VHDL Design of a RISC Processor: Control Unit

Final Project Proposal

By:
Kyle Wilken
and
Aida Todri

Advisor:
Dr. Vinod Prasad
Bradley University

December 8, 2000
**Project Summary:**
In conjunction with other groups the VHSIC Hardware Description Language (VHDL) will be used to design and implement a functional 8-bit Reduced Instruction Set Computer (RISC) Processor. This group is responsible for constructing an instruction set architecture, determining the structure of the internal registers and memory, and designing the control unit of the processor. The control unit of the processor reads an instruction from memory, generates the appropriate control signals for the rest of the processor, and handles program branches. Our VHDL design and simulation will be done using Mentor Graphics ModelSim software. A XESS Development Board with one of the XILINX XC4000e family of FPGA’s will be used for the implementation. Our group will focus on the Control Unit of the processor. In addition to the design of the processor a basic command-line compiler will be written using the C programming language.

**RISC Overview:**
The concept of a RISC Processor is based upon the idea that a small, basic instruction set in conjunction with a “smart” compiler can deliver superior performance over a Complex Instruction Set Computer (CISC) with a large number of specialized instructions. The simplicity of the operations performed ideally allows every instruction to be completed in one processor cycle.

The basic data path of a RISC processor is shown below in Figure 1. The Instruction Decoder loads the instruction pointed to by the program counter (PC) from processor memory. The Instruction Decoder then generates the appropriate control signals for the Execute unit, which performs the desired function (arithmetic, logic, etc.) on the data. The Writeback unit then updates the memory with any new values.

![Figure 1: RISC Processor Datapath](image-url)
**System Level Block Diagram:**
A more detailed look at the layout of the RISC processor is shown below in Figure 2. The process starts out at the branch selector, which loads the program counter with either the next sequential address or the address of a program branch depending on the value of the branch select signal. In the case of an interrupt, the branch address input would contain the address of the appropriate interrupt handler. The instruction is then fetched from program memory and sent into the instruction decoder, which loads the operand and function select buses and generates control signals for the rest of the processor. The execute stage performs an operation or interacts with the data memory. After the execute stage the result is written to the processor registers if applicable.
**Sub-System Block Diagram:**

The focus of this project is the control unit of the processor, that is the section responsible for the *Fetch* and *Decode* stages. A closer look at the control unit is shown below in Figure 3.

---

**Figure 3: Control Unit Block Diagram**

The inputs and outputs for Figure 3 above are discussed in detail on the following page.
**Instruction Multiplexer Inputs:**
- **BR[0..11] (Branch Address):** Contains the target address (PC + offset) for a branch. In the case of a program branch, this address will point to the appropriate instruction.
- **PC+1 (Incremented Program Counter):** Unless a program branch occurs the next instruction executed is the sequential instruction in program memory.
- **BS (Branch Select):** If this bit is set then the instruction address corresponding to a program branch is selected, otherwise the sequential (PC+1) instruction is selected.

**Instruction Multiplexer Outputs:**
- **W (Instruction Address):** The address of the next location in program memory to be loaded into the program counter.

**Instruction Decoder Inputs:**
- **IR (Instruction Register):** The actual instruction, containing the opcode, target offset in the case of a branch, and possibly some immediate data.
- **PC (Program Counter):** In case of a branch the target offset is added to this value.

**Instruction Decoder Outputs:**
- **MW (Memory Write):** Allows data to be written to the processor registers.
- **RW (Read-after-write):** Allows read of processor register.
- **DA (Data Address):** Address of processor register data to read/write.
- **DS (Data Select):** Select Function output, data output, or target offset.
- **FS (Function Select Bus):** Word which contains the op-code which will tell the ALU what type of operation to perform.
- **A (A Data Bus):** Word which contains data for the A input of ALU.
- **B (B Data Bus):** Word which contains data for the B input of ALU.

