**Chapter 7, Threads and Interprocess Communication homework 3**

The purpose of this homework is to begin to develop a hierarchical testbench and practice using synchronized mailboxes to pass transactions. The device under test is a RISC processor, called the RISC\_SPM.

Design a hierarchical testbench for the RISC\_SPM design as depicted in Figure 1. For this homework only design the Generator, Agent, and Driver and connect the DUT (i.e. the RISC\_SPM). The Generator is essentially playing the part of memory, providing instructions and memory fetches. The Agent does nothing at this time except pass a transaction from the Generator to the Driver. The Driver converts the transaction to the pin level interface of the DUT. For the case of instruction fetches it will assemble the instruction. For memory fetches it will provide a random numerical value. The Generator should be capable of creating the 2 branch instructions (BR and BRZ) and the HALT instruction but for now constrain the instructions so no branch instructions or halt instructions are generated. The RISC\_SMP design will be provided to you. More details of the RISC\_SPM are in the appendix.



Figure 1: Hierarchical Testbench

The minimum requirements are:

1. Create a separate class to encapsulate the Generator, Agent, and Driver.
2. Use mailboxes to pass transactions between the Generator, Agent, and Driver.
3. The Generator and Agent operate only at the Transaction level.
4. The Generator randomly creates 10 random transactions.
5. The mailboxes must be synchronized as described in sections 7.6.4-7.6.7 of the book.

Deliverables:

1. All of your code
2. Waveform clearly showing the 10 transactions. The waveform should show at a minimum
   1. The memory bus
   2. ASCII version of the presently executing instructions.
   3. Clock and reset
   4. The 4 registers, R0\_out, R1\_out, R2\_out, and R3\_out. These signals are in the Processor block.

**Appendix: RISC\_SPM Description**

The RISC\_SPM’s instruction set is in Table 1, short instruction format in Table 2, long instruction format in Table 3, and the architecture in Figure 2. Note that some instructions require more than 1 memory access to fetch the instruction. For example, a RD instruction will require 1 memory access to fetch the first byte of instruction, a 2nd memory access to fetch the 2nd byte of the instruction indicating the memory address to read from, and a 3rd memory access to fetch the data specified by this address. The data paths are all 8-bits wide.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction** | **Instruction Word** | | | **Action** | **Num memory accesses** |
| **opcode** | **src** | **destination** |
| NOP | 0000 | N/A | N/A | none | 1 |
| ADD | 0001 | src | dest | dest <= src + dest | 1 |
| SUB | 0010 | src | dest | dest <= dest - src | 1 |
| AND | 0011 | src | dest | dest <= src && dest | 1 |
| NOT | 0100 | src | dest | dest <= ~src | 1 |
| RD | 0101 | N/A | dest | dest <= memory[Add\_R] | 3 |
| WR | 0110 | src | N/A | memory[Add\_R] <= src | 3 |
| BR | 0111 | N/A | N/A | PC <= memory[Add\_R] | 3 |
| BRZ | 1000 | N/A | N/A | PC <= memory[Add\_R] | 3 |
| RDI | 1001 | N/A | dest | dest<=memory[PC] | 2 |
| HALT | 1111 | N/A | N/A | Halts execution until reset | 1 |

Table 1: RISC\_SPM Instruction Set

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **opcode** | | | | **source** | | **destination** | |
| opcode[3] | opcode[2] | opcode[1] | opcode[0] | source[1] | source[0] | desti [1] | dest [0] |

Table 2: Short (1-byte) instructions – NOP/ADD/SUB/AND/NOT/HALT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **opcode** | | | | **source** | | **destination** | |
| opcode[3] | opcode[2] | opcode[1] | opcode[0] | source[1] | source[0] | desti [1] | dest [0] |
| **address** | | | | | | | |
| address[7] | address[6] | address[5] | address[4] | address[3] | address[2] | address[1] | address[0] |

Table 3: Long (2-byte) instructions – RD/WR/BR/BRZ/RDI



Figure 2: RISC\_SPM Architecture

Additional opcode description:

* NOP – No operation performed
* ADD – Add contents of source and destination registers. Store results in destination register.
* AND – Form bit-wise AND of source and destination registers. Store results in destination register.
* NOT – Form bit-wise complement of source register Store results in destination register.
* SUB– Subtract contents of source from destination registers. Store results in destination register.
* RD – Fetches a memory word from the location specified by the second byte and loads the result into the destination register. The source register bits are unused
* RDI – Loads the second byte of the instruction into the destination register. The source register bits are unused
* WR – Writes the contents of the source register to the word in memory specified by the address held in the second byte. The destination register bits are don't-cares, i.e. unused.
* BR – Branches the activity flow by loading the program counter with the word at the location (address) specified by the second byte of the instruction. The source and destination bits are don't-cares, i.e. unused.
* BRZ – Branches the activity flow by loading the program counter with the word at the location (address) specified by the second byte of the instruction if the zero flag register is asserted.
* HALT – Halts the processor