ECE 552 - Introduction to Computer Architecture

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

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.

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.

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 iiiiiiii 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 and Z = V = 0) or (N = Z = 0 and V = 1
010	GT	Greater Than ((Z = N = V = 0) or (Z = 0 and N = V = 1
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 and V = 0) 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 of WiscF12 Instructions

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