**Programmers Model:**
From the programmer’s viewpoint the processor provides a simple direct addressing mode and a relatively large address space for a 8-bit processor. The programmer has access to the following memory:

- 4kb of general program space
- Sixteen 8-bit registers, R0 – R15.
- 256 bytes of “scratch pad” RAM
- 4 external interrupts
**Instruction Set Architecture:**
The performance of a processor depends heavily on the variety and type of instructions a processor can execute. While the RISC architecture inherently limits the number of instructions present, all the basic operations are available. The processor will use a load/store architecture with a direct addressing mode. The four different types of instruction words are shown below in Figure 4.

1. **IMMEDIATE DATA**
   - **OPERATION (IM)**
   - OPCODE DEST DATA
   - 15 12 11 8 7 0

2. **EXTERNAL MEMORY**
   - **OPERATION (EM)**
   - OPCODE SRC/DST ADDRESS
   - 15 12 11 7 0

3. **INTERNAL MEMORY**
   - **OPERATION (R)**
   - OPCODE SEL DEST SRC
   - 15 12 11 7 4 3 0

4. **PROGRAM BRANCH**
   - **OPERATION (BR)**
   - OPCODE OFFSET
   - 15 12 11 0

**Figure 4: Instruction Word Format**
The instructions available to the programmer are shown below in Figure 5.

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>TYPE</th>
<th>OPCODE</th>
<th>SEL</th>
<th>MNEMONIC</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO OPERATION</td>
<td>R</td>
<td>0000</td>
<td>0000</td>
<td>NOP</td>
<td>PC &lt;= PC + 1</td>
</tr>
<tr>
<td>ADD</td>
<td>R</td>
<td>0001</td>
<td>1001</td>
<td>ADD R[x], R[y]</td>
<td>R[x] &lt;= R[x] + R[y]</td>
</tr>
<tr>
<td>ADD IMMEDIATE</td>
<td>IM</td>
<td>0010</td>
<td>XXXX</td>
<td>ADD R[x], #data</td>
<td>R[x] = R[x] + #data</td>
</tr>
<tr>
<td>SUBTRACT</td>
<td>R</td>
<td>0001</td>
<td>0110</td>
<td>SUB R[x], R[y]</td>
<td>R[x] &lt;= R[x] - R[y]</td>
</tr>
<tr>
<td>INCREMENT</td>
<td>R</td>
<td>0011</td>
<td>XXXX</td>
<td>INC Rx</td>
<td>R[x] &lt;= R[x] + 1</td>
</tr>
<tr>
<td>DECREMENT</td>
<td>R</td>
<td>0001</td>
<td>1111</td>
<td>DEC Rx</td>
<td>R[x] &lt;= R[x] - 1</td>
</tr>
<tr>
<td>SHIFT LEFT</td>
<td>R</td>
<td>0100</td>
<td>0000</td>
<td>SHL R[x], R[y]</td>
<td>R[x] = R[x] &lt;&lt; R[y]</td>
</tr>
<tr>
<td>SHIFT RIGHT</td>
<td>R</td>
<td>0100</td>
<td>0001</td>
<td>SHR R[x], R[y]</td>
<td>R[x] = R[x] &gt;&gt; R[y]</td>
</tr>
<tr>
<td>LOGICAL NOT</td>
<td>R</td>
<td>0101</td>
<td>0000</td>
<td>NOT Rx</td>
<td>R[x] &lt;= NOT R[x]</td>
</tr>
<tr>
<td>LOGICAL NOR</td>
<td>R</td>
<td>0101</td>
<td>0001</td>
<td>NOR R[x], R[y]</td>
<td>R[x] &lt;= R[x] NOR R[x]</td>
</tr>
<tr>
<td>CLEAR</td>
<td>R</td>
<td>0101</td>
<td>0011</td>
<td>CLR R[x]</td>
<td>R[x] &lt;= '0'</td>
</tr>
<tr>
<td>LOGICAL NAND</td>
<td>R</td>
<td>0101</td>
<td>0100</td>
<td>NAND R[x], R[y]</td>
