Engr433 Class Project Autumn, 2019

Title

8-bit RISC embedded processor design

Objectives

- Learn about basic microprocessor architecture;
- Learn about multi-input system controllers;
- Learn to design and debug a complex digital system;
- Learn to address FPGA design flow and implementation issues.

References

- This handout;
- Xilinx documents on the class web page.

Preamble

There's never been a better time to design a new microprocessor. With Intel on the skids and all of it's competitors wondering what to do next, the time is ripe for Walla Walla University to increase it's share in the merchant semicondutor market. The following description provides the basis for a new and improved RISC processor.

General Architecture

A microprocessor can be treated as a sophisticated programmable controller. The architecture of a microprocessor includes a finite state machine (FSM) controller and a data path unit (DPU) as shown in Figure 1. Note that all input and output conditioning logic has been omitted. The FSM controller can be

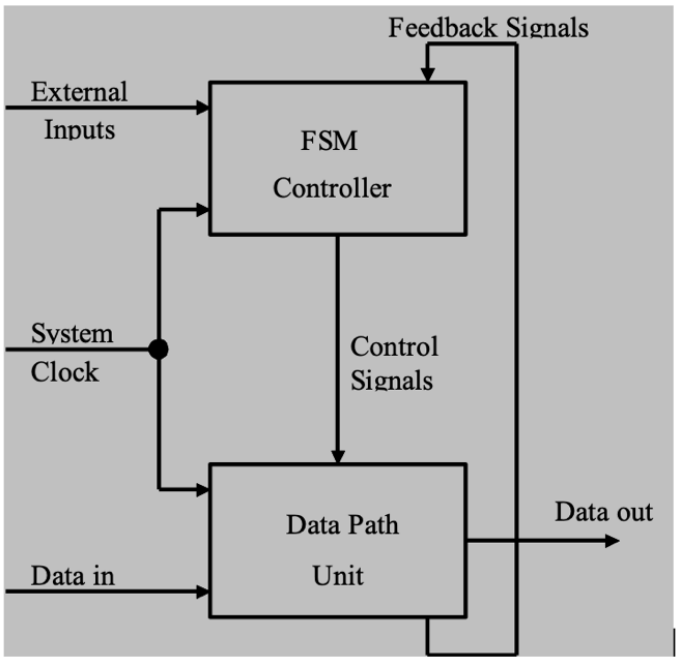

Figure 1 – Microprocessor Architecture.

modeled as a standard Mealy machine with the external inputs and feedback signals acting as the system inputs. The outputs of the Mealy machine are the control signals to the DPU. All data in the processor is handled by the data path unit.

Figure 2 below shows an expanded view of the FSM controller. All instructions are stored in the instruction memory. The outputs of the program counter act as an address input to the instruction memory. Each instruction is interpreted by the controller which then generates appropriate signals to control the DPU. The execution of the instruction is carried out entirely in the DPU.

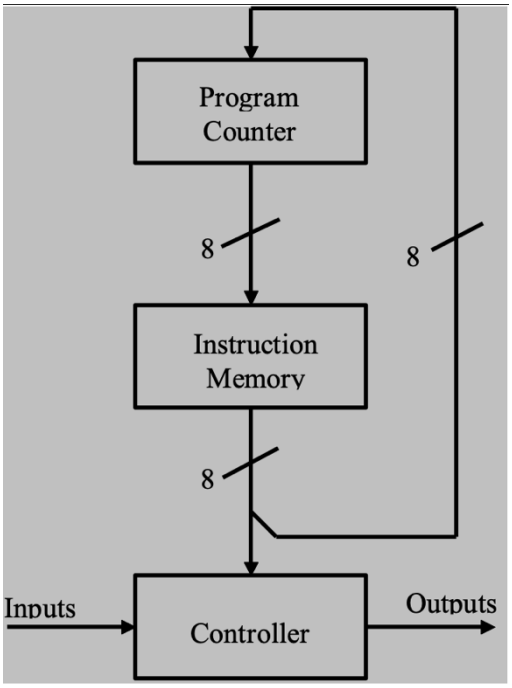

Figure 2 - FSM Controller.

