**ECE/CS 552: INTRODUCTION TO COMPUTER ARCHITECTURE**
(Spring 2015)

PROJECT DESCRIPTION
**Due Dates**: First Phase on April 9, 2015 and Second Phase on May 5, 2015

# 1 The Architecture

WISC-S15 is a simple, but powerful, 16-bit computer with a load/store architecture. Design and implement this architecture using **synthesizable** Verilog code. **Work in groups of two.** By February 26, 2015, please upload the names and email addresses of your group members to the website.

WISC-S15 has a register file, a 3-bit FLAG register, and fourteen powerful instructions. The register file is comprised of sixteen 16-bit registers. Registers $14 and $15 have a special purpose. Register $14 serves as a Data Segment (DS) register for load and store instructions while $15 serves as a Stack Pointer (SP) for Call and Return instructions. The FLAG register contains three bits: Zero (Z), Overflow (V), and Sign bit (N). All its addresses are word addresses (1 word = 2 bytes).

WISC-S15's instructions can be categorized into three major classes: Arithmetic, Load/Store, and Control.

## 1.1 Arithmetic Instructions

Eight arithmetic and logical instructions belong to this category. They are ADD, SUB, NAND, XOR, INC, SRA, SRL, and SLL.

The ADD, SUB, NAND, and XOR instructions have a three address format. The assembly level syntax for these instructions is

Opcode rd, rs, rt

The two operands are (rs) and (rt) and the destination is register rd. The ADD and SUB instructions respectively add and subtract the two operands in twos-complement representation and save result in register rd. The NAND and XOR instructions respectively perform bitwise-NAND, and bitwise-XOR operation on the two operands and save the result in register rd. The ADD, SUB, NAND, and XOR instructions also set or clear the Zero (Z), Overflow (V), and Sign (N) bits in the FLAG register. 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. The NAND and XOR instructions clear the V flag. The N flag is set if and only if the result of the ADD and SUB instruction is negative. The NAND and XOR instructions clear the N flag.

The INC, SRA, SRL, and SLL instructions have the following assembly level syntax.

Opcode rd, rs, imm

The imm is a 4-bit immediate operand in twos-complement representation for the INC instruction and unsigned representation for the SRA, SRL, and SLL instructions. The INC instruction adds sign-extended imm field to (rs) and saves the result in rd. The INC instruction sets the Z, V, and N flags. SRA, SRL, and SLL shift (rs) by number of bits specified in the imm field and saves the result in register rd. SRA is shift right arithmetic, SRL is shift right logical, and SLL is shift left logical. The SRA, SRL, and SLL instructions leave the flags unchanged.

The machine level encoding for the eight arithmetic instruction is

0aaa dddd ssss tttt

where aaa represents the opcode (see Table 2), dddd and ssss respectively represent the rd and rs registers. The tttt field represents either the rt register or the imm field.

### 1.2 Load/store instructions

There are four instructions which belong to this category: LW, SW, LHB, and LLB. The assembly level syntax for the LW and SW instructions is

Opcode rt, offset

The LW instruction loads register rt with contents the location specified by offset. The offset is sign-extended and added to the contents of Data Segment (DS) register to compute the address of the memory location to load. The SW instruction saves (rt) to the location specified by the offset. The address of the memory location is computed as in LW.

The machine level encoding of these two instructions is

10aa tttt oooo oooo

where aa specifies the opcode, tttt identifies rt and oooo oooo is the offset in twos-complement representation.

LHB instruction loads the most significant 8 bits of register rt with the bits in the immediate field. The least significant 8 bits of the register rt are left unchanged. Similarly, the LLB instruction loads the least significant 8 bits of register rt with the bits in the immediate field. The most significant 8 bits of register rt are left unchanged. The assembly level syntax for LHB and LLB instructions is

Opcode rt, immediate

The machine level encoding for this instruction is

| ccc | Condition |
|---|---|
| 000 | Equal (Z = 1) |
| 001 | Less Than (N = 1 and V = 0) |
| 010 | Greater Than (Z = N = V = 0) |
| 011 | Overflow (V = 1) |
| 100 | Not Equal (Z = 0) |
| 101 | Greater or Equal (Complement of Less Than) |
| 110 | Less or Equal ((N = 1 and V = 0) or Z = 1) |
| 111 | True |

Table 1: Encoding for Branch conditions

10aa tttt uuuu uuuu

where aa, tttt, and uuuuuuuu respectively specify the opcode, register rt and the 8-bit immediate value.

### 1.3 Control Instructions

There are three instructions which belong to this category: Branch, Call, and Return.

