Alexander_Sulaberidze HW6: Processor

Refer to the end of the markdown file for a summary on the assembler instructions

Description:

The main part of the Processor consists of the following:

- **The 64K x 16Bit InstructionROM, which contains the assembler code for the program written by the user**
- **The CPU which handles the 16Bit Instructions in the InstructionROM pointed to by the Program Counter inside of the CPU**
- **The 4K x 16Bit RAM, which is the memory used by the program to complete tasks and move data back and forth between the CPU and the RAM**
    - Address 0 through 4 of the RAM is reserved for the following:
        - RAM(0) -> Zero Instruction stores trash in here
        - RAM(1) -> Stores Jump Register Value during an ISR Routine
        - RAM(2) -> Stores Temp Register Value during an ISR Routine
        - RAM(3) -> Stores A Register Value during an ISR Routine
        - RAM(4) -> Stores B Register Value during an ISR Routine
    - The Last 16 Bits are reserved for the GPIO_Read Instructions that read the GPIO Pins into the RAM
- **The Peripherals specified by the Assignment**

The CPU Contains 5 Registers including the Program Counter, all of which (Except the program counter) is managed by the registerManager subcircuit:

- **The jmpRegister:** Points to the instruction in the InstructionROM that will be jumped to during a successful Jump operation.
    - Supports loading and storing from/into the RAM
    - Contents **can not** be moved into a different register
- **The tempRegister:** Used as the register we compare the B Register to during branching commands
    - Supports loading and storing from/into the RAM
    - Contents **can not** be moved into a different register
- **The A Register:** One of the 2 main registers, is one of the inputs to the ALU.
    - Supports loading and storing from/into the RAM
    - Contents **can** be moved into a different register
- **The B Register:** The other one of the 2 main registers, is one of the inputs to the ALU.
    - Supports loading and storing from/into the RAM
    - Contents **can** be moved into a different register

The CPU Supports 2 main types of instructions Specified by the MSB (Bit 15), Each of which supports 2 subtypes of instructions specified by the second MSB (Bit 14), all of this plus an extra type of instruction for communicating with the GPIO Peripheral:

- **Bit 15 = 0 --> Memory Access Instruction**
    - **Bit 14 = 0 --> Store Instruction**
    - **Bit 14 = 1 --> Load Instruction**
- **Bit 15 = 1 --> CPU Instruction**
    - **Bit 14 = 0 --> ALU Instruction**
    - **Bit 14 = 1 --> Jump instruction**
- **1111 xxxx xxxx xxxx -> GPIO Instruction**

The CPU Also supports a 3 bit ALU Error Code Generation

- **Bit 0 -> Zero Div Flag**
- **Bit 1 -> Overflow Flag (Not Actually An Error)**
- **Bit 2 -> Negative Flag (Not Actually An Error)**

The Rest of the details about the instructions will be discussed further in the text.

CPU Subcircuit 1: ALU (Used during an ALU Instruction (10xx xxxx xxxx xxxx)):

2 Modes: ALU Mode & Generate Const mode. Works Completely Asynchronously (No Clock).

The Mode is Chosen by bit 11 (ALU Op Code) of the instruction.

- **bit 11 = 0** *ALU Mode takes the outputs of the A and B Register as inputs and does the operation specified by bits 0 through 3 on them.*
- **bit 11 = 1** Generate Const Mode takes bits 0 through 11, pads them from the left with 0000 and sends them as the output. (Supports generating a 11 bit constant value to load into a register)

The Operation type during ALU Mode is selected by bit 3.

- ***bit 3 = 0*** *--> Logic Instruction*
- ***bit 3 = 1*** *--> Arithm. Instruction*

ALU Subcircuit 1: Logic Unit:

- **Choice between 8 different logic operations using bits 0 through 2**
- **Instructions**: -- 000 --> AND -- 001 --> OR -- 010 --> XOR -- 011 --> NAND -- 100 --> NOR -- 101 --> NOT (On the contents of the A Register) -- 110 --> Log. Shift left (On the contents of the A Register) -- 111 --> Log. Shift right (On the contents of the A Register)

ALU Subcircuit 2: Arithm. Unit:

- **Choice between 8 different arithm. operations using bits 0 through 2**
- **Instructions**: -- 000 --> Sum -- 001 --> Subtract -- 010 --> Multiply -- 011 --> Divide (A by B) -- 100 --> Arithm. Shift right (On the contents of the A Register) -- 101 --> Remainder (of A by B) -- 110 --> Add 0 to A (For moving to another register) -- 111 --> Add 0 to B (For moving to another register)

The ALU also supports 3 Flags that only trigger during Arithm. Mode:

- *Zero Div Flag* (If during Divide Mode the B Register contains the value 0)
- *Overflow Flag* (If the resulting number is larger than 16 bits)
- *Negative Flag* (If the resulting number is negative)

Examples:

- **Bits 0 through 3 = 0101** --> Logical NOT Operation (on A)
- **Bits 0 through 3 = 0001** --> Logical AND Operation (on A and B)
- **Bits 0 through 3 = 1101** --> Remainder Operation (on A and B)
- **Bits 0 through 12 = 1100000001111** --> Generate Const Operation that outputs 0000 1000 0000 1111

CPU Subcircuit 2: registerManager:

Manages the Registers (Except the Program Counter) of the CPU by Handling the first 4 opCodes of the instruction.

Takes In the Output of the RAM and the Output of the ALU plus the first 4 opCodes to decide how to transfer the data between the registers and/or between a register and the RAM

Outputs the outputs from the A and B Registers (To feed into the ALU), the output from the jump Register (To handle jump instructions down the line), and the value to be stored in the RAM (Always outputs this, only actually stores the data if the code is: 00xx xxxx xxxx xxxx)

Works Synchronously, has a clock input, and a reset input that resets all of the registers

- **opCodes 3 & 4 choose the target/source register for the operation.**
  - xx00 xxxx xxxx xxxx -> register B
  - xx01 xxxx xxxx xxxx -> register A
  - xx10 xxxx xxxx xxxx -> temp register
  - xx11 xxxx xxxx xxxx -> jump register

CPU Subcircuit 3: Program Counter:

Essentially just a Counter Register

CPU Subcircuit 4: jmpManager:

Manages Jump Instructions read by outputting the signal that allows the CPU to overwrite the contents of the Program Counter with the contents in the jmpRegister, Effectively jumping to the instruction pointed to by the jmpRegister

Does the aforementioned while handling the conditional statements specified by the Jump Instruction The Code Corresponding to a Jump Instruction: 11xx xxxx xxxx xxxx

In this case, opCodes 3 & 4 act as the Jump Conditional Managers

- 1100 xxxx xxxx xxxx -> Unconditional Jump
- 1101 xxxx xxxx xxxx -> BREQ (Jump if the Contents of Temp and B are equal)
- 1110 xxxx xxxx xxxx -> BRNQ (Jump if the Contents of Temp and B are not equal)
- 1111 xxxx xxxx xxxx -> Reserved for GPIO Instructions

CPU Subcircuit 5: ISRHandler:

Handles Interrupt signals and jumping to and from ISR Routines, contains the RA Register to store the return address to go back to after the ISR Routine.

Contains A ROM (InterruptVectorTable) that stores the addresses in the instruction memory that point to the ISR Routines.

ISR Routines are limited to 256 Instructions, the handler leaves the ISR Routine after 256 Clock Cycles.

Features:

- **6 Inputs: (JumpReg, JumpLoadCommand, Interrupt, InterruptFlags, currPCVal, Clock)**

- **2 Outputs: (PCLoadVal, PCLoadCom)**

- **If the Interrupt Signal is 0, Just passes the jumpRegisterValue and the JumpLoadCommand unchanged**
- **When the Interrupt signal is pulled high, we go into ISRMode, jumping to the address pointed to by the vector table for the next 50 cycles and then returning to the original PC value**
- **Saving and returning the state should be handled by the user within the ISR Routine using the reserved RAM Addresses**
    - Each ISR Routine should have the following form:
        - Store Jump RAM(1) -> **3001**
        - Store Temp RAM(2) -> **2002**
        - Store A RAM(3) -> **1003**
        - Store B RAM(4) -> **0004**
        - ISR Routine lines -> **Multiple Lines**
        - Load Jump RAM(1) -> **7001**
        - Load Temp RAM(2) -> **6002**
        - Load A RAM(3) -> **5003**
        - Load B RAM(4) -> **4004**

