsb   ; rename register s2 as "b_lsb"
        NAMEREG s3, b_msb   ; rename register s3 as "b_lsb"

        ADD   a_lsb, b_lsb  ; add LSBs, keep result in a_lsb
        ADDCY a_msb, b_msb  ; add MSBs, keep result in a_msb
        RETURN
```

Figure 3-8: **16-Bit Addition Using ADD and ADDCY Instructions**

```
    ADD sX,01 ; increment register sX
    SUB sX,01 ; decrement register sX
```

Figure 3-10: **Incrementing and Decrementing a Register**

```
Negate:
    ; invert all bits in the register performing a one's complement
    XOR sX,FF
    ; add one to sX
    ADD sX,01
    RETURN
```

Figure 3-12: **Destructive Negate (2's Complement) Function Overwrites Original Value**

NOP, CARRY set/clear

```
nop:
    LOAD sX, sX
```

Figure 3-18: **Loading a Register with Itself Acts as a NOP Instruction**

```
clear_carry_bit:
    AND sX, sX  ; register sX unaffected, CARRY flag cleared
```

Figure 3-20: **ANDing a Register with Itself Clears the CARRY Flag**

```
set_carry:
    LOAD sX, 00
    COMPARE sX, 01 ; set CARRY flag and reset ZERO flag
```

Figure 3-21: **Example Operation that Sets the CARRY Flag**

Test Instruction

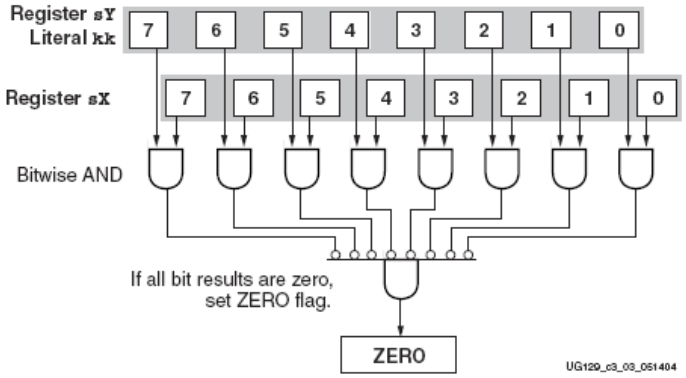


Figure 3-22: **The TEST Instruction Affects the ZERO Flag**

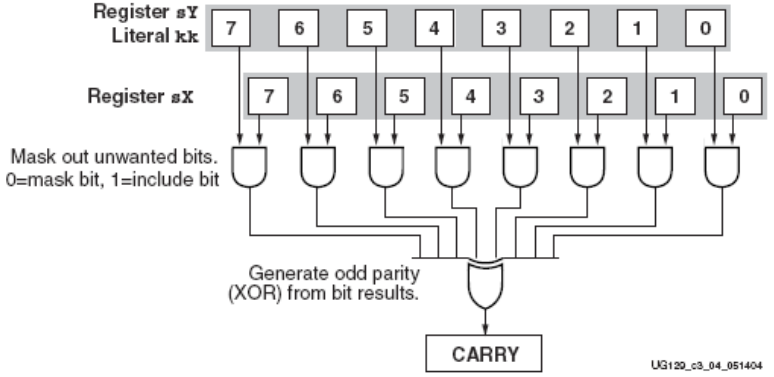


Figure 3-24: **The TEST Instruction Affects the CARRY Flag**

NOTE: Only affects ZERO ad CARRY flags; does not affect register contents

```
LOAD s0, 05 ;    s0 = 00000101
TEST s0, 04 ;  mask = 00000100
            ; CARRY = 1, ZERO = 0
```

Figure 3-23: **Generate Parity for a Register Using the TEST Instruction**