The (delayed) Branch instruction conditionally jumps to the address obtained by adding the 8-bit immediate (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 second next instruction (i.e., address of Branch instruction + 2). Also, this is delayed branch instruction. Therefore, irrespective of the branch condition, the instruction immediate following the branch instruction (i.e., instruction at address of Branch instruction + 1) will always be executed. The eight possible conditions are Equal (EQ), Less Than (LT), Greater Than (GT), Not Equal (NE), Greater or Equal (GEQ), Less or Equal, Overflow, and True. The True condition corresponds to an unconditional branch. The status of the condition is obtained from the FLAG register. The assembly level syntax for this instruction is

B cond, offset

The machine level encoding for this instruction is

Opcode xccc iiii iiii

where x stands for don't care, ccc specifies the condition as in Table 1 and iiiiiiii represents the 8-bit signed offset in twos-complement representation.

The Call instruction saves the contents of the program counter (address of the Call instruction + 1) on top of stack and jumps to the procedure whose start address is partly specified in the instruction. After saving the program counter, the stack pointer (i.e., register

| Function | Opcode |
|---|---|
| ADD | 0000 |
| SUB | 0001 |
| NAND | 0010 |
| XOR | 0011 |
| INC | 0100 |
| SRA | 0101 |
| SRL | 0110 |
| SLL | 0111 |
| LW | 1000 |
| SW | 1001 |
| LHB | 1010 |
| LLB | 1011 |
| B | 1100 |
| CALL | 1101 |
| RET | 1110 |

Table 2: Table of opcodes

$15) is decremented by 1. The stack pointer always points to the first empty location above the top of stack. The assembly level syntax for this instruction is

CALL target

The machine level encoding for this instruction is

Opcode gggg gggg gggg

where gggg gggg gggg specifies the least 12 bits of the target jump address. The most significant four bits of the target jump address are set equal to the four most significant bits of the PC (after it has been incremented by 1 to point to the next instruction).

The Return instruction pops the top of stack into program counter. Since the stack pointer always points to the first empty location above top of stack, the stack pointer must be first decremented by 1 before poping the contents to the program counter. The assembly level syntax for this instruction is

RET

The machine level encoding for this instruction is

Opcode xxxx xxxx xxxx

### 1.4 Memory System

The memory system for the processor is comprised of 64 Kword (1 word = 16 bits) main memory, 32 word instruction cache, and a simplified data cache. The data cache is as big as the main memory and as fast as the instruction cache. Further all accesses to the data cache result in a hit.

The instruction cache is direct mapped. It is organized into 8 equal blocks. The width of the data bus between the instruction cache and the main memory is 2 words (4 bytes). The instruction cache has an access time of one clock cycle. The main memory has an access time of four clock cycles. You can assume that there will no write operation into instruction cache.

Verilog codes for the main memory, the instruction cache, and data cache will be provided.

### 1.5 Reset Sequence

WISC-S15 has a Reset input. Instructions are executed when Reset input is low. If the Reset input goes high for one clock cycle, the contents of the program counter are cleared and the contents of the stack pointer are set of FFFF.

## 2 The Implementation

In implementing WISC-S15, you can use synthesizable Verilog. Your design must use a five stage pipeline similar to that in the text. You need not implement data forwarding but must implement delayed branching. If needed for design optimization decisions, assume that the instruction frequencies are as shown in Table 3.

## 3 Submission Requirements

The project is to be done in two phases. In the first phase (**due on April 9, 2015**), you are required to implement WISC-S15 without the instruction cache. That is, assume that the instruction cache is as big as the main memory with one clock cycle access delay. Further all accesses to the instruction cache result in hits. In the second phase (**due on May 5, 2015**), the direct mapped instruction cache as described above must be implemented.

### 3.1 Extra Features

You can get bonus points by implementing an interesting feature which is not part of this specification. For example, use of data forwarding is an additional feature. Implementation of a direct mapped small data cache is also an additional feature. Implementation of exception handling is also an additional feature.

| Instruction | Relative Frequency |
|---|---|
| ADD | 15 % |
| SUB | 15 % |
| NAND | 4 % |
| XOR | 3 % |
| INC | 10 % |
| SRA | 1 % |
| SRL | 2 % |
| SLL | 3 % |
| LW | 13 % |
| SW | 6 % |
| LHB | 3 % |
| LLB | 4 % |
| B | 15 % |
| CALL | 3 % |
| RET | 3 % |

Table 3: Relative frequency of instructions in the benchmark programs.

You will get bonus points only if you have a completely working design meeting all the specifications in this document. Remember to save the working design before you begin work on the extra feature.

Implementing additional interesting instructions will also be considered an extra feature. If you implement an additional instruction, clearly specify its assembly level syntax and machine level encoding. Also fully define the actions carried out by the instruction. Write a small program to illustrate the potential improvement in runtime that can result of including your new instruction.

The amount of bonus points will depend on the feature. It will not exceed 25% of the overall grade.