InterruptVectorTable

Currently has 8 Addresses that can be changed by hand By Default: (InterruptFlags -> Address in hex)

- 001 -> F8FF **(IFOCA)**
- 010 -> F9FF **(IFOCB)**
- 011 -> FAFF
- 100 -> FBFF **(IFO)**
- 101 -> FCFF
- 110 -> FDFF
- 111-> FEFF **The rest of the unassigned addresses do nothing and are left for future extensions**

To Use the Interrupts, the ISR Routines have to be written at the corresponding ISR Addresses from the InterruptVectorTable

Timer Peripheral

The Same Timer from Assignment 5 with small changes:

Added Reset Signal Overflow Signal improved so it does not fire at startup anymore.

GPIO Peripheral (8 Bit)

4 GPIO Pins, Each with an 8 Bit Config Register

Each of the pins have a register input and a register load signal which allows the user to load the Config into it Asynchronously by hand.

The GPIO Peripheral comes with its own GPIO Handler, that can execute GPIO Instructions (1111 xxxx xxxx xxxx), these instructions allow the user to read the GPIO Pin data, and or configure the GPIO Pin Config using code

Each of the pins also takes in the output from the Timer Peripheral for configuring the PWM Mode

Configuring the GPIO_Config_X-s is up to the user and is not regulated by the processor itself.

The 5 MSB of the GPIO_Config_X do nothing, only the 3 LSB have part in configuring the pin.

- **Bit 0 -> I/O Bit:**
    - 0 For Input Mode (Whatever is at the GPIO_IN_X, shows up at the Output)
    - 1 For Output Mode (Configured by the next 2 bits)
- **Bit 1 -> HIGH/LOW Bit: (For Output Mode)**
    - 0 For LOW
    - 1 For HIGH
- **Bit 2 -> Timer Mode Bit (For Output Mode)**
    - Feeds the output of the Timer Peripheral to the GPIO_OUT_X

GPIO Handler:

Stores to the last 4 Addresses of the RAM in GPIO_Read mode. (FFF - Pin 4, FFE - Pin 3, FFD - Pin 2, FFC - Pin 1) If the Pin is read as high, all 16 bits go high Writes to the GPIO_Config_X Registers in GPIO_Write mode

GPIO Instruction Guide:

1111 aabx pppp pppp

- aa refer to the select bits, they choice which GPIO_Pin_x we are operating on.
- b refers to the mode, b = 0 -> Read Mode, b = 1 -> Write Mode
- pppp pppp refers to the parameters we write to the GPIO_Config_x, when in write mode, as already mentioned, only the 3 LSB Matter here

Summary of the Assembler Language

0000 0000 0000 0000 –> Zero Instruction, Stores Register B at RAM(0) Ram(0) is reserved, so effectively, this does nothing.

Bit 15 = opCode1 –> MemoryAccess Mode(0)/CPU Instr Mode(1)

Bit 14 = opCode2 --> Store(00)/Load(01)/ALU Instr(10)/JUMP(11)

Bit 13-12 = opCode3 & 4--> For 11, Jump Conditions, for the rest, target/src Registers -For the Jump Conditions refer to the jmpManager Description -For the target/src Registers, refer to the registerManager Description

Bit 11 = ALU opCode --> In ALU Instr mode, ALU Mode(0)/Generate CONST Mode(1)

Bits 0-3 (In ALU Mode: 10xx 0xxxx xxxx xxxx) = ALU Instructions

- Refer to the ALU Subcircuit Descriptions

Bits 0-12 (In Memory Access Mode: 0xxx xxxx xxxx xxxx) = RAM Address Bits 0-12 (In Generate Const Mode: 10xx 1xxx xxxx xxxx) = Const Value

Testing

Setup:

Reset Simulation GPIO Peripheral Setup Timer Peripheral Setup

To Test, you need to write the Assembler Instructions into the Instruction ROM by hand, I will provide some test codes below: (The format is the following, Hex - Binary -> Description):

I recommend advancing the clock by hand, cycle by cycle, in order to see how the code works