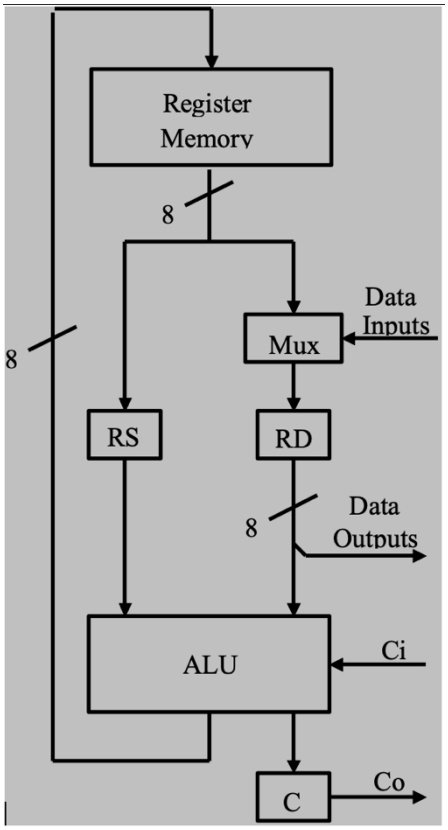

Figure 3 – Data Path Unit.

Figure 3 shows an expanded diagram of the data path unit. Note that this is a general model. The ALU accepts two 8-bit inputs along with the Ci and generates an 8-bit output and a Co. The ALU is purely combinational. The output of the ALU is stored back in the register memory. It is necessary to get data into the DPU via switches, memory, etc. and also necessary to get data from the DPU to external sources like memory, displays, etc.

Figure 4 shows a possible programming model for the RISC processor. Note that this model includes four 8-bit registers for data storage (R0-R3), 256 bytes of 8-bit instruction memory, an 8-bit ALU with a carry bit, 8 switches for data input, and 8 LED's for data output.

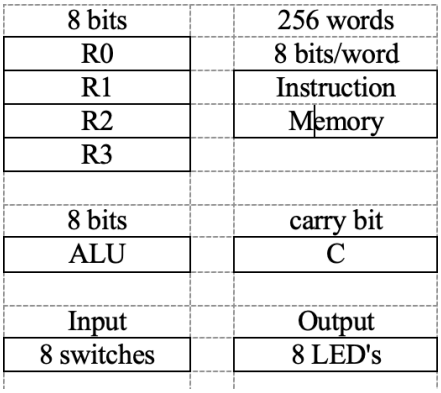

Figure 4 – Programming Model.

Instruction Set

Your design must implement, at a minimum, the instructions shown below. You could loosely group these instructions into 3 classes: register instructions, branch instructions, and halt-I/O instructions. Note that the Carry flag is affected only if an arithmetic instruction is executed and Co equals 1. This happens whenever an addition overflows or when a subtraction does not produce a borrow.

Mnemonic	Operation	Description
ADDC	RDn ← RSn + RDn + C_i	Add the source register to the destination register to the Carry in flag and store the result back into the destination register. Carry flag is set if the result of the addition overflows.
AND	RDn ← RSn • RDn	Logically AND the source and destination registers and place the result back into the destination register.
CLRC	0 → C	Clear the Carry flag.
HALT	None	HALT stops the processor from further execution until a reset or continue signal is received.
JC	IF C then PC ← Addr8 else PC ← PC + 2	If the carry flag is set then the next byte following the JC instruction is loaded into the PC, otherwise the PC is incremented by 2.
JNC	See above.	See above.
JMP	PC ← Addr8	The byte following the JMP instruction is loaded into the PC.
JZ	If Rn = 0 Then PC ← Addr8 Else PC ← PC + 2	If register N is zero, then the byte following the JZ instruction, Addr8, is loaded into the PC, otherwise the PC is incremented by 2.
MOVI	Data8 → Rn	Loads the Register Rn with the 8-bit data, Data8.
MOVR	RSn → RDn	Moves 8-bit data between registers.
NOP	No operation	No operation is performed.
OR	RDn ← RSn | RDn	The source register, RSn and the destination register, RDn are logically OR'ed together and the result is stored in RDn.
PI	Rn ← Pin	The contents of the input port are loaded into the register Rn.
PO	Rn → Pout	The contents of Rn is transferred to the output port
SETC	C ← 1	Set the carry flag to 1.
SUBB	RDn ← RSn - RDn - ~Ci	The source register and the complement of the carry flag are subtracted from the destination register and the result is stored in the destination register. Co is equal to the borrow out from subtraction.