### 3.2 First Phase Report

The first phase report should include the following parts:

1. A brief description of or an introduction to your project. Report its special features, any particular effort you have made to optimize the design, and any major problems you encountered in implementing the design.
2. A statement indicating whether or not your design meets all the requirements specified in this document.
3. Include your entire Verilog code including all testbenches.
4. Synthesize your code and include the area report after the synthesis is completed.

5. The simulation results for test programs (will be given later) for both pre-synthesis and post-synthesis Verilog code. To limit the output size, sample the signals at appropriate times. Clearly mark the crucial points in the output, especially the ones which demonstrate the correctness of your design. Draw a line to indicate the start of each new instruction in the output. Identify the instruction being executed as a comment. Your output should include the contents of registers such as the PC, IR, the outputs of ALU, memory, and register file. Also clearly identify the testbench you used in the simulation. Simulation results with very few comments will be ignored.

6. Summary sheet containing the following information.

   - Names of students
   - Area of your design (except memory, cache).
   - A three column table containing, name of program, execution time of program pre-synthesis Verilog code, execution time of program post-synthesis Verilog code.

### 3.3 Second Phase Report

The second phase report should include the following parts:

1. A brief description of or an introduction to your project. Report its special features, any particular effort you have made to optimize the design, and any major problems you encountered in implementing the design.

2. A statement indicating whether or not your design meets all the requirements specified in this document.

3. Include your entire Verilog code including all testbenches.

4. Synthesize your code and include the area report after the synthesis is completed.

5. The simulation results for test programs (will be given later). To limit the output size, sample the signals at appropriate times. Clearly mark the crucial points in the output, especially the ones which demonstrate the correctness of your design. Draw a line to indicate the start of each new instruction in the output. Identify the instruction being executed as a comment. Your output should include the contents of registers such as the PC, IR, the outputs of ALU, memory, and register file. Also clearly identify the testbench you used in the simulation. Simulation results with very few comments will be ignored.

6. A detailed description of the methodology you used to test the correctness of your design. Include a well commented copy of the testbenches you used to create test inputs and/or verify the outputs from the various blocks. A significant part of your project grade will be based on the testing methodology you used.

7. A signed statement of work by each member of the group. All members of a group may not receive the same grade if the grader feels that there is a significant imbalance in the amount of work done by the group members.

8. Summary sheet containing the following information.

    - Names of students
    - Area of your design (except memory, cache).
    - A three column table containing, name of program, execution time of program pre-synthesis Verilog code, execution time of program post-synthesis Verilog code.

# 4 Grading Scheme

Your project will be graded as follows.

**First Phase Functional Correctness** (20 points): The design will be considered to be functionally correct if the simulation output for the test program matches what was expected. The graders reserve the right to ask you to run the simulation in their presence to check its correctness. If your design is functionally correct before and after synthesis, you will get the full 20 points.

Otherwise, it is your responsibility to illustrate what works and what does not. The grade will be based on the grader's assessment of how much effort will be required to get a fully functional design.

**First Phase Report Quality** (5 points): Organization, readability, documentation, English, spelling, and comments will be factors in assigning grades for this part. **If the summary sheet is not present, you will zero points for this category.**

**Second Phase Functional Correctness** (20 points): The design will be considered to be functionally correct if the simulation output for the test program matches what was expected. The graders reserve the right to ask you to run the simulation in their presence to check its correctness. If your design is functionally correct, you will get the full 20 points.

Otherwise, it is your responsibility to illustrate what works and what does not. The grade will be based on the grader's assessment of how much effort will be required to get a fully functional design.

**Second Phase Quality of design** (20 points): You are eligible for these points only if the design is fully functional. If your design is fully functional, then the score will be 20×Quality_percentile, where the Quality_percentile is defined as follows. We define Quality to be the value of the performance metric. Quality_Percentile is the percentage of groups with lower Quality (i.e., higher value for Area × Execution time) than that of your design (Non-working designs are assumed to have lower quality than all working designs).

**Second Phase Report Quality** (10 points): Organization, readability, documentation, English, spelling, and comments will be factors in assigning grades for this part. The

report should also contain a detailed description of the methodology you used to test the correctness of your design. The description of the verification methodology will form a significant part of this grade. **If the summary sheet is not included, you can get at most 5 points for this category.**

**Project Demonstration** (25 points): Each group will be asked to demonstrate their project soon after the project report is due and before the final exams begin. The demonstration will be used to primarily evaluate the following.

- **Verification methodology**: Designs tested with well thought out systematic procedures will get higher scores. You will be asked to show the testbenches you have used to create test inputs and/or verify the outputs from the various blocks. You will be asked to justify why you chose those inputs to verify the design.
- **Team participation**: Questions will be directed to individuals in each team. Individuals will be asked to explain the functioning of blocks designed by their teammates.

**Bonus** (25 points): Bonus points will be awarded only if your design is fully functional as per the specifications in this document. Bonus points will be based on the quality and nature of the extra features.