CS2211: STUMP – A Mini-Risc Processor

Introduction

In this experiment, you will:

- Design the datapath for a simple RISC processor. A set of building blocks has been provided as library components; you merely need to connect them together!
- Design and build the control logic for the processor.
- Simulate the complete processor using Viewsim command files.
- Write a small test program using the assembler provided.
- Download the microprocessor to the Xilinx board and demonstrate the program running.

You must first create yourself the correct Powerview environment; run the script ***mk_pv 221***. This will create a CS221 project in your powerview directory and insert a symbol for the processor in the *sym* subdirectory.

If you have forgotten how to use Powerview, reread the introductory powerview manuals available in the laboratory. In the 1st year, simulation was carried out in a rather ad-hoc interactive manner. A more systematic approach is required for this design and the marking scheme for the experiment reflects this. It is therefore important that you know how to drive Viewsim and in particular how to write Viewsim command files. A quick guide is given in Appendix H, “Using Viewsim,” on page 26.

Specification of the Processor

Architecture

The processor is a 16-bit machine with a RISC style architecture. Operands for ALU operations come from registers inside the processor and the result is returned to a register. Separate instructions are provided to move data between the registers and external memory.

There are 8 registers. R0 is always zero and can be used as a source operand allowing *Move* instructions to be synthesised from an *Add* instruction. R0 may be written to, but the result is always discarded allowing *Compare* instructions to be synthesised from *Subtract* instructions. Register R7 is the program counter and, from a programmer’s view of the machine, has equal status with the other registers allowing PC-relative addressing to be supported.

The processor has a 4-bit condition code register shown in Table 1. Each arithmetic and logical

CC 3	CC 2	CC 1	CC 0
Sign Flag (N)	Zero Flag (Z)	Overflow Flag (V)	Carry-Out Flag (C)

Table 1: Condition Code Bits

instruction has two variants; in one case the condition code register is updated to reflect the result of the ALU operation, in the second case the register is not updated. Load, Store and Branch instructions never change the condition code register.

There are 3 instruction formats in a fixed-length 16-bit instruction. The machine operates on 16-bit words only. Byte addressing is not supported.

Instruction Set

There are 8 basic instructions shown in Table 2. Some other instructions such as *Cmp*, *Nop* and

Instruction Code	Instruction	Explanation
000	ADD	2’s complement add
001	ADC	2’s complement add with carry-in
010	SUB	2’s complement subtract
011	SBC	2’s complement subtract with borrow
100	AND	Bitwise AND of two 16-bit words
101	OR	Bitwise OR of two 16-bit words
110	LD/ST	Load register from memory ***or*** Store register to memory
111	B*cc*	Branch if condition *cc* is satisfied.

Table 2: Basic Instruction Set

Mov can be expressed directly in terms of these basic instructions and are supported by the assembler. Other instructions may be synthesised from the combinations of the basic instruction set as shown in Appendix F, “Extending the Instruction Set,” on page 25.

Shift instructions are somewhat special. Single-bit *shift-left* instructions can be derived from the basic instruction set. *Shift-right* instructions have been added as a rather ugly “kludge” and are dealt with in next section.

Instruction Formats

There are just 3 instruction formats which are shown below:

Type 1: 2 Source Registers

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="3">INSTR</td><td>0</td><td>LD
CC</td><td colspan="3">DST</td><td colspan="3">SRC A</td><td colspan="3">SRC B</td><td colspan="2">SHIFT</td></tr>
</table>

Type 2: 1 Source Register, 1 Immediate Value

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="3">INSTR</td><td>1</td><td>LD
CC</td><td colspan="3">DST</td><td colspan="3">SRC A</td><td colspan="5">Immediate</td></tr>
</table>

Type 3: Conditional Branch

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Condition				Offset							

The processor is a 3-address machine. In the case of arithmetic and logical instructions (instruction codes 0 … 5), the two source operands are either two registers (Type-1 instructions) or a register and a 5 bit *signed* immediate value (Type-2 instructions). The result of the operation is returned to the destination register (DST) and the condition codes are updated depending on the state of bit 11 (LDCC set - update condition-codes; LDCC clear - do not update condition codes)

In the case of a LD/ST instruction, bit 11 determines the direction of data transfer: if LDCC is set, the operation is *store to memory*; if LDCC is clear, the operation is *load from memory*. The memory address is constructed from the sum of the two source operands, i.e. the two registers specified by SRC A and SRC B for Type-1 instructions or the register specified by SRC A and the 5-bit signed immediate for Type-2 instructions. The register specified by DST is the register to be stored in memory for a Store operation or the register to be loaded from memory for a Load operation. LD/ST operations do not affect the condition code bits.

Bits 1 and 0 in Type-1 instructions are used to control various shift-right operations. Operand A is shifted *before it reaches* the ALU as shown in Table 3:

Operation	Instr Bit 1	Instr Bit 0	SRC A, bit 15:=	Carry-out:=
No shift	0	0	A15	0
ASR	0	1	A15	A0
ROR	1	0	A0	A0
RRC	1	1	C_{in}	A0

Table 3: Shift Operations

Type-3 instructions implement 16 conditional branch instructions shown in Table 4. The range of the branch target address is PC + 1 + 8 bit signed offset.

Experiment

Paper Design

You are required to implement a non-pipelined version of the STUMP processor described above. Each instruction should divided into 3 phases, each phase taking a single clock cycle to execute as shown in Table 5.

Produce a paper design of the datapaths required. All of the building blocks (apart from a control block) that you require are shown in Appendix A, “Datapath Blocks,” on page 13. Do

Mnemonic	Bits 11:8	Condition	
BAL	0000	Always	
BNV	0001	Never	
BHI	0010	C+Z=0	comparison: unsigned arithmetic
BLS	0011	C+Z=1	
BCC	0100	C=0	overflow test: unsigned arithmetic
BCS	0101	C=1	
BNE	0110	Z=0	zero test
BEQ	0111	Z=1	
BVC	1000	V=0	overflow test: signed arithmetic
BVS	1001	V=1	
BPL	1010	N=0	comparison: signed arrithmetic
BMI	1011	N=1	
BGE	1100	$\overline{N}.V+N.\overline{V}=0$	
BLT	1101	$\overline{N}.V+N.\overline{V}=1$	
BGT	1110	$(\overline{N}.V+N.\overline{V})+Z=0$	
BLE	1111	$(\overline{N}.V+N.\overline{V})+Z=1$	

Table 4: Conditional Branch Instructions

Phase 1	Fetch	The next instruction is fetched from memory
Phase 2	Decode/ Execute	The instruction is decoded and executed
Phase 3	Write	ALU operations are written back into the register bank Load/Store operations access memory

Table 5: Instruction Phases

the paper design *before* coming to the laboratory and get it *checked* before entering it as a Viewdraw schematic

Hints

Knowing where to begin can be difficult. Start with the register bank and the ALU. Consider how they are connected and how they interface to the "outside" via an address bus and a data bus.

Whenever data, instructions or condition codes etc. need to be preserved from one phase to the next, a register has to be inserted. Whenever a datapath can be driven from more than one source, a multiplexer must be inserted.