```
generate_parity:
   TEST sX, FF   ; include all bits in parity generation
```

Figure 3-25: **Generate Parity for a Register Using the TEST Instruction**

Compare Instruction

Table 3-3: **COMPARE Instruction Flag Operations**

Flag	When Flag=0	When Flag=1
ZERO	Operand_1 ≠ Operand_2	Operand_1 = Operand_2
CARRY	Operand_1 ≥ Operand_2	Operand_1 < Operand_2

Shift & Rotate Instructions

Table 3-4: **PicoBlaze Shift Instructions**

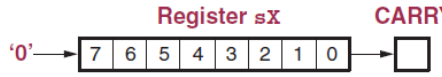
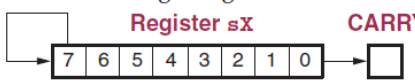
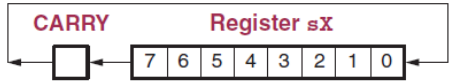
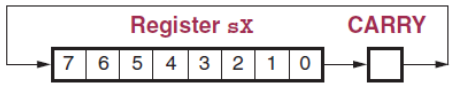
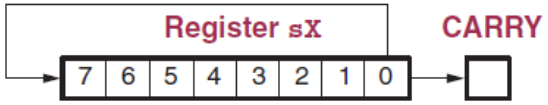
	Shift Left		Shift Right
SL0	Shift Left with '0' fill. 	SR0	Shift Right with '0' fill. 
SL1	Shift Left with '1' fill. 	SR1	Shift Right with '1' fill. 
SLX	Shift Left, eXtend bit 0. 	SRX	Shift Right, sign eXtend. 
SLA	Shift Left through All bits, including CARRY. 	SRA	Shift Right through All bits, including CARRY. 

Table 3-5: **PicoBlaze Rotate Instructions**

	Rotate Left		Rotate Right
RL		RR	

Moving Data

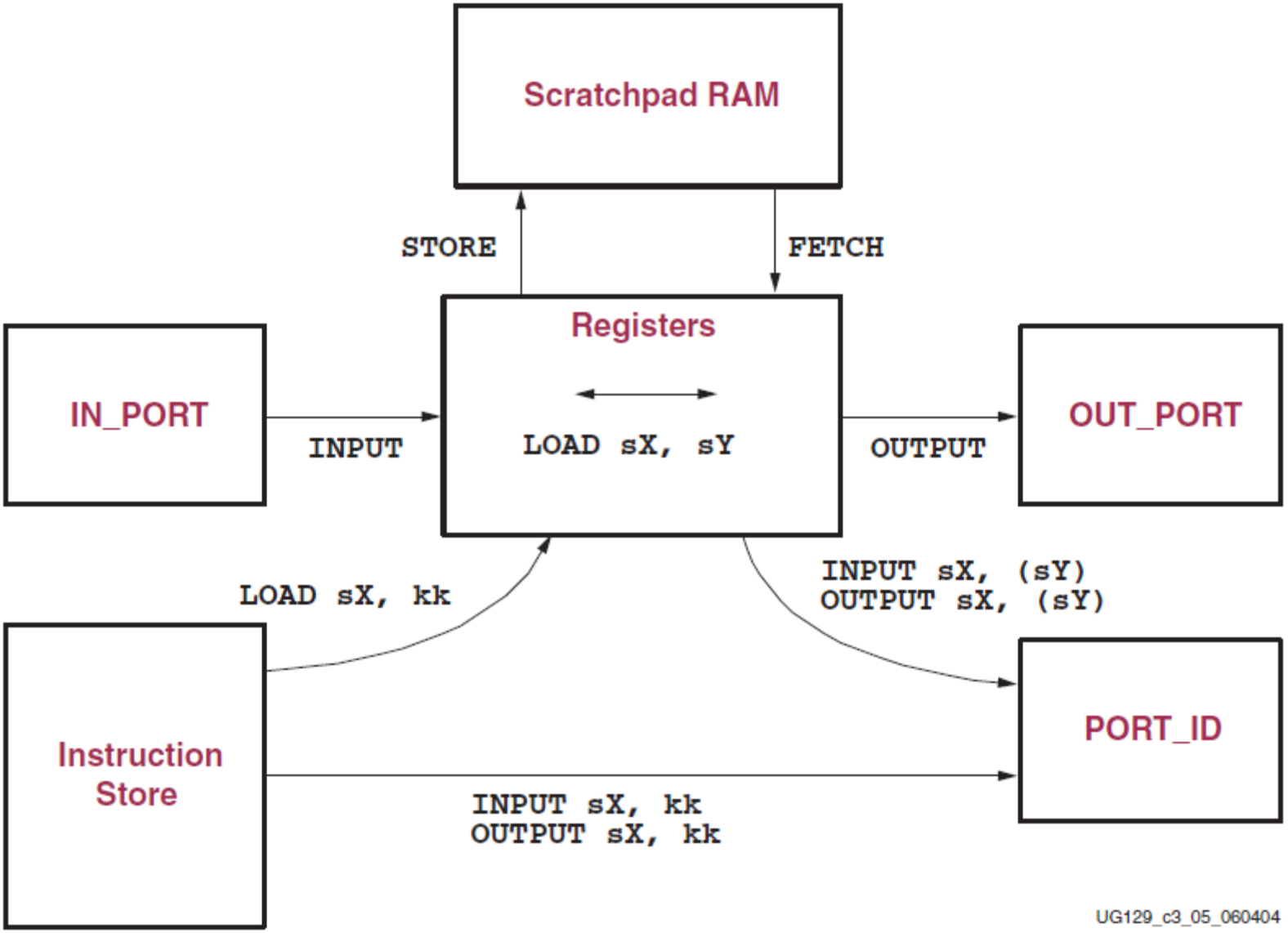
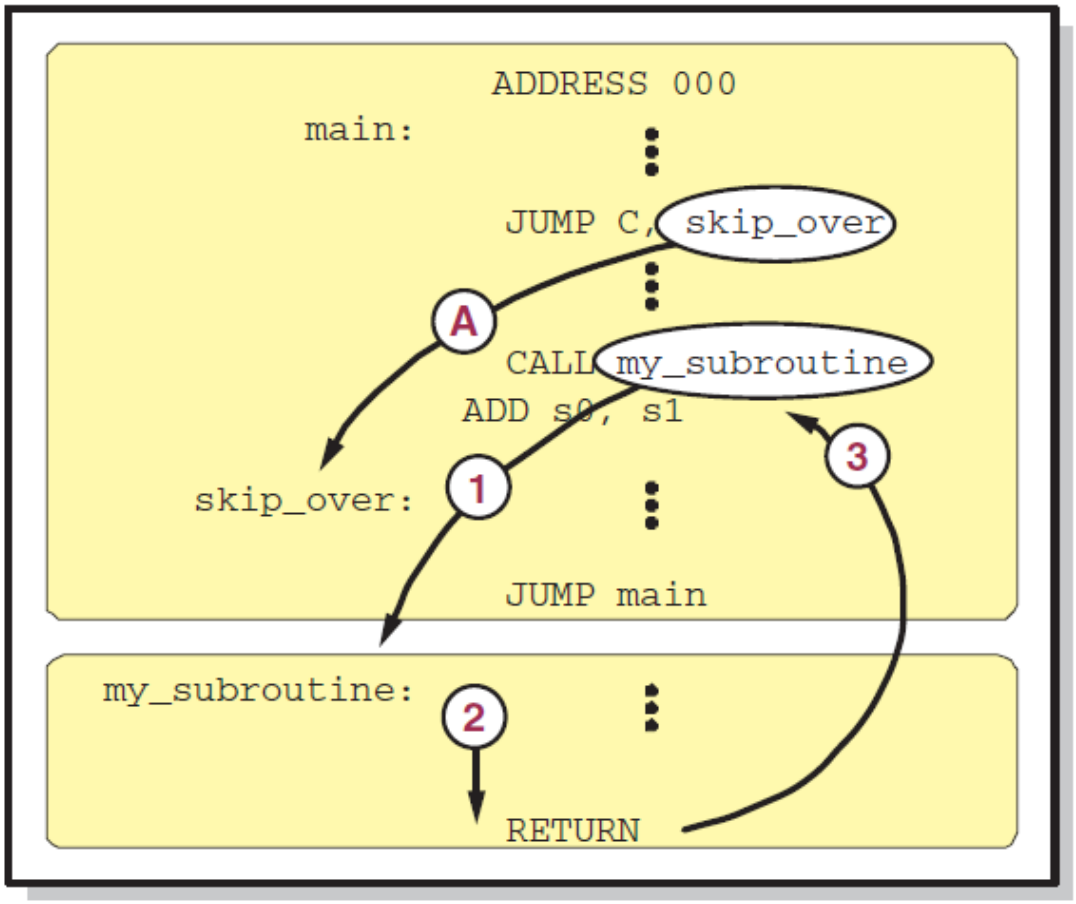


Figure 3-26: **Data Movement Instructions**

Program Flow



Jump Instn:

- does not affect CALL/RETURN Stack
- does not affect CARRY and ZERO
- all jumps are absolute
  i.e., no relative jumps

Call Instn:

- if condn is false, then no effect except wastage of 2 clock cycles
- if condn is true, then
  - push **current** PC on stack
