John Gangemi
U 6871-4612
CDA 4203 Sec 001 Spring 2015
2/2/15

Homework 2: PicoBlaze Microcontroller

1. (5 pts.) PicoBlaze Architecture

Answer the following questions about the PicoBlaze Architecture:

a. List all storage units and what kind of data they can store (data, instructions, addresses)?

i. 1K x 18 Instruction PROM [instructions]

ii. 16 Byte-wide registers [data]

iii. 64-byte scratchpad RAM [data]

iv. Call/Return stack [instruction addresses]

b. What is the address range (in hex) for the instruction PROM?

i. PROM can hold 210 instructions where the range is from 0 to 2n-1 in binary (11111111112) and in hexadecimal the range is from 000 to 3FF.

c. How many cycles does an instruction consume?

i. All instructions under all conditions will execute over 2 clock cycles

d. What is the purpose of NOP instruction?

i. To stall program execution by a single instruction for more predictable timing.

e. List all instructions that ***directly*** manipulate the call/return stack.

i. CALL, RETURN, RETURNI, enabled INTERRUPT event, RESET event

f. How many clock cycles does instruction decode take?

i. Instruction decode takes one clock cycle.

g. What is the purpose of the interrupt input? How can we disable interrupts?

i. To handle multiple asynchronous external events. To disable interrupts call the DINT instruction.

h. What flags does the test instruction affect? Does it affect the register contents?

i. The test instruction only affects the CARRY and ZERO flags, and does not affect register contents.

i. When we execute a call instruction, does the PicoBlaze save the register file contents?

i. No, the register file address space is not directly modified by the CALL instruction per Table 3-2 from PicoBlaze Documentation.

j. What is the difference between ADD and ADDCY instructions? Explain why we need two variants of ADD instructions?

i. ADD and ADDCY instructions compute the sum of two 8-bit operands, however the ADDCY instruction accounts for the CARRY flag having been set. Having both variants increases program flexibility.

2. (5 pts.) PicoBlaze -- Word Parity:

```
IN_PORT DSIN $00
OUT_PORT DSOUT $01

LOOP:
 IN s0, IN_PORT
 DINT
 TEST s0, $FF
 EINT
 LOAD s1, $00
 ADDC s1, $00
 OUT s1, OUT_PORT
 JUMP LOOP
```

3. (15 pts.) Simple Loopback System

a. **(2 pts.)** Explain under what condition(s) should the following signals be asserted?

i. read_from_uart

- data_present (port_id 4) = 1 and read_strobe = 1 and buffer_full (port_id 5) = 0

ii. write_to_uart

- write_strobe = 1

b. **(4 pts.)** cold_start: Write code to output a message "Welcome to Loopback!" to the serial port. You need encode the message in ASCII format.

```
; adapted from PicoBlaze KCPSM6 Instruction Set Reference Documentation
; topic: "Text Strings"
; author: Xilinx
; page: 99

buffer_full DSIN $05
tx_data DSOUT $03

LOOP:
  LOAD s0, $61   ;W
  CALL TRANSMIT
  LOAD s0, $65   ;e
  CALL TRANSMIT
  LOAD s0, $6B   ;l
  CALL TRANSMIT
  LOAD s0, $63   ;c
  CALL TRANSMIT
  LOAD s0, $6F   ;o
```

```
  CALL TRANSMIT
  LOAD s0, $6D   ;m
  CALL TRANSMIT
  LOAD s0, $65   ;e
  CALL TRANSMIT
  LOAD s0, $20   ;space
  CALL TRANSMIT
  LOAD s0, $74   ;t
  CALL TRANSMIT
  LOAD s0, $6F   ;o
  CALL TRANSMIT
  LOAD s0, $20   ;space
  CALL TRANSMIT
  LOAD s0, $4C   ;L
  CALL TRANSMIT
  LOAD s0, $6F   ;o
  CALL TRANSMIT
  LOAD s0, $6F   ;o
  CALL TRANSMIT
  LOAD s0, $70   ;p
  CALL TRANSMIT
  LOAD s0, $62   ;b
  CALL TRANSMIT
  LOAD s0, $61   ;a
  CALL TRANSMIT
  LOAD s0, $63   ;c
  CALL TRANSMIT
  LOAD s0, $6B   ;k
  CALL TRANSMIT
  LOAD s0, $21   ;!
  CALL TRANSMIT
  LOAD s0, $0   ;null
  CALL TRANSMIT
  JUMP LOOP

TRANSMIT:
  IN s1, buffer_full
  COMP s1, $00   ; set zero flag if buffer_full == 0
  JUMP NZ, TRANSMIT   ; loop recursive if zero flag not set
  OUT s0, tx_data   ; transmit data
  RET
```

c. **(4 pts.)** led_echo: Write code to read switches and write it, inverted, to the LEDs.

```
switches DSIN $00
leds DSOUT $01

LOOP:
  IN s0, switches
  XOR s0, $FF
  OUT s0, leds
  JUMP LOOP
```

d. **(5 pts.)** rs232_echo: Write code to check if a byte has been received by UART_RX. If so, send it back to PC via UART_TX.

```
data_present DSIN $04
buffer_full DSIN $05
rx_data DSIN $02
tx_data DSOUT $03

LOOP:
  IN s2, rx_data
  CALL CHECK_DATA
  CALL CHECK_BUFF
  OUT s2, tx_data
  JUMP LOOP

CHECK_DATA:
  IN s0, data_present
  COMP s0, $01   ; set zero flag if data_present == 1
  RET Z   ; return if zero is set
  JUMP CHECK_DATA

CHECK_BUFF:
  IN s1, buffer_full
  COMP s1, $00   ; set zero flag if buffer_full == 0
  RET Z   ; return if zero is set
  JUMP CHECK_BUFF
```