

*Department of Electrical and Computer Engineering*  
*University of Wisconsin – Madison*  
ECE 552 Introductions to Computer Architecture  
**Project Description (Last update: Nov. 7, 2012)**

Project report due date: **December 14, 2012**

Project FAQ page: [http://homepages.cae.wisc.edu/~chen/ece552/quartus\\_faq.htm](http://homepages.cae.wisc.edu/~chen/ece552/quartus_faq.htm)

WISC-F12 is a 16-bit processor. In this homework, you will implement a single cycle data path to implement WISC-F12's instructions using Altera Quartus II version 9.0 with Stratix EP1S80F1508C5 device.

The register file comprises sixteen 16-bit registers. Registers \$14 and \$15 are special purpose registers, where Register \$14 serves as the Data Segment (DS) register for load and store instructions, and Register \$15 serves as the Stack Pointer (SP) for Call and Return instructions. The FLAG register contains three bits: Zero (Z), Overflow (V), and Sign bit (N).

All memory systems in WISC-F12 are WORD addressed. In other words, the smallest unit that can be accessed from the memory is a word rather than a byte.

The instruction set architecture (ISA) for WISC-F12 is detailed as follows.

### 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 operand 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 two's-complement representation and save the result in the 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 a FLAG register so that these conditions may be used by later instructions.

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

Opcode rd, rs, imm

The imm field is a 4-bit immediate operand in two's-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.

The FLAG registers are modified only by the following arithmetic/logic instructions: ADD, SUB, NAND, XOR, and INC. The shift instructions SRA, SRL, and SLL do NOT modify the values in the FLAG registers.

- The Z flag is set to 1 if the output of the operation is zero; otherwise, it is cleared to 0.
- The V flag is set to 1 by the ADD SUB, or INC instructions if the operation results in an overflow; otherwise, it is cleared to 0. The NAND and XOR instructions clear the V flag.

- The N flag is set to 1 by the ADD SUB, or INC instruction if the operation result is negative; otherwise, it is cleared to 0. The NAND and XOR instructions clear the N flag.

SRA, SRL, and SLL shift (rs) by number of bits specified in the imm field and saves the result in register rd. With these shift instructions, the 4-bit imm field is treated as an unsigned 4-bit integer. 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/logic/shift instructions 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

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 Branch instruction conditionally jumps to the address obtained by adding the sign-extended 8-bit immediate offset to the address of the next instruction (i.e., address of Branch instruction + 1).

The eight possible conditions are Equal (EQ), Less Than (LT), Greater Than (GT), Not Equal (NE), Greater or Equal (GE), Less or Equal (LE), Overflow (OV), and unconditional branch (UB).

The True condition corresponds to an unconditional branch. The remaining conditions are determined based on the 3-bit flag N, V, and Z which should be set by an ADD or SUB instruction executed prior to the conditional branch instruction.

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  $iiiiiii$  represents the 8-bit signed offset in two's-complement representation:

Table 1: Encoding for Branch conditions

$ccc$	$code$	$Condition$
000	EQ	Equal ( $Z = 1$ )
001	LT	Less Than ( $(N = 1 \text{ and } Z = V = 0) \text{ or } (N = Z = 0 \text{ and } V = 1 \text{ (overflow)})$ )
010	GT	Greater Than ( $(Z = N = V = 0) \text{ or } (Z = 0 \text{ and } N = V = 1 \text{ (overflow)})$ )
011	OV	Overflow ( $V = 1$ )
100	NE	Not Equal ( $Z = 0$ )
101	GE	Greater or Equal (Complement of Less Than)
110	LE	Less or Equal ( $(N = 1 \text{ and } V = 0) \text{ or } Z = 1$ )
111	UB	Unconditional Branch

The Call instruction saves the next instruction address (address of the Call instruction + 1) on top of stack and jumps to the procedure whose starting address is partly specified in the instruction. After saving the program counter, the stack pointer (i.e., register \$15) is incremented 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 RET 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 popping the contents to the program counter. This step of decrement stack pointer must be accomplished using another instruction prior to the RET instruction.

The HALT instruction halts the processor. The processor writes back all unsaved pipeline values and should no longer fetch any instruction. This is very useful for debugging and for verification purposes.