  - load PC with label
  - executes until RETURN

RETURN:

- pop the stack
- increment the value
- load into PC.

Instruction Conditional Execution

Table 3-6: **Instruction Conditional Execution**

Condition	Description
<none>	Always true. Execute instruction unconditionally.
C	CARRY = 1. Execute instruction if CARRY flag is set.
NC	CARRY = 0. Execute instruction if CARRY flag is cleared.
Z	ZERO = 1. Execute instruction if ZERO flag is set.
NZ	ZERO = 0. Execute instruction if ZERO flag is cleared.

Interrupts

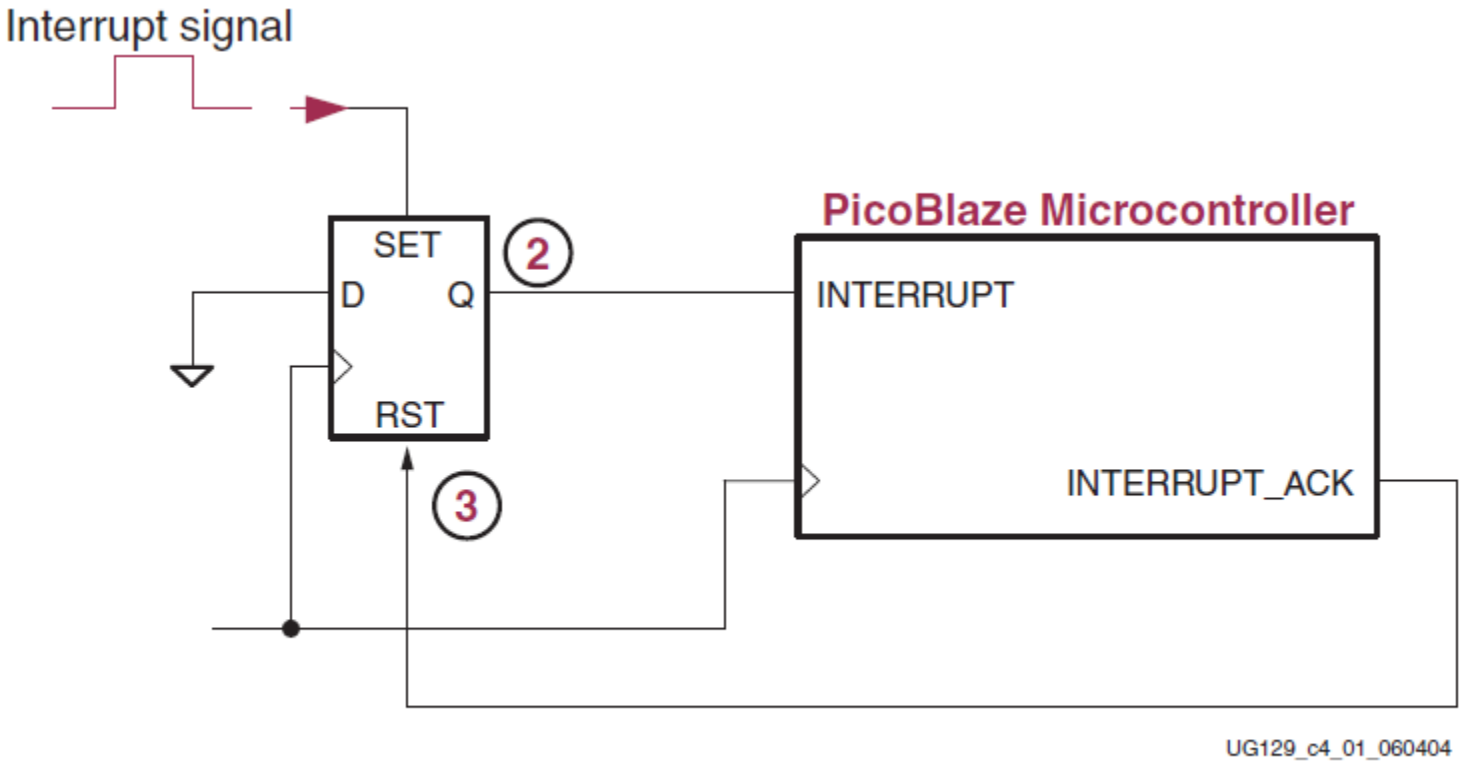
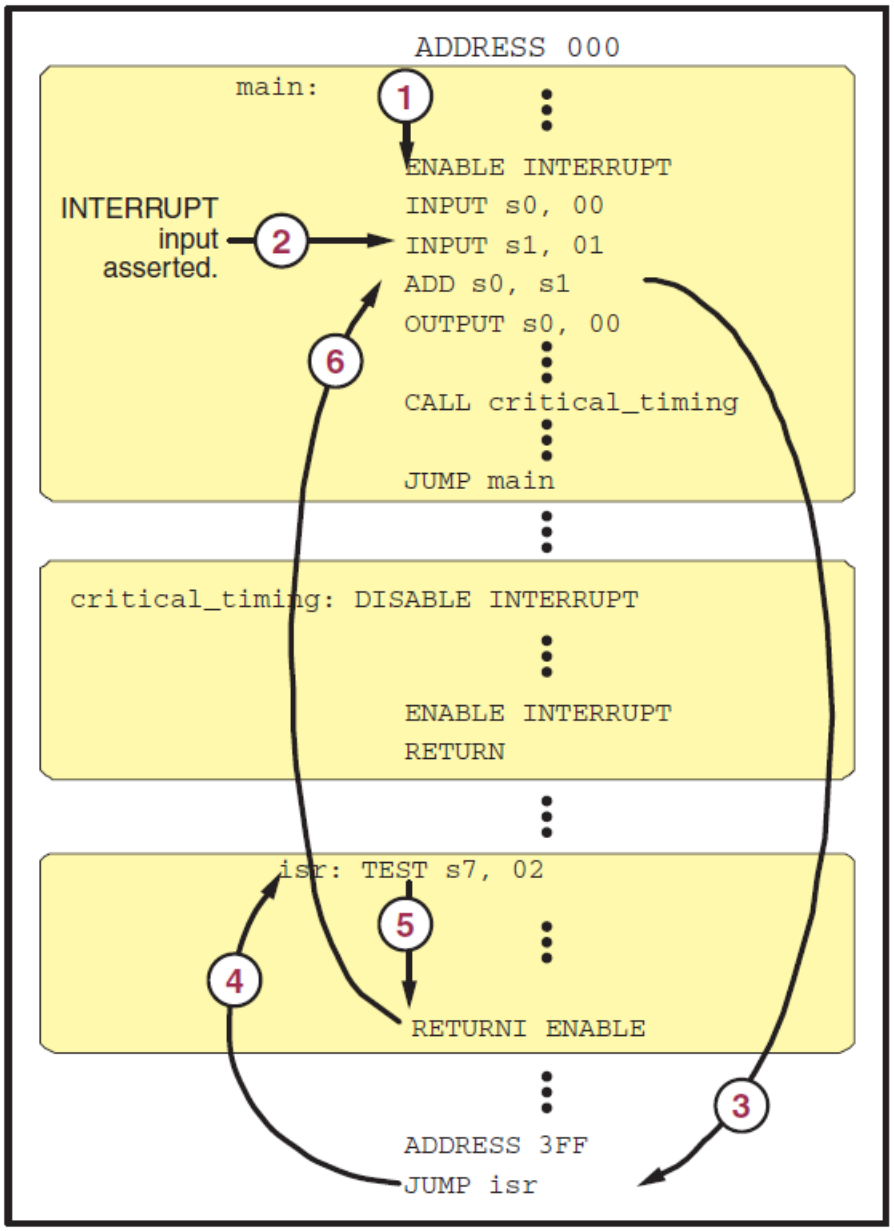


Figure 4-1: **Simple Interrupt Logic**

Interrupt signal must be applied atleast 2 clock cycles

Interrupt Flow - Example



The interrupt input is not recognized until the INTERRUPT_ENABLE flag is set.

In timing-critical functions or areas where absolute predictability is required, temporarily disable the interrupt. Re-enable the interrupt input when the time-critical function is complete.

Always return from a sub-routine call with the `RETURN` instruction.

The interrupt input is automatically disabled.

Use the `RETURNI` instruction to return from an interrupt.

The interrupt vector is always located at the most-significant memory location, where all the address bits are ones. Jump to the interrupt service routine.

UG129_c4_02_051404

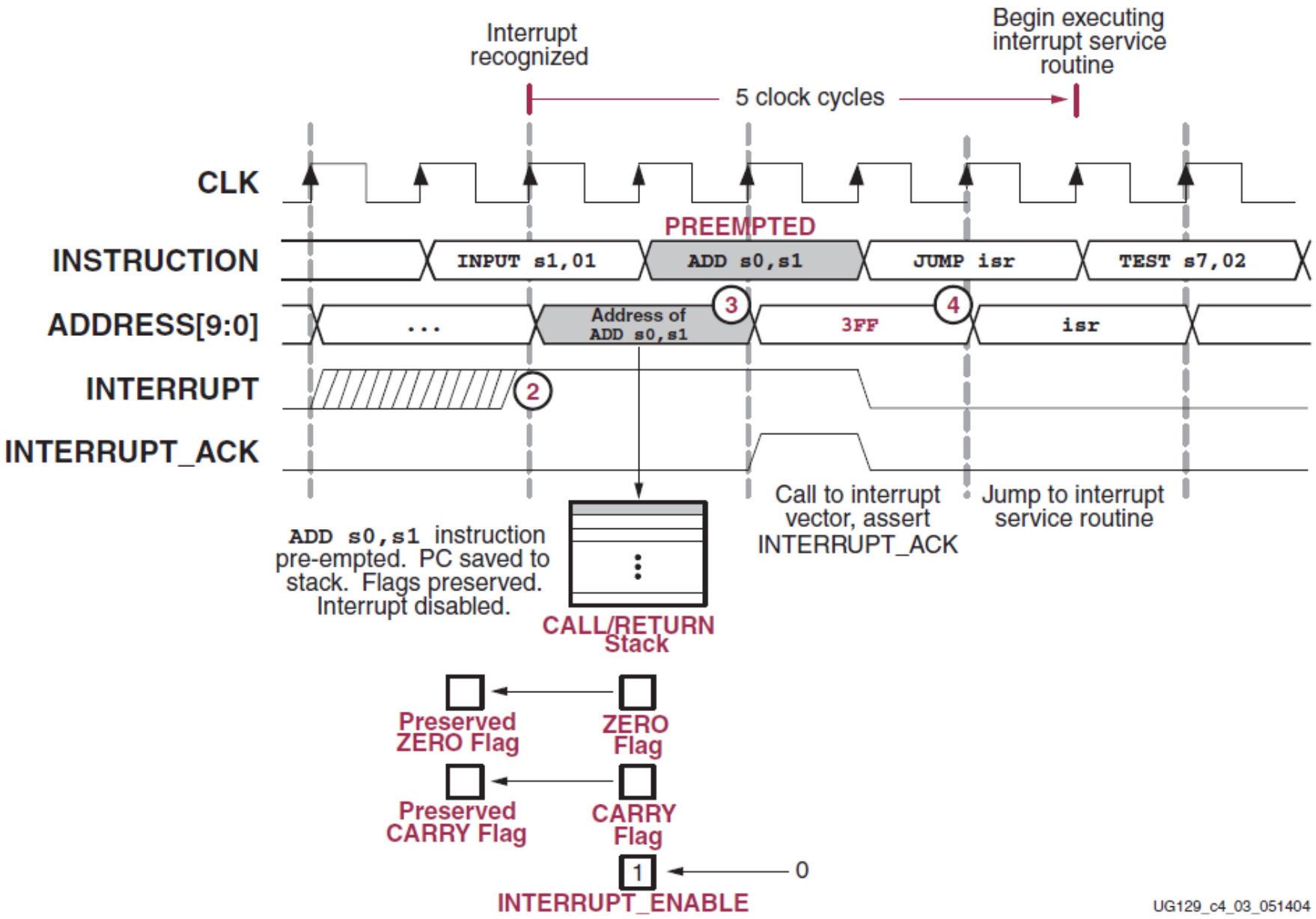


Figure 4-3: **Interrupt Timing Diagram**

Scratchpad RAM

Direct Addressing:

```
scratchpad_transfers:
    STORE sX, 04  ; Write register sX to RAM location 04
    FETCH sX, 04  ; Read RAM location 04 into register sX
```

Figure 5-1: **Directly Addressing Scratchpad RAM Locations**

Indirect Addressing:

```
            NAMEREG s0, ram_data
            NAMEREG s1, ram_address

            CONSTANT ram_locations, 40      ; there are 64 locations
            CONSTANT initial_value, 00      ; initialize to zero

            LOAD ram_data, initial_value    ; load initial value
            LOAD ram_address, ram_locations ; fill from top to bottom

ram_fill:   SUB ram_address, 01             ; decrement address
            STORE ram_data, (ram_address)   ; initialize location
            JUMP NZ, ram_fill               ; if not address 0, goto
                                            ; ram_fill
```

Figure 5-2: **Indirect Addressing Initializes All of RAM with a Simple Subroutine**

Input Port

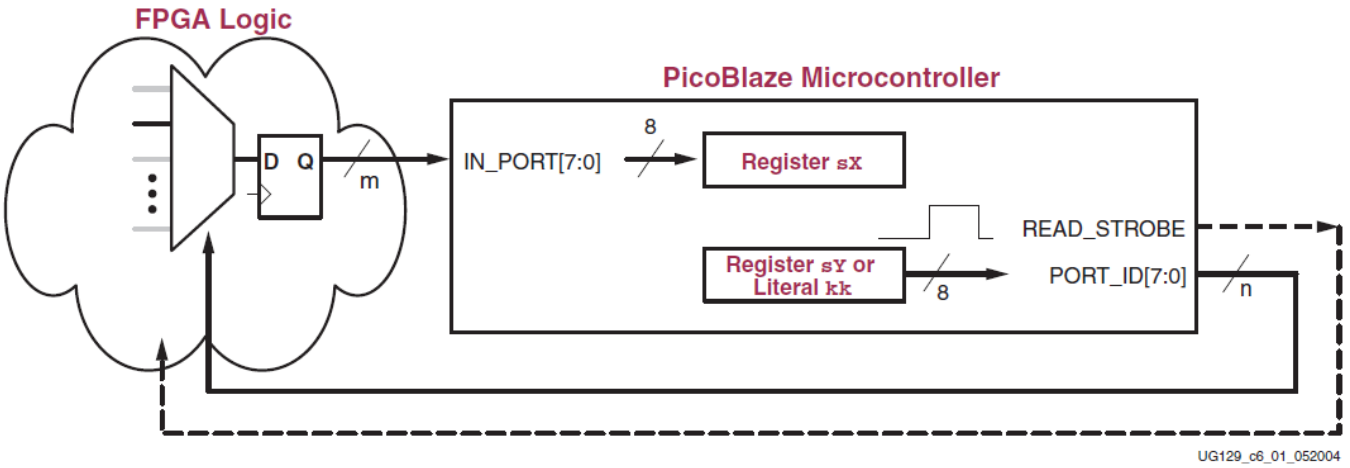


Figure 6-1: **INPUT Operation and FPGA Interface Logic**

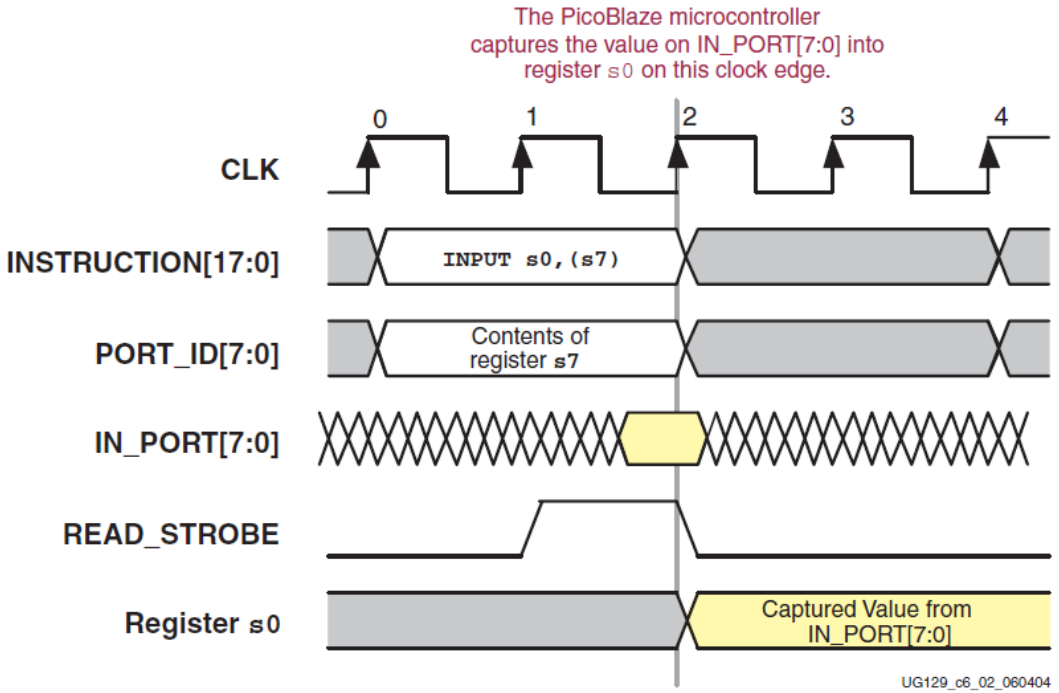


Figure 6-2: **Port Timing for INPUT Instruction**

Multiple Input Ports - Example

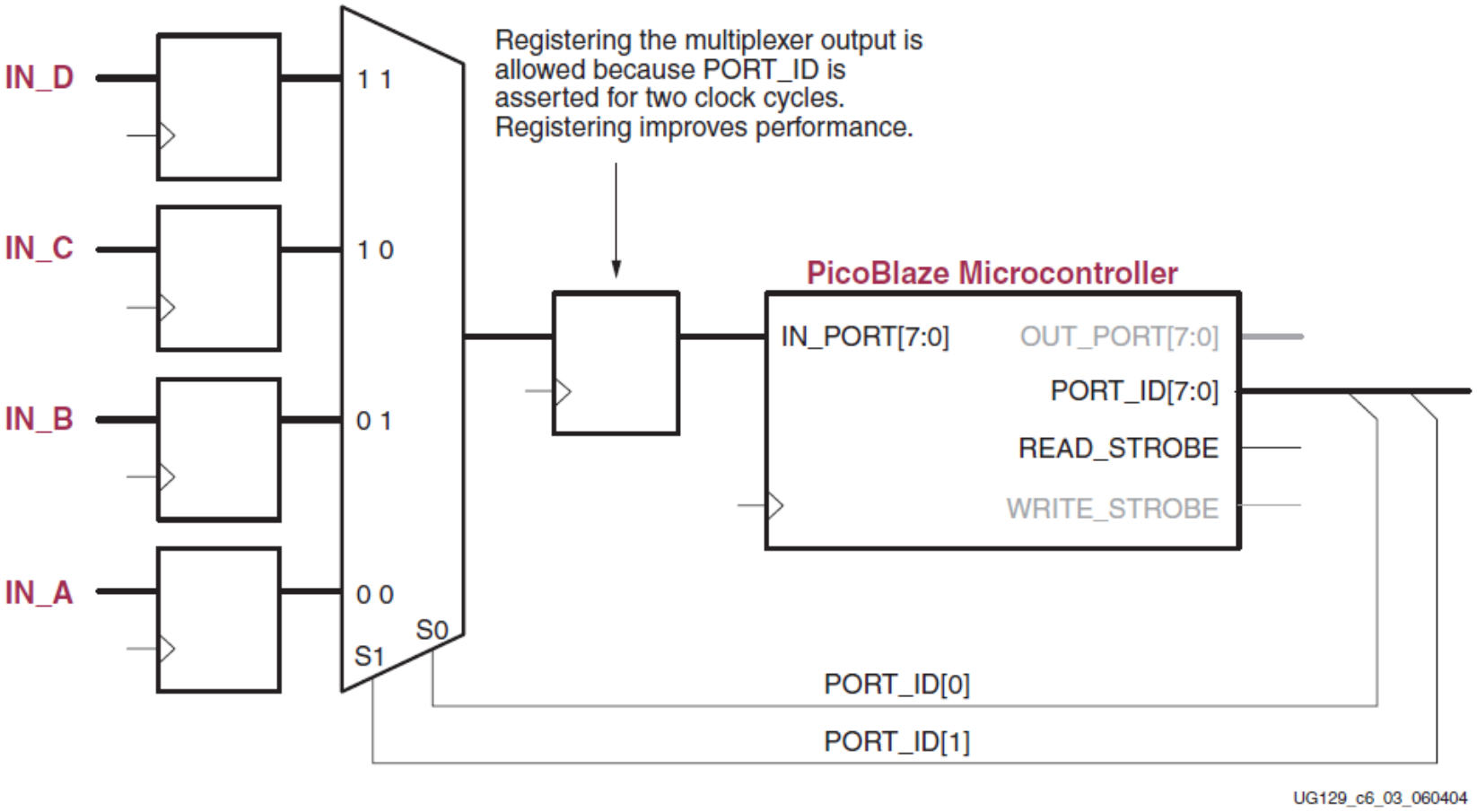


Figure 6-3: **Multiplex Multiple Input Sources to Form a Single IN_PORT Port**

Output Port

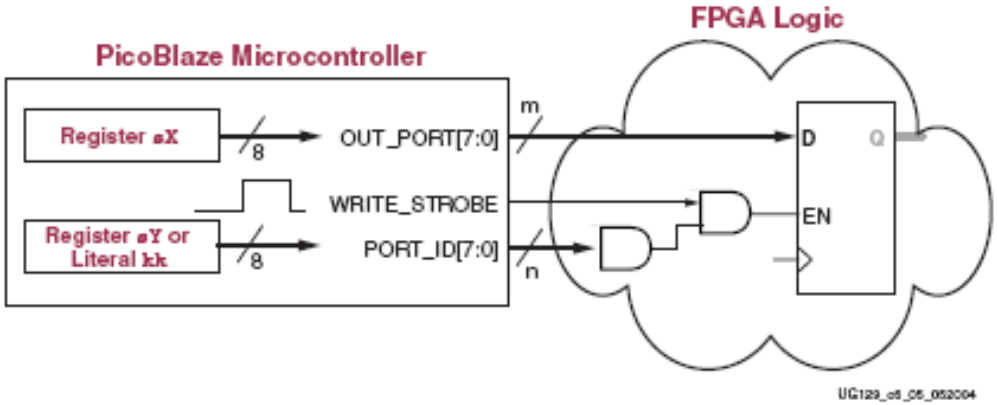


Figure 6-5: **OUTPUT Operation and FPGA Interface**

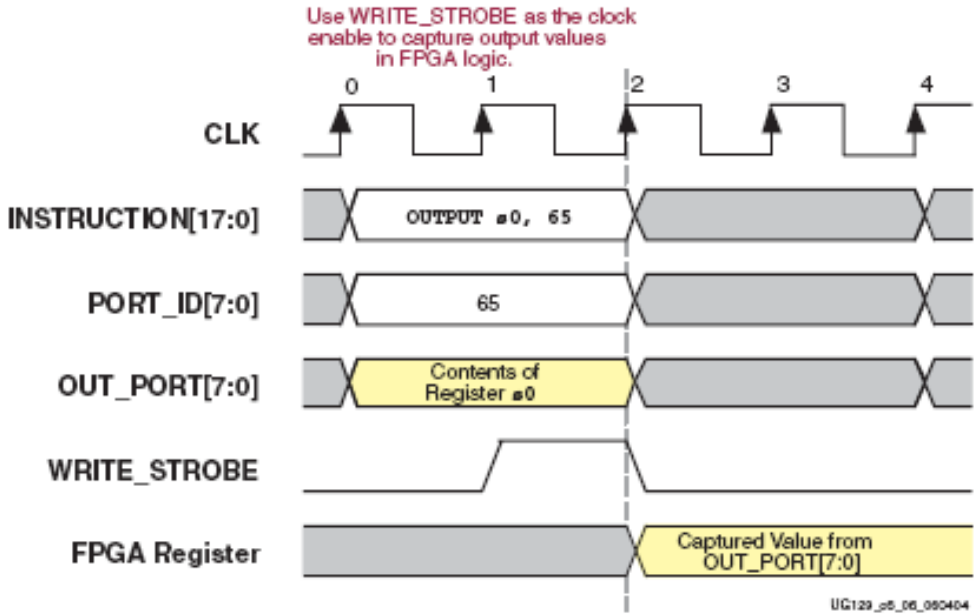


Figure 6-6: **Port Timing for OUTPUT Instruction**

Multiple Outputs (≤8)

NOTE: One hot encoding of outputs

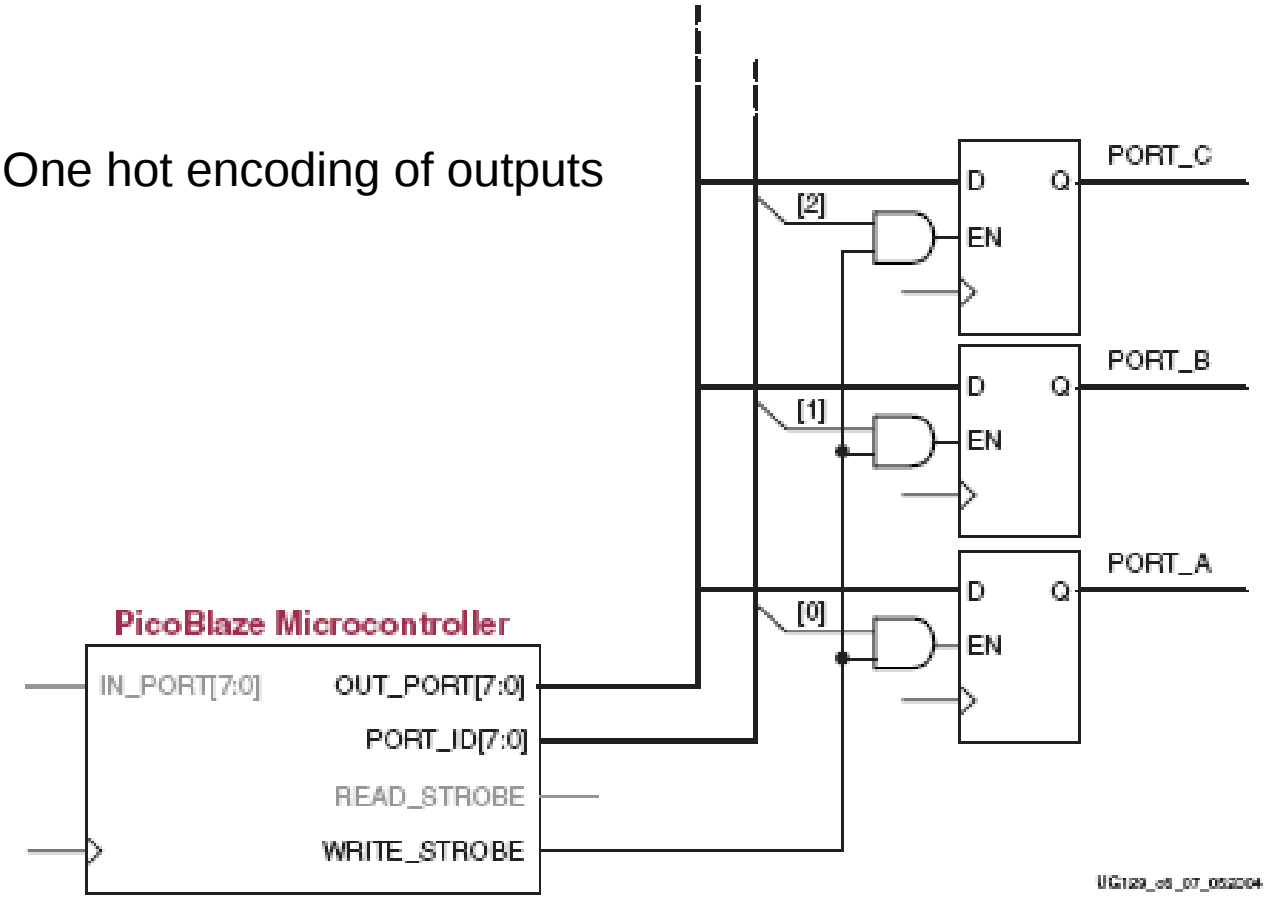


Figure 6-7: **Simple Address Decoding for Designs with Few Output Destinations**

PicoBlaze Assembler

Available online at http://zombie.narmos.org/~cwbell/picoasm/

