NANYANG TECHNOLOGICAL UNIVERSITY

CZ3001

PROJECT

ASSIGNED: XX/XX/2013, DUE XX/XX/2013, 6:30PM

I. PUTTING IT TOGETHER: PHASE 1 – SIMPLE CPU

In phase 1 of the project, we will now use the ALU and RF, and modify our control unit from Assignment 3 in order to create a simple 16-bit CPU with a load/store architecture.

Your CPU has a register file, a 3-bit FLAG register, and sixteen instructions. The register file has sixteen 16-bit registers, with R0 hardwired to 0x0000 (as in Assignment 2). Register R15, now has a special purpose; it is used as a Return Address register for the JAL instruction. The FLAG register has 3 bits: Zero (Z), Overflow (V), and Sign bit (N)

We can classify our CPU's instructions into four major classes: Arithmetic, Memory, Load Immediate, and Control.

A. Arithmetic Instructions

There are eight arithmetic and logical instructions: ADD, SUB, AND, OR, SLL, SRL, SRA, and RL. The ADD, SUB, AND and OR instructions have a three address format, with assembly-level syntax:

opcode Rd, Rs, Rt

Rs and Rt are source registers and Rd is the destination register.

The SLL, SRL, SRA, and RL instructions have a two address and one immediate format, with assembly-level syntax

opcode Rd, Rs, imm

Rd is the destination register, Rs is the source register and imm is used as the shift amount for ALU.

Only the Arithmetic instructions modify the state of the FLAG registers; state is modified as follows:

The Z flag is set if and only if the output of the operation is zero.

The V flag is set by the ADD and SUB instructions if and only if the operation results in an overflow (addition of two positive numbers produces negative result or subtraction of two negative numbers produces a positive result). The AND and OR instructions always clear the V flag.

The N flag is set if and only if the result of the ADD and SUB instruction is negative. Note that the N flag **should not** be set if the value is negative due to overflow (e.g. N flag is set if and only if the result is **validly** (i.e. not due to overflow) negative).

The SLL, SRL, SRA, and RL instructions do not modify the FLAG register.

B. Memory Instructions

There are two memory instructions: LW and SW. The assembly level syntax is:

opcode Rd, Rs, offset

And the bit level format is:

1 0 0 op	Rd	Rs	offset	
15	12 11	8 7	4 3	0

For the LW instruction, the op bit is ‘0’, Rd is the destination register, and memory address is register Rs + offset. For the SW instruction, the op bit is ‘1’, the data in Register Rd will be stored to the memory address (register Rs + offset).

C. Load Immediate Instruction

There are two instructions: LHB and LLB, with assembly-level syntax:

opcode Rd, imm

And the bit-level format is:

1 0 1 op	Rd	immediate	
15	12 11	8 7	0

The LHB instruction loads the most significant 8 bits of register Rd with the bits in the immediate field, and the least significant bits are left unchanged.

The LLB instruction loads register Rd with the sign-extended 8-bits immediate field.

D. Control Instructions

There are four instructions: B, JAL, JR, and EXEC.

The assembly level syntax for the B (branch) instruction and bit-level format are:

B cond, offset

opcode	0 cond	offset	
15	12 11	8 7	0

The B (branch) instruction conditionally jumps to the address obtained by adding the 8-bit (two’s complement format signed) offset to the contents of the program counter. Assume that the value of the program counter used in this addition is the address of the next instruction (i.e. the address of the B instruction + 1). There are eight possible conditions: Equal (EQ), Not Equal (NE), Greater Than (GT), Less Than (LT), Greater or Equal (GEQ), Less or Equal (LEQ), Overflow (O), and True (T). Many of these conditions are determined based on the 3-bit flag N, V, Z, which should be set by an ADD, SUB, AND or OR instruction executed prior to the conditional branch instruction. The True condition corresponds to an unconditional branch.

