Computer Sciences Department
University of Wisconsin-Madison
CS/ECE 552 – Introduction to Computer Architecture
WISC-SP20 Microarchitecture Specification

In this document, we describe the microarchitecture, including register file specifications, memory system organization, etc. you will use for your CS/ECE 552 project. The WISC-SP20 architecture that you will design for the final project shares many resemblances to the MIPS R2000 described in the text. The major differences are a smaller instruction set and 16-bit words for the WISC-SP20. Similarities include a load/store architecture and three fixed-length instruction formats.

1. Registers

There are eight user registers, R_0-R_7. Unlike the MIPS R2000, R_0 is *not* always zero. Register R_7 is used as the link register for JAL or JALR instructions. The program counter is separate from the user register file. A special register named EPC is used to save the current PC upon an exception or interrupt invocation.

2. Memory System

The WISC-SP20 is a Harvard architecture, meaning instructions and data are located in different physical memories. It is byte-addressable, word aligned (where a word is 16 bits long – note that this is different from some of the examples in class), and big-endian. The final version of the WISC-SP20 will include a multi-cycle memory and one level of cache. However, initial versions of the machine will contain a single cycle memory. See the project deadlines for more details.

The WISC-SP20 cache replacement policy is deterministic. See the cache module description for an outline of the algorithm you must use.

NOTE: For phase1 and phase2, you will work with a simplified memory model which supports un-aligned accesses.

3. Pipeline

The final version of the WISC-SP20 contains a five-stage pipeline identical to the MIPS R2000. The stages are:

1. Instruction Fetch (IF)
2. Instruction Decode/Register Fetch (ID)
3. Execute/Address Calculation (EX)
4. Memory Access (MEM)
5. Write Back (WB)

See Figure 4.35 on page 289 and Figure 4.36 on page 291 of the text for good starting points.

4. Optimizations

Your goal in optimizations is to reduce the CPI of the processor or the total cycles taken to execute a program. While the primary concern of the WISC-SP20 is correct functionality, the architecture must still have a reasonable clock period. Therefore, you may not have more than one of the following in series during any stage:

- register file
- memory or cache
- 16-bit full adder
- barrel shifter

You may implement any type of optimization to reduce the CPI (as long as it's a valid optimization). The required optimizations are:

- Register file bypassing
- There are two register forwarding paths in the WISC-SP20:
    - Forwarding from beginning of the MEM stage to beginning of EX stage (EX → EX forwarding)
    - Forwarding from beginning of the WB stage to the beginning of the EX stage (MEM → EX forwarding)
- All branches should be predicted not-taken. This means that the pipeline should continue to execute sequentially until the branch resolves, and then squash instructions after the branch if the branch was actually taken.

5. Exceptions: extra credit

Exception handling is extra credit. If you choose not to implement exception handling, an illegal instruction should be treated as a NOP.

`IllegalOp` is the only defined exception in the WISC-SP20 architecture. It is invoked when the opcode of the currently executing instruction is not a recognized member of the ISA. Upon finding an illegal opcode, the computer shall save the current PC into the reserved register EPC and then load address 0x02, which is the location of the IllegalOp exception handler. Note that if you choose to implement exceptions, address 0x00 must be a jump to the start of the main program.

The exception handler itself need not be complex. At a minimum it should load the value 0xBADD into R_7 and then call the RTI instruction.