Upload a PSM assembly file, and the application will generate the Verilog ROM for you to include in the project.

PBlazIDE – PicoBlaze Instruction Set Simulator

- Free, graphical, integrated development environment for Windows
- Syntax color highlighting
- Instruction set simulator (ISS)
  - ♦ Breakpoints
  - ♦ Register display
  - ♦ Memory display
- Source code formatter (“pretty print”)
- KCPSM3-to-pBlazIDE import function/syntax conversion
- HTML output, including color highlighting

Download the pBlazIDE software:

http://www.mediatronix.com/pBlazeIDE.htm

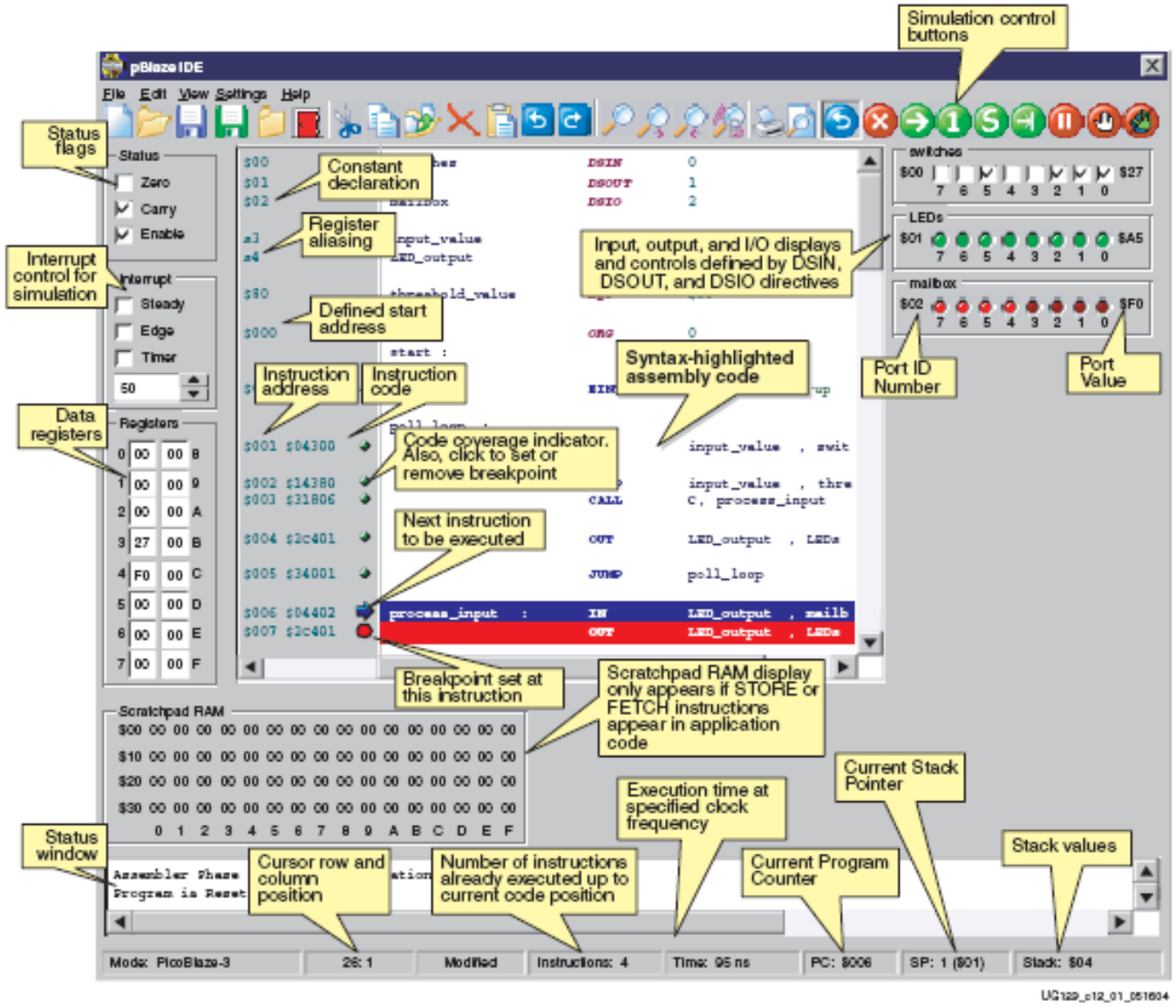


Figure 12-1: **The pBlazIDE Instruction Set Simulator (ISS)**

Table 10-2: **Instruction Mnemonic Differences between KCPSM3 and pBlazIDE**

KCPSM3 Instruction	pBlazIDE Instruction
`RETURN`	`RET`
`RETURN C`	`RET C`
`RETURN NC`	`RET NC`
`RETURN Z`	`RET Z`
`RETURN NZ`	`RET NZ`
`RETURNI ENABLE`	`RETI ENABLE`
`RETURNI DISABLE`	`RETI DISABLE`
`ADDCY`	`ADDC`
`SUBCY`	`SUBC`
`INPUT sX, (sY)`	`IN sX, sY` (no parentheses)
`INPUT sX, kk`	`IN sX, kk`
`OUTPUT sX, (sY)`	`OUT sX, sY` (no parentheses)
`OUTPUT sX, kk`	`OUT sX, kk`
`ENABLE INTERRUPT`	`EINT`
`DISABLE INTERRUPT`	`DINT`
`COMPARE`	`COMP`
`STORE sX, (sY)`	`STORE sX, sY` (no parentheses)
`FETCH sX, (sY)`	`FETCH sX, sY` (no parentheses)

Table 10-3: **Directives**

Function	KCPSM3 Directive	PBlazIDE Directive
Locating Code	`ADDRESS 3FF`	`ORG $3FF`
Aliasing Register Names	`NAMEREG s5, myregname`	`myregname EQU s5`
Declaring Constants	`CONSTANT myconstant, 80`	`myconstant EQU $80`
Naming the program ROM file	Named using the same base filename as the assembler source file	`VHDL 'template.vhd', 'target.vhd', "entity_name"`

Example 1: ADD16