The assembly level syntax for RET and HALT instruction is

Opcode

The machine level encoding for RET and HALT instruction is

Opcode  $xxxx\ xxxx\ xxxx$

Table 2: Table of WiscF12 Instructions

Function	Opcode	Reg. Transfer Operation	Remarks
ADD	0000	$rd \leftarrow rs + rt$	Set or clear ZVN
SUB	0001	$rd \leftarrow rs - rt$	Set or clear ZVN
NAND	0010	$rd \leftarrow \sim(rs \wedge rt)$	Set or clear Z, clear VN
XOR	0011	$rd \leftarrow rs \oplus rt$	Set or clear Z, clear VN
INC	0100	$rd \leftarrow rs + \text{sgnext}(\text{imm})$	Set or clear ZVN
SRA	0101	$rd \leftarrow \gg rs \text{ by imm bits}$	Filled with sign-bit from left
SRL	0110	$rd \leftarrow \gg rs \text{ by imm bits}$	Filled with 0 from left
SLL	0111	$rd \leftarrow \ll rs \text{ by imm bits}$	Filled with 0 from right

LW	1000	$rt \leftarrow M[\text{madr}]$	$\text{madr} = [\text{ds}] + \text{sgnext}(\text{offset})$
SW	1001	$M[\text{madr}] \leftarrow rt$	
LHB	1010	$rt[15:8] \leftarrow \text{immediate}$	
LLB	1011	$rt[7:0] \leftarrow \text{immediate}$	
B	1100	$\text{cond: PC} \leftarrow \text{badr}$	$\text{Badr} = \text{PC} + 1 + \text{sgnext}(\text{offset})$
CALL	1101	$M[\text{sp}] \leftarrow \text{PC} + 1; \text{PC} \leftarrow \text{jadr}; \text{sp} \leftarrow \text{sp} + 1$	$\text{Jadr} = (\text{PC}+1)[15:12] \mid \text{target}$
RET	1110	$\text{PC} \leftarrow M[\text{sp}]$	After $\text{sp} \leftarrow \text{sp} - 1$ is executed.
HALT	1111		Disable all write operations

#### 1.4 Memory System

Two versions of WISC-F12 memory system will be designed: (A) Single cycle and basic pipeline, and (B) cache-enabled. All memory components are word addressable.

A FLAG register contains three bits: Zero (Z), Overflow (V), and Sign bit (N) will be implemented in the controller to facilitate branch condition evaluation by the branch instruction B.

##### Version A

The version A WISC-F12 memory system supports single cycle execution and basic pipeline implementation. In particular, it supports same cycle read from IM and DM. In other words, reading instruction and data can be accomplished within the same clock cycle when address is furnished. It also assumes that writing into DM can be accomplished in the next clock cycle.

Version A memory system consists of

- A  $16 \times 16$  register file (RF)
- A  $16 \times 16$  instruction memory (IM), and
- A  $16 \times 16$  data memory (DM).

The register file (RF) comprises sixteen 16-bit registers. Registers \$14 and \$15 are special purpose registers, where Register \$14 serves as the Data Segment (DS) register for load and store instructions, and Register \$15 serves as the Stack Pointer (SP) for Call and Return instructions.

The IM and DM shall be realized with the same type of 16-word, 16 bit/word register files used for RF.

##### Version B

The version B WISC-F12 memory system will include

- a register file contains sixteen 16-bit registers,
- an optional cache that supports same cycle read if a hit,
- a  $64K \times 16$  same cycle read instruction memory (IM), and
- a  $64K \times 16$  multi-cycle read/write data memory (DM).

The RF shall be identical to that in the version A memory system.

IM is read-only. All instructions will be preloaded into IMs at compile time by utilizing a memory initialization file (.mif). Detailed information about the .mif file can be found here:

[http://quartushelp.altera.com/9.1/mergedProjects/reference/glossary/def\\_mif.htm](http://quartushelp.altera.com/9.1/mergedProjects/reference/glossary/def_mif.htm).

You can create an IM by using 1-port ROM through MegaWizard Plug-In Manager, and use the instantiated IM in your Schematic file (\*.bdf) or in your Verilog file.

A DM can also be instantiated through MegaWizard, except that in this case you need to create a custom 1-port RAM variation.

Specific pre-designed IM and DM may be provided as part of project supplementary files.

The optional cache may be realized with one or more register files and supporting combinational logics.

### 1.5 Reset Sequence

WISC-F12 has a **positive Reset** input. Instructions are executed when *Reset* is low. When the *Reset* goes high, the program counter is reset to 0x0000 and the stack pointer is reset to 0xB in version A or 0xFF00 in version B memory system.