Table - Encoding for Condition Codes

Condition Code	**Condition**
000	Equal (Z=1)
001	Not Equal (Z=0)
010	Greater Than (Z=N=0)
011	Less Than (N=1)
100	Greater or Equal (Z=1 or Z=N=0)
101	Less or Equal (N=1 or Z=1)
110	Overflow (V=1)
111	True

The JAL (jump and link) instruction saves the contents of the program counter (address of the JAL instruction + 1) to the Return Address register (R15) and jumps to the procedure whose starting address is (partly) specified in the instruction. The assembly-level syntax and bit level encoding for this instruction is:

JAL target

opcode	target
15 12	11 0

The target field is a 12-bit two's complement signed immediate offset that is added to the current PC (which is PC+1 relative to the JAL instruction) to form the target jump address.

The JR (jump register) instruction jumps to the address specified by Rd. When Rd == R15, JR is used as a return from procedure instruction. The assembly-level syntax and bit-level encoding for the JR instruction is:

JR Rd

opcode	Rd	Don't care
15 12	11 8	7 0

The EXEC (execute) instruction jumps to the address specified by Rd, executes a single instruction from that location in memory, then immediately resumes execution at the instruction following the EXEC instruction (PC+1). If the instruction at the target changes the PC (i.e. it is also a control instruction), the change is ignored. However, if it is any other instruction type it must execute correctly. The assembly-level syntax and bit-level encoding for the EXEC instruction is:

EXEC Rd

opcode	Rd	Don't care
15 12	11 8	7 0

A full listing of the instructions and opcode encodings is given in

Instruction Name	Opcode
ADD	0000
SUB	0001
AND	0010
OR	0011
SLL	0100
SRL	0101
SRA	0110
RL	0111
LW	1000
SW	1001
LHB	1010
LLB	1011
B	1100
JAL	1101
JR	1110
EXEC	1111

E. The Memory System

The address space of your processor is 16-bits; each memory location is 16-bits in size (word addressable), so the total memory capacity is 2^{16}*16 bits = 128Kbytes. In the first phase of the project, you will have separate instruction and data memories. The instruction memory has a 16-bit address input and a 16-bit data output with no control signals. The data memory has a 16-bit address input, a 16-bit data input, a 16-bit data output, and an active-low write signal for specifying a write operation. If the write signal is low, the memory with write the input data bits to the specified address. The read operation is always active for both instruction and data memories; the access time for both memories is one cycle. The Verilog modules for the instruction and data memories will be provided to you.

F. Implementation

1. <u>Pipelining</u>

Your design must use a five stage pipeline (IF, ID, EX, MEM, WB) similar to the textbook. You should use (and will need to modify) your ALU, RF, and control implementations from Assignments 1-3. You must implement hazard detection so that your pipeline correctly handles all data and control dependences.

2. <u>Reset</u>

Your CPU has a Reset input. Instructions are executed when the Reset signal is low. If the Reset input goes high for one clock cycle (synchronous reset), the contents of the program counter and register file are cleared to zero.

3. Grading

For phase one, you should prepare a written report containing:

1. A brief description & introduction to the project – describe special features, any particular effort you made to optimize the design and any major problems encountered
2. A statement indicating whether you have successfully fulfilled all requirements for phase 1.
3. Verilog print-outs of your major blocks
4. Simulation results for test programs
5. Detailed description of the method you used to test the correctness of your design, including annotated simulations of key programs used to create test inputs and verify outputs of your blocks. A significant part of your grade will be determined by testing methodology – poor methodology will not receive full marks regardless of output correctness.

II. Phase Two – The Memory System

Now in phase two, you will replace the individual instruction and data memories, with a single shared memory, which has a latency of 3 cycles for read and write. Both caches are direct-mapped, and have a block size of 8 words (16 Bytes), and a data array size of 512 words (1024 Bytes organized as 64 lines). Therefore, address[2:0] is the block offset, address[8:3] is the index, and the remaining bits are tag bits. Besides the tab bits, the tag array of your cache should have one valid bit per entry indicating if the cache line is empty (0-invalid, 1-valid). A block diagram of your new memory system is shown in Figure .