```
1.    a_lsb EQU s0      ; rename register s0 as “a_lsb”
2.    a_msb EQU s1      ; rename register s1 as “a_msb”
3.    b_lsb EQU s2      ; rename register s2 as “b_lsb”
4.    b_msb EQU s3      ; rename register s3 as “b_msb”
5.
6.    LOAD a_msb, $AA             ; 1010 1010
7.    LOAD a_lsb, $FF             ; 1111 1111

1.    LOAD b_msb, 01              ; 0000 0001
2.    LOAD b_lsb, 01              ; 0000 0001

1.    ADD a_lsb, b_lsb                          ; add LSBs, keep result in a_lsb
2.    ADDC a_msb, b_msb          ; keep result in a_msb; Note ADDC is same as ADDCY

1.   ; We are adding  these two numbers:
2.   ; a = 1010 1010 1111 1111
3.   ; b = 0000 0001 0000 0001
4.   ;       1010 1100 0000 0000  expected result
5.   ;     a_msb (register s1) = AC
6.   ;     a_lsb   (register s0) = 00
```

Simulating **Input** Ports

```
; pBlazIDE syntax to define an input port
; input_port_name DSIN <port_id#>[, “<input_file_name>”]
;
switches DSIN $00, “switch_inputs.txt”
readport DSIN $1F
```

Example File
“switch_inputs.txt”

```
$FF
01
02
03
$A5
$5A
```

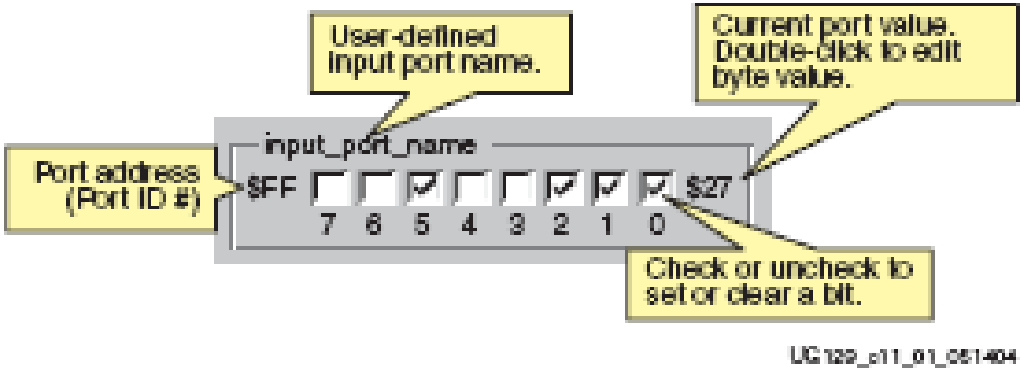


Figure 11-4: **The pBlazIDE DSIN Directive Defines an Input Port**

Simulating **Output** Ports