---

Test Sum & Move(Should result in 4 being stored at address 7 of the RAM)

9081 - 1001 1000 0000 0001 -> Load Const 1 Into A 8803 - 1000 1000 0000 0011 ->Load Const 3 Into B A008 - 1010 0000 0000 1000 -> Sum & load into Temp 2005 - 0010 0000 0000 0101 -> Store Temp at address 5 of the RAM 6005 - 0110 0000 0000 0101 -> Load address 5 of RAM contents into temp 2007 - 0010 0000 0000 0111 -> Store temp at address 7 of the RAM

Test XOR (Should Res in the Bits Being Flipped, so 0733 Or 0000 0111 0011 0011, at address 4 of the RAM)

98CC - 1001 1000 1100 1100 ->Const (in binary)0000 0000 1100 1100 Into A 8FFF - 1000 1111 1111 1111 ->Const (in binary)0000 0111 1111 1111 into B 8002 - 1000 0000 0000 0010 ->XOR and load into B 0008 - 0000 0000 0000 1000 -> Store B at address 8 of the RAM

Test Simple Loop (Should Result in RAM(4) Containing 10 (000a))

0 - **8800 - 1000 1000 0000 0000 -> Load 0 into B** 1 - **A805 - 1010 1000 0000 0101 -> Load 5 into Temp** 2 - **B80E - 1011 1000 0000 1110 -> Load 14 into Jump** 3 - **D000 - 1101 0000 0000 0000 -> BREQ** 4 - **0006 - 0000 0000 0000 0110 -> Store B RAM(6)** 5 - **8802 - 1000 1000 0000 0010 -> Load 2 into B** 6 - **9008 - 1001 0000 0000 1000 -> Sum and load into A** 7 - **1005 - 0001 0000 0000 0101 -> Store A RAM(5)** 8 - **9801 - 1001 1000 0000 0001 -> Load 1 into B** 9 - **4006 - 0100 0000 0000 0110 -> Load B RAM(6)** 10 - **8008 - 1000 0000 0000 1000 -> Sum and load into B** 11 - **5005 - 0101 0000 0000 0101 -> Load A RAM(5)** 12 - **B802 - 1011 1000 0000 0010 -> Load 2 into Jump** 13 - **C000 - 1100 0000 0000 0000 -> Jump**

Test Timer Config + PWM (GPIO_PIN_2_PWM)

GPIO Config

- **Set GPIO_Config_2 to 0000 0101**
- **Load it inside by setting bit 1 of GPIO_LOAD_1_4 to 1 and turn the bit back to 0**

Timer Config

- **Set the PSC to the desired value**
- **Set the OCA Register to 0000 0100**
- **Set the OCB Register to 0001 0000**
- **Set the Param Register to 0000 1000 to set it to Waveform Generation Mode**
- **Load all of the values into their registers by setting all 3 of the TIMER_CONFIG_LOAD bits high and back low**

This generate a PWM Signal that will be on when the Timer is larger than OCA and Smaller than OCB and route the signal GPIO_PIN_2

Test ISRHandler

Timer Config:

- **Set the PSC to the desired value**

- **Set the OCA Register to 0000 0100**

- **Set the Param Register to 0000 0010 to turn the OCAIE Interrupt on, this will cause the interrupt 8 clock cycles after the start of the program**

- **Can be tested with the for loop Code**

ISR Routine Code:

At F8FF enter the following code:

- **3001** -> Store Jump RAM(1)
- **2002** -> Store Temp RAM(2)
- **1003** -> Store A RAM(3)
- **0004** -> Store B RAM(4)
- **9805** -> Load Const 5 into A
- **8803** -> Load Const 3 into B
- **900A** -> Multiply and store in A
- **100A** -> Store A RAM(10)
- **7001** -> Load Jump RAM(1)
- **6002** -> Load Temp RAM(2)
- **5003** -> Load A RAM(3)
- **4004** -> Load B RAM(4)

This ISR Routine should leave a 000f (15) at RAM(10), as well as the contents of the 4 registers at the start of the routine at RAM 1 through 4, as well as doing whatever the program was originally meant to do.

Test GPIO

If the GPIO_Pin_3 is configured to input mode, running the line f800, should read the value of the pin (set by hand) into address ffe