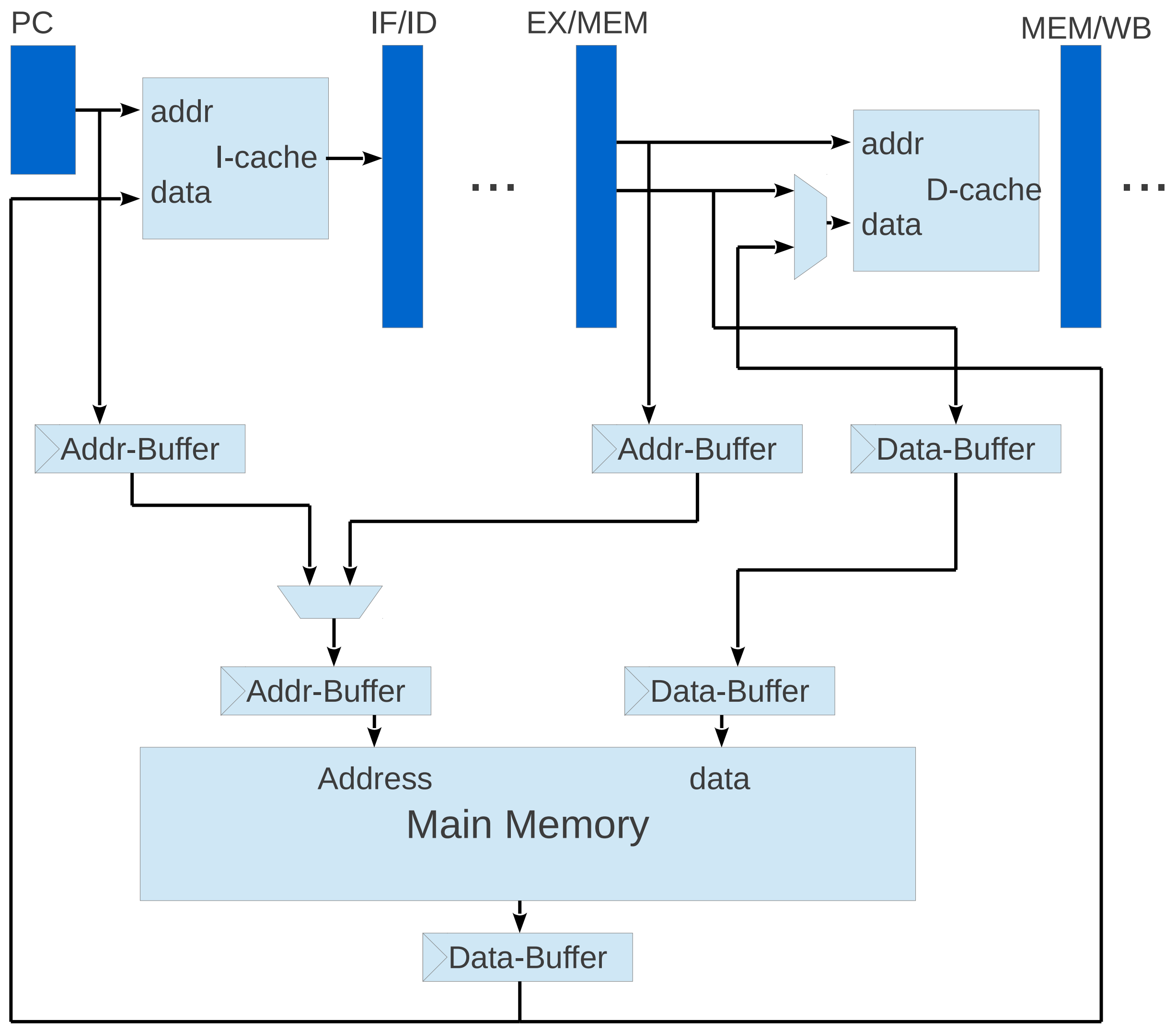

Figure - Memory System

From the figure we can see that the I-cache and D-cache share one main memory. For the instruction cache, when there is a fetch hit in the I-cache, the pipeline will not be stalled. When there is a fetch miss in the I-cache, the address is stored in the address buffer of the main memory after two cycles. The address buffer should hold the address for three cycles until the main memory fetches the instructions out and stores them in its data-buffer. After that, the data

buffer will write the instructions to the I-cache. You should generate the write-enable signal for the I-cache and update the tag array appropriately.
Since your I-cache has an 8-word line, you must fill the entire line from the memory to the cache. Therefore, the read-miss delay is 8 times of the delay for reading one word. You can assume that the I-cache is never written by the processor; the I-cache is only written to from fetch misses.