```
; pBlazIDE syntax to define a write-only output port
; output_port_name DSOUT <port_id#>[,“<output_file_name>”]
;
LEDs DSOUT $01, “output_values.txt”
writeport DSOUT $1E
```

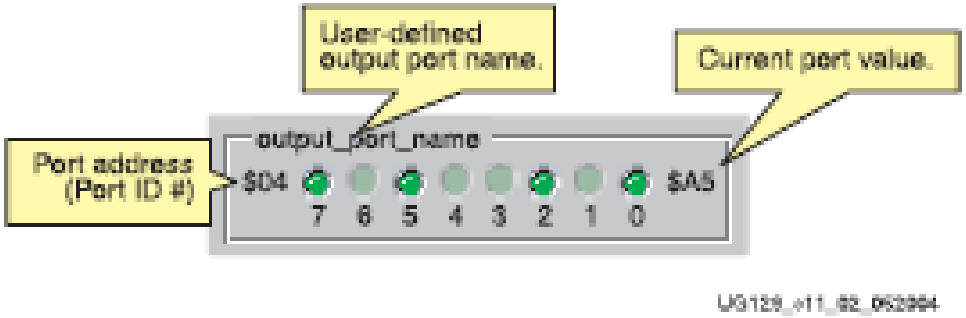


Figure 11-6: **The pBlazIDE DSOUT Directive Defines an Output Port**

Simulating **Input/Output** Ports

```
; pBlazIDE syntax to define a readable output port
; input_output_port_name DSIO
    <port_id#>[“<output_file_name>”]
mailbox DSIO $02, “mailbox_out.txt”
readwrite DSIO $1D
```

DSIO directive – models an output port that connects to <u>both IN_PORT and OUT_PORT ports</u> and has the <u>same port address</u> for input and output operations.

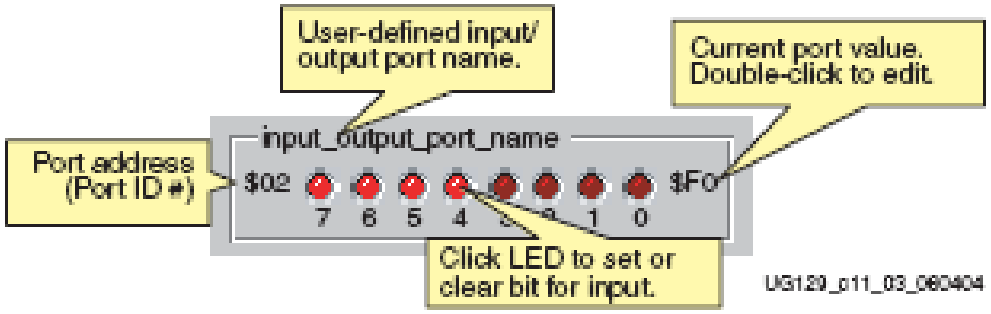


Figure 11-8: **The pBlazIDE DSIO Directive Defines an Output Port That Can Be Read Back**

Example 2: ADD16 with external inputs

```
1.    a_msb_in DSIN $00
2.    a_lsb_in   DSIN $01

1.    a_msb_out DSOUT $02
2.    a_lsb_out   DSOUT $03
3.
4.    a_lsb EQU   s0 ; rename register s0 as “a_lsb”
5.    a_msb EQU s1 ; rename register s1 as “a_msb”
6.    b_lsb EQU   s2 ; rename register s2 as “b_lsb”
7.    b_msb EQU s3 ; rename register s3 as “b_lsb”
8.
9.   ; LOAD a_msb, $AA      ; 1010 1010
10.  ; LOAD a_lsb, $FF                ; 1111 1111

1.    IN a_msb, a_msb_in
2.    IN a_lsb, a_lsb_in

1.    LOAD b_msb, 01                  ; 0000 0001
2.    LOAD b_lsb, 01                  ; 0000 0001

1.    ADD a_lsb, b_lsb        ; add LSBs, keep result in
      a_lsb
2.    ADDC a_msb, b_msb ; add MSBs, keep result in
      a_msb

1.    OUT a_msb, a_msb_out
2.    OUT a_lsb, a_lsb_out
```

Example:
; a = 1010 1010 1111 1111
; b = 0000 0001 0000 0001
; = 1010 1100 0000 0000
; a_msb (register s1) = AC
; a_lsb (register s0) = 00

Assembly code template: KCPSM3 Assembler

```
    NAMEREG sX, <name>   ; Rename register sX with <name>
    CONSTANT <name>, 00 ; Define constant <name>, assign value

      ; ROM output file is always called
      ;    <filename>.vhd

    ADDRESS 000          ; Programs always start at reset vector 0

    ENABLE INTERRUPT     ; If using interrupts, be sure to enable
                         ;   the INTERRUPT input

BEGIN:
     ; <<< your code here >>>

    JUMP BEGIN           ; Embedded applications never end

ISR: ; An Interrupt Service Routine (ISR) is
     ;   required if using interrupts
     ; Interrupts are automatically disabled
     ;   when an interrupt is recognized
     ; Never re-enable interrupts during the ISR
    RETURNI ENABLE       ; Return from interrupt service routine
                         ;   Use RETURNI DISABLE to leave interrupts
                         ;   disabled

    ADDRESS 3FF          ; Interrupt vector is located at highest
                         ;   instruction address
    JUMP ISR             ; Jump to interrupt service routine, ISR
```

Figure B-1: **PicoBlaze Application Program Template for KCPSM3 Assembler**

Assembly code template: pBlazIDE

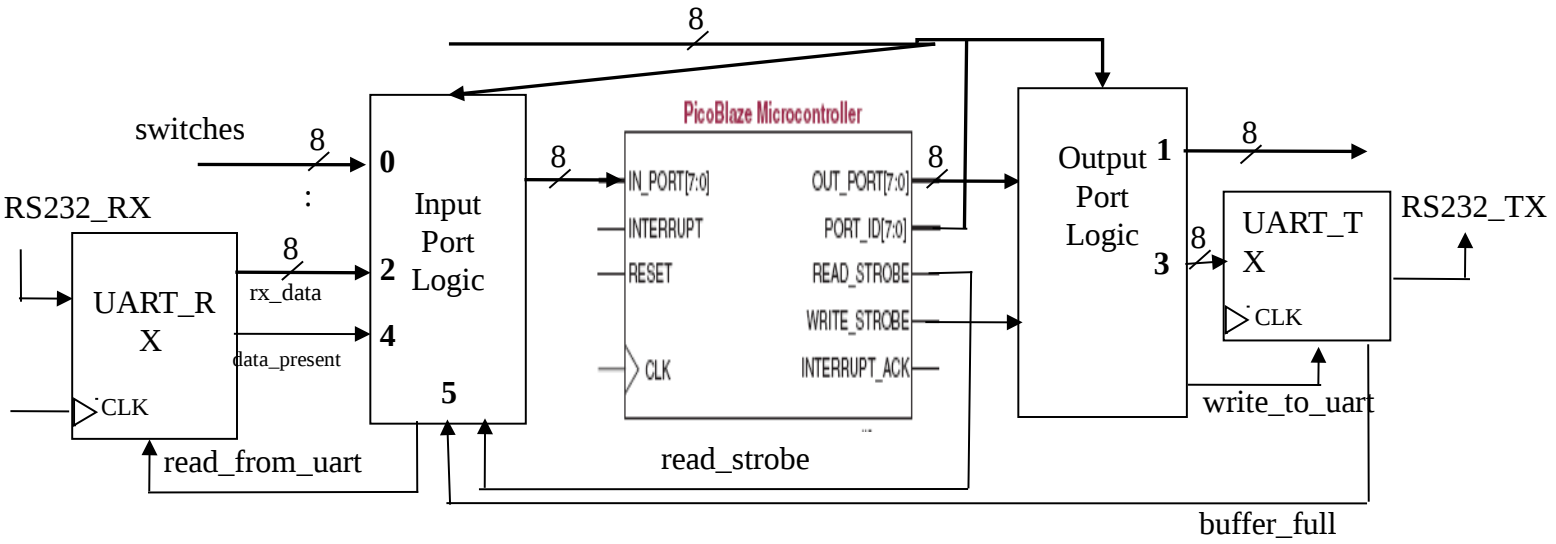
```
    <name> EQU sX          ; Rename register sX with <name>
    <name> EQU $00         ; Define constant <name>, assign value

    ; name ROM output file generated by pBlazIDE assembler
    VHDL "template.vhd", "target.vhd", "entity_name"