Table 1 on the right gives some examples of register instructions.

Mnemonic		Operation
Clear	R2	R2 - R2 → R2
Increment	R3	R3 + 1 → R3
Shift Left	R1	R1 + R1 → R1
ADD	R1, R0	R1 + R0 → R0
Subtract	R2, R0	R0 - R2 → R0
Move	R3, R2	R3 → R2

Table 1 - Example Register Instructions.

Sample Program

The program in Table 2 multiplies two 8-bit unsigned numbers to produce a 16-bit product. Note that the multiplicand is set in the switch register before the program is started. When the program halts the first time, the multiplier is set in the switches and then the machine is restarted. When it halts again, the lights display the most significant eight bits of the product, and when restarted it will halt and display the least significant eight bits. You should check over the program to verify your understanding of the instruction set.

Address	Operation	Comments
0	SW → R1, Halt	X → R1
1	R0 - R0 → R0	Clear R0
2	R0 - R1 - 1 → R0	- (X + 1) → R0
3	SW → R1	Y → R1
4	R2 - R2 → R2	Clear product
5	R3 - R3 → R3	
6	R0 + 1 → R0	Increment R0 and
7	BR to ADDR B if C = 1	test for zero
8	R1 + R2 → R2	Add Y to product
9	R3 + C → R3	
A	Branch to ADDR 6	Loop back
B	R3 → LED's, Halt	Display product
C	R2 → LED's, Halt	
D	Branch to ADDR 0	Start over

Table 2 - Sample program to multiply two 8-bit numbers.

Assignment

Implement the RISC processor as described above. Test your system with the multiplication program shown in Table 2. Next, write and test a program that inputs a pattern from the switch register and rotates it in the lights. The pattern should spend an equal amount of time in each position.

Implementation

You must implement your design using VHDL on the Xilinx FPGA boards. The following inputs and outputs are recommended for ease of operation and debugging:

Reset - A button to initialize control and clear PC.
Start - A button that starts program execution at the current location pointed to by PC.
Incpc - A button to increment the PC when the machine is halted.
Ss - A button that will single step through each micro-instruction.
Sw - Eight toggle switches used to input data to a program.
Dataout - One seven-segment display that shows the contents of RD at all times.
Pcout - One seven-segment display that shows the contents of PC at all times.

Grading Criteria

Your final project is worth 25% of your grade this quarter. Here are the guidelines for how the final project points will be allocated:

- Hardware checkout – 60%

You must bring your project to the class room at the final test time or before and demonstrate it to the instructor.

- Project Assignment #1 – 5%
- Final report – 15%
  Each group must turn in a report. A sample report will be available in Dr Nelson's office.
  Your report must contain the following elements at a minimum:
  - Cover sheet (including title of your processor)
  - Table of Contents
  - Introduction
  - Design philosophy
  - Detailed architecture description
  - Results
  - Lessons learned
  - Conclusions
  - Appendices
    - Schematics
    - VHDL code
    - Design path
    - Instruction set details
    - Example programs
  - Schematics, pictures, and videos as appropriate.
- Team evaluation and attendance – 10%
- Instructor evaluation – 10%

Hints

- **Think Block Diagram**!
- Read this handout several times until a solid grasp of the overall project is firmly in hand.
- Began discussions with your lab partner on project management and partitioning of responsibilities.
- Do top down design, i.e. start with a general block diagram and add details as you progress.
- Define micro-operations for each of the instructions and create a state machine accordingly.
- Iterate on the above two steps until you are certain your design is debugged. Iterate again.
- Code, test, and debug.
- **Think Block Diagram**!