For the data cache, the read operation is similar to the I-cache. For writes, you will implement a write-through no-allocate policy. When there is a write instruction (SW) in your pipeline, your cache needs to check if it hits in the cache. If it is a write hit, the data-cache will write the word to the cache and also to the main memory. If it is a write hit, the processor also writes to the cache and the memory, but the cache line is marked as invalid (because the data was written to a line containing data from a different address).

Because you only have one 16-bit write bus to the memory and each write takes multiple cycles, you cannot execute SW instructions back-to-back. Therefore, you need a hazard detection unit that separates SWs by stalling (inserting NOPs into your pipeline). This is a structural hazard detection unit. Figure shows the timeline of correct operation when two SW instructions are back-to-back.

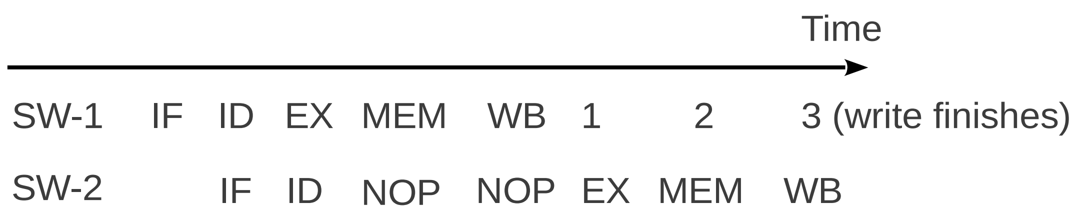
Time
SW-1 IF ID EX MEM WB 1 2 3 (write finishes)
SW-2 IF ID NOP NOP EX MEM WB

Figure - Hazard detection for SW instructions

Since the I-cache and D-cache share main memory, they cannot access the memory concurrently. You need to implement selection logic to pick one that accesses the memory. You can either give priority to one cache over another, or implement a locking scheme that guarantees mutual exclusion. Verilog modules will be provided to you for all the memories and caches, but you will need to modify the control and hazard detection abilities of your CPU to correctly handle the operation.

III. Phase 2A (Optional)

You will get bonus points if you implement extra features to enhance your CPU. As a general guideline, a very simple addition may add 1% to your project grade. It is unlikely that any combination of advanced features will cumulatively exceed a 25% increase to your project grade. Given the project weight on your course grade, this means that your final grade will improve by 0.2 to 5 marks overall, which may boost you between one grade and the next, but is unlikely to boost you by more than one grade. In order to qualify for any extra features, you must be able to demonstrate a fully-functional project according to the original specifications. Before beginning any extra features, save a working copy of your project. In addition, any extra features must satisfy the following requirements:

- The feature must enhance the performance of the CPU; do not implement features that make the performance worse.
- The feature must not break the original design (unless you will demonstrate both designs)
- The feature must be non-trivial – difficulty of the feature will determine the points received.

Before beginning any extra features discuss with the instructors.

IV. GRADING

The project should be performed in groups of three. Your report for the final project phase should include:

1. A brief description & introduction to the project – describe special features, any particular effort you made to optimize the design and any major problems encountered, emphasizing the differences between your phase 1 and phase 2 designs.
2. A statement indicating whether you have successfully fulfilled all requirements for phase 2.
3. Verilog print-outs of your major blocks, and top level diagram (hand drawn ok) of your CPU
4. Simulation results for test programs, including sufficient signals to see instructions from I-cache/memory, ALU operands results and control signals, Register file read, write and control signals, data written back at the WB stage. You should highlight the signals and annotate the simulations to demonstrate correctness.
5. Detailed description of the method you used to test the correctness of your design, including annotated simulations of key programs used to create test inputs and verify outputs of your blocks. A significant part of your grade will be determined by testing methodology – poor methodology will not receive full marks regardless of output correctness.
6. If your project implements any extra features, provide a description highlighting those features and estimate the performance enhancement due to these extra features for the testing program. Bonus points will not be considered if the design is incorrect.

In addition, each group will be asked to demonstrate their project. During the demonstration, all team members must be present and prepared to answer detailed questions about the design. During the demo, you will show the overall design, simulate a few test programs (including ones provided in advance and extra programs provided on the day of the demonstration), answer questions that instructors may have and demonstrate any extra features in your design.