    <name> DSIN <port_id> ; Create input port, assign port address
    <name> DSOUT <port_id>; Create output port, assign port address
    <name> DSIO <port_id> ; Create readable output port,
                          ;   assign port address

    ORG 0; Programs always start at reset vector 0

    EINT                  ; If using interrupts, be sure to enable
                          ;   the INTERRUPT input

BEGIN:
     ; <<< your code here >>>

    JUMP BEGIN            ; Embedded applications never end

ISR:                      ; An Interrupt Service Routine (ISR) is
                          ;   required if using interrupts
                          ; Interrupts are automatically disabled
                          ;   when an interrupt is recognized
                          ; Never re-enable interrupts during the ISR
    RETI ENABLE           ; Return from interrupt service routine
                          ;   Use RETURNI DISABLE to leave interrupts
                          ;   disabled

    ORG $3FF              ; Interrupt vector is located at highest
                          ;   instruction address
    JUMP ISR              ; Jump to interrupt service routine, ISR
```

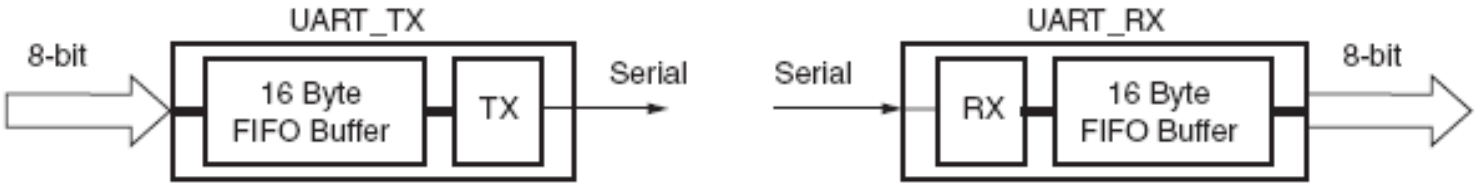
Figure B-2: **PicoBlaze Application Program Template for KCPSM3 Assembler**

Example 3: ADD16 with interrupts

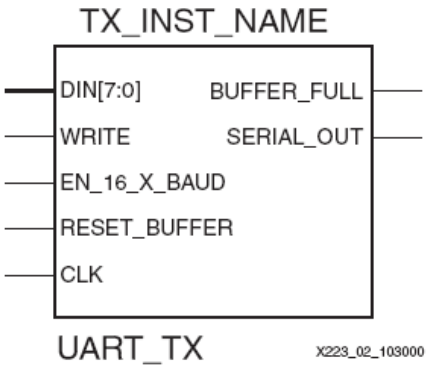
```
1.      a_lsb EQU s0              ; rename register s0 as “a_lsb”
2.      a_msb EQU s1              ; rename register s1 as “a_msb”
3.      b_lsb EQU s2              ; rename register s2 as “b_lsb”
4.      b_msb EQU s3              ; rename register s3 as “b_lsb”
5.      counter_port    DSOUT     4
6.      interrupt_counter   EQU   s4
7.
8.      ORG 0                     ; Programs always start at reset vector 0
9.
10.     EINT                      ; If using interrupts, be sure to enable
11.                               ; the INTERRUPT input
12.     BEGIN:
13.
14.     LOAD a_msb, $AA           ; 1010 1010
15.     LOAD a_lsb, $FF           ; 1111 1111

1.      LOAD b_msb, 01            ; 0000 0001
2.      LOAD b_lsb, 01            ; 0000 0001

1.      ADD a_lsb, b_lsb          ; add LSBs, keep result in a_lsb
2.      ADDC a_msb, b_msb         ; add MSBs, keep result in a_msb
3.
4.      JUMP BEGIN                ; Embedded applications never end
5.
6.      ISR:
7.      ADD interrupt_counter, 1                ; increment counter
8.      OUT interrupt_counter, counter_port     ; display counter
9.                                ; An Interrupt Service Routine (ISR) is required if using interrupts
10.                               ; Interrupts are automatically disabled when an interrupt is recognized
11.                               ; Never re-enable interrupts during the ISR
12.     RETI ENABLE               ; Return from interrupt service routine
13.                               ; Use RETURNI DISABLE to leave interrupts disabled
14.     ORG $3FF                  ; Interrupt vector is located at highest  instruction address
15.     JUMP ISR                  ; Jump to interrupt service routine, ISR
```

Simple LoopBack System

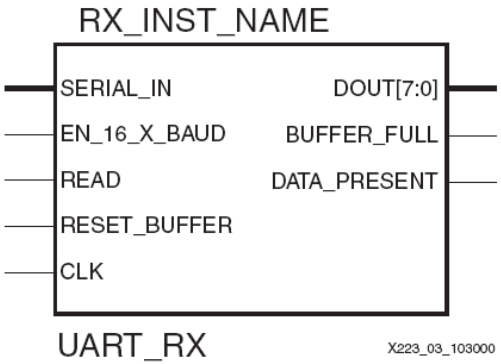




UART Modules

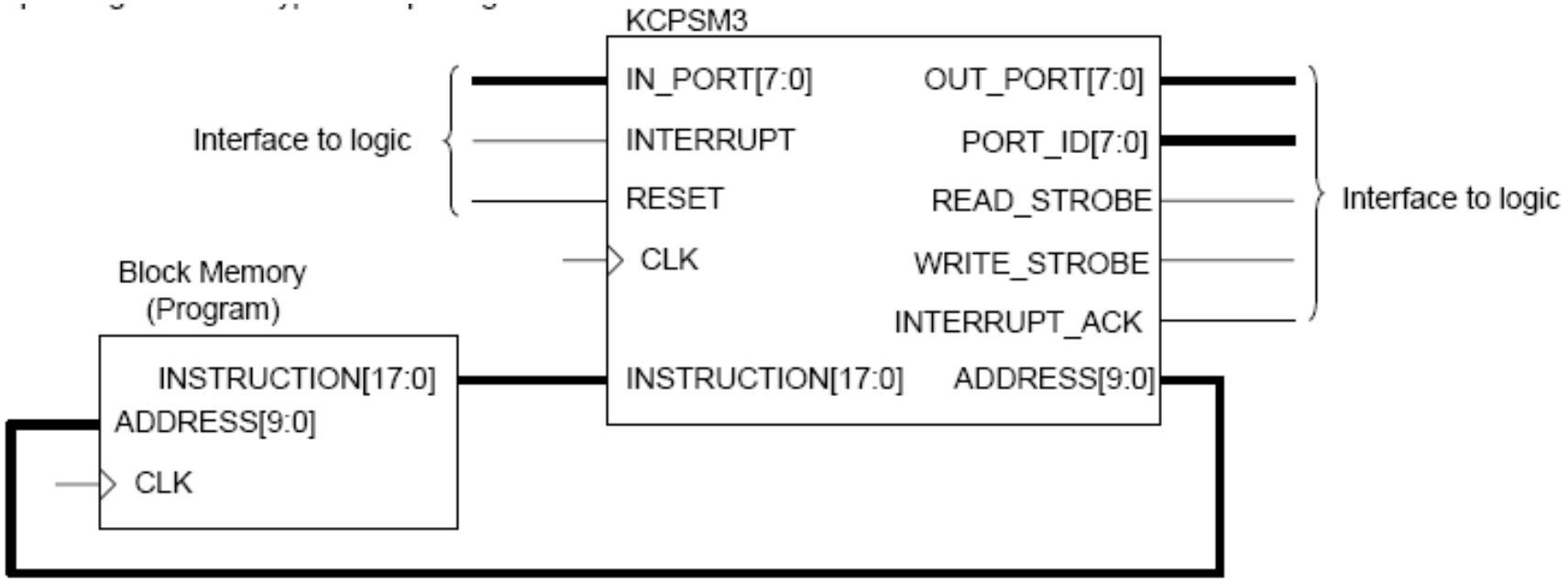


- Features
  - Simple
  - 1 Start bit
  - 8 serially transmitted bits (LSB First)
  - 1 Stop Bit
  - 16 byte FIFO buffer



- Transmitter Signals
  - DIN[7:0]
  - WRITE
  - SERIAL_OUT
  - BUFFER_FULL
- Common Signals
  - CLK
  - EN_16_X_BAUD
  - RESET_BUFFER
  - BUFFER_FULL
- Receiver Signals
  - DOUT[7:0]
  - DATA_PRESENT
  - READ
  - SERIAL_IN

KCPSM3 Module



KCPSM3 Architecture