<td>R[x] &lt;= R[x] NAND R[y]</td>
</tr>
<tr>
<td>LOGICAL XOR</td>
<td>R</td>
<td>0101</td>
<td>0110</td>
<td>XOR R[x], R[y]</td>
<td>R[x] &lt;= R[x] XOR R[y]</td>
</tr>
<tr>
<td>LOGICAL AND</td>
<td>R</td>
<td>0101</td>
<td>1011</td>
<td>AND R[x], R[y]</td>
<td>R[x] &lt;= R[x] AND R[y]</td>
</tr>
<tr>
<td>LOGICAL OR</td>
<td>R</td>
<td>0101</td>
<td>1011</td>
<td>OR Rx, Ry</td>
<td>R[x] &lt;= R[x] OR R[y]</td>
</tr>
<tr>
<td>LOGICAL NAND</td>
<td>R</td>
<td>0101</td>
<td>1100</td>
<td>SET R[x]</td>
<td>R[x] &lt;= '1'</td>
</tr>
<tr>
<td>LOAD INDIRECT</td>
<td>EM</td>
<td>0110</td>
<td>XXXX</td>
<td>LD R[x], R[y]</td>
<td>R[x] &lt;= MEM[R[y]]</td>
</tr>
<tr>
<td>STORE INDIRECT</td>
<td>EM</td>
<td>0111</td>
<td>XXXX</td>
<td>STR R[y], R[x]</td>
<td>MEM[R[y]] &lt;= R[x]</td>
</tr>
<tr>
<td>LOAD REGISTER</td>
<td>EM</td>
<td>1000</td>
<td>XXXX</td>
<td>LD Rx, #address</td>
<td>R[x] &lt;= MEM[#address]</td>
</tr>
<tr>
<td>STORE REGISTER</td>
<td>EM</td>
<td>1001</td>
<td>XXXX</td>
<td>STR #address, Rx</td>
<td>MEM[#address] &lt;= R[x]</td>
</tr>
<tr>
<td>JUMP</td>
<td>BR</td>
<td>1010</td>
<td>0000</td>
<td>Jmp #address</td>
<td>#address -&gt; PC</td>
</tr>
<tr>
<td>BRANCH IF ZERO</td>
<td>BR</td>
<td>1010</td>
<td>0001</td>
<td>JZ #offset</td>
<td>IF R[x] == 0 THEN PC &lt;= PC + offset</td>
</tr>
<tr>
<td>BRANCH IF NOT ZERO</td>
<td>BR</td>
<td>1010</td>
<td>0011</td>
<td>JNZ #offset</td>
<td>IF R[x] &lt;&gt; 0 THEN PC &lt;= PC + #offset</td>
</tr>
<tr>
<td>SET IF LESS THAN</td>
<td>R</td>
<td>1011</td>
<td>XXXX</td>
<td>SLT R[x], R[y]</td>
<td>IF R[x] &lt; R[y] THEN R[x] = 1</td>
</tr>
<tr>
<td>ENABLE INTERRUPTS</td>
<td>R</td>
<td>1100</td>
<td>XXXX</td>
<td>EL intcode</td>
<td>Lowest four bits enable IR3 - IR0</td>
</tr>
<tr>
<td>RETURN FROM INTERRUPT</td>
<td>BR</td>
<td>1101</td>
<td>XXXX</td>
<td>RETI</td>
<td>PC &lt;= PC' + 1</td>
</tr>
<tr>
<td>MOVE</td>
<td>R</td>
<td>1110</td>
<td>XXXX</td>
<td>MOV R[x], R[y]</td>
<td>R[y] &lt;= R[x]</td>
</tr>
</tbody>
</table>

**Figure 5: Instruction Set**

A notable absence from the instruction set above is the *Jump on Carry* instruction. Since the processor has no carry register this instruction is not included. Instead, the *Set if Less Than* is included which sets Register X if the value in Register X is less than that in Register Y.
Time Table:

- **January**
  - Gate level design of instruction multiplexer
  - Gate level design of instruction decoder

- **February**
  - VHDL coding
  - Interfacing with other processor groups

- **March**
  - Timing analysis and simulation of VHDL model
  - Implementation and testing on FPGA board

- **April**
  - Debugging on FPGA board
  - Write compiler in C
  - Prepare for research expo

Equipment List:

**Software**: Mentor Graphics ModelSim, C

**Hardware**: XESS FPGA Evaluation Board