ECE 552 - Introduction to Computer Architecture

WiscF12 16-Bit Processor with 5-Stage Pipeline

Instruction Set Architecture

Aman Rakesh Chadha
9068354597
## 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

\[
\text{Opcode } rd, rs, rt
\]

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:

\[
\text{Opcode } rd, rs, imm
\]

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.
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 two's-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.
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

\[ B \text{ cond, offset} \]

The machine level encoding for this instruction is

\[ \text{Opcode } xccc \text{ iiiii} \]

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>
<thead>
<tr>
<th>ccc</th>
<th>code</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>EQ</td>
<td>Equal (Z = 1)</td>
</tr>
<tr>
<td>001</td>
<td>LT</td>
<td>Less Than ((N = 1 and Z = V = 0) or (N = Z = 0 and V = 1)</td>
</tr>
<tr>
<td>010</td>
<td>GT</td>
<td>Greater Than ((Z = N = V = 0) or (Z = 0 and N = V = 1)</td>
</tr>
<tr>
<td>011</td>
<td>OV</td>
<td>Overflow (V = 1)</td>
</tr>
<tr>
<td>100</td>
<td>NE</td>
<td>Not Equal (Z = 0)</td>
</tr>
<tr>
<td>101</td>
<td>GE</td>
<td>Greater or Equal (Complement of Less Than)</td>
</tr>
<tr>
<td>110</td>
<td>LE</td>
<td>Less or Equal ((N = 1 and V = 0) or Z = 1)</td>
</tr>
<tr>
<td>111</td>
<td>UB</td>
<td>Unconditional Branch</td>
</tr>
</tbody>
</table>

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

\[ \text{CALL } \text{target} \]

The machine level encoding for this instruction is

\[ \text{Opcode } gggg \text{ gggg gggg} \]

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 of WiscF12 Instructions

<table>
<thead>
<tr>
<th>Function</th>
<th>Opcode</th>
<th>Reg. Transfer Operation</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>0000</td>
<td>rd ← rs + rt</td>
<td>Set or clear ZVN</td>
</tr>
<tr>
<td>SUB</td>
<td>0001</td>
<td>rd ← rs - rt</td>
<td>Set or clear ZVN</td>
</tr>
<tr>
<td>NAND</td>
<td>0010</td>
<td>rd ← ~(rs ∧ rt)</td>
<td>Set or clear Z, clear VN</td>
</tr>
<tr>
<td>XOR</td>
<td>0011</td>
<td>rd ← rs ⊕ rt</td>
<td>Set or clear Z, clear VN</td>
</tr>
<tr>
<td>INC</td>
<td>0100</td>
<td>rd ← rs + sgnex(imm)</td>
<td>Set or clear ZVN</td>
</tr>
<tr>
<td>SRA</td>
<td>0101</td>
<td>rd ← &gt;&gt; rs by imm bits</td>
<td>Filled with sign-bit from left</td>
</tr>
<tr>
<td>SRL</td>
<td>0110</td>
<td>rd ← &gt;&gt; rs by imm bits</td>
<td>Filled with 0 from left</td>
</tr>
<tr>
<td>SLL</td>
<td>0111</td>
<td>rd ← &lt;&lt; rs by imm bits</td>
<td>Filled with 0 from right</td>
</tr>
<tr>
<td>LW</td>
<td>1000</td>
<td>rt ← M[madr]</td>
<td>madr = [ds] + sgnext(offset)</td>
</tr>
<tr>
<td>SW</td>
<td>1001</td>
<td>M[madr] ← rt</td>
<td></td>
</tr>
<tr>
<td>LH</td>
<td>1010</td>
<td>rt[15:8] ← immediate</td>
<td></td>
</tr>
<tr>
<td>LL</td>
<td>1011</td>
<td>rt[7:0] ← immediate</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1100</td>
<td>cond: PC ← badr</td>
<td>Badr = PC + 1 + sgnex(offset)</td>
</tr>
<tr>
<td>CA</td>
<td>1101</td>
<td>M[sp] ← PC + 1; PC ← jadr; sp ← sp + 1</td>
<td>Jadr = (PC+1)[15:12]</td>
</tr>
<tr>
<td>RE</td>
<td>1110</td>
<td>PC ← M[sp]</td>
<td>After sp ← sp - 1 is executed.</td>
</tr>
<tr>
<td>HA</td>
<td>1111</td>
<td></td>
<td>Disable all write operations</td>
</tr>
</tbody>
</table>