Assignment for Laboratory 9

Computer Science 240

Due: Before lab (hand in only the first page)

You will be spending the next two labs implementing and experimenting with the mini-MIPS machine we learned about at the beginning of the semester. This machine is very similar to the single-cycle MIPS machine that you will be studying in lecture. After reading the specification for the simple architecture described on the following pages, answer these questions:

1. How many instructions are there in the instruction set?

2. How many bits are there in each instruction?

3. What instruction is represented by the hexadecimal value 0x0101? (each digit represents 4 bits)

4. What is the 16-bit binary form of the following instruction (don't forget that the order or operands is different in mini-MIPS than in MIPS):

ADD R1 R1 R4

What are the contents of Register 1 and Register 4 after this instruction is executed?

5. Given the following instruction at address 8 in memory:

    8: BEQ R5 R6 C

Assume register 5 contains FFFE, and register 6 contains FFFE.

After this instruction is executed, what will be the address of the next instruction?

6. Repeat question 5, but assume that the original value of register 5 = 0003, and register 6 = 0002. What will be the address of the next instruction?

Description of Mini-MIPS Architecture

The device you will design is a simple RISC-style CPU. As is traditional in a RISC architecture, a small number of highly efficient instructions and addressing modes will be defined.

The word length for this device is 16 bits. The CPU reads and writes words, and addresses in instruction and data memory may begin at even or odd addresses.

The starting address of every program must be address 0, since the program counter is initialized to 0 by a reset.

Because instructions are 16 bits in length, the program counter must be incremented by 2 to move to the next instruction.

There are 16 registers available, 14 general-purpose registers (R2-R15) and 2 constant value registers (R0-R1). The R0 register always contains 0 and the R1 register always contains 1. Neither R0 nor R1 can be modified by any instruction.

Instructions specify register IDs using 4-bit values. The program counter is an 8-bit register, but it cannot be directly accessed by any instruction.

Your CPU will implement the following instruction set (all instructions are one word in length):

Instruction	Meaning	Op 4-bit	Rs 4-bit	Rt 4-bit	Rd 4-bit
LW R_t,offset(R_s)	R_t loaded with word from Data Memory at address(Rs + offset)	0000	0-15	0-15	offset
SW R_t,offset(R_s)	Data Memory address(Rs + offset) stored with word from Rt	0001	0-15	0-15	offset
ADD R_s,R_t,R_d	R_d := R_s + R_t	0010	0-15	0-15	0-15
SUB R_s,R_t,R_d	R_d := R_s - R_t	0011	0-15	0-15	0-15
AND R_s,R_t,R_d	R_d := R_s AND Rt	0100	0-15	0-15	0-15
OR R_s,R_t,R_d	R_d := R_s OR R_t	0101	0-15	0-15	0-15
SLT R_s,R_t,R_d	If R_s<R_t then R_d:=1 else R_d := 0	0110	0-15	0-15	0-15
BEQ R_s,R_t,offset	If R_s=R_t then pc:=pc+2+(offset*2) else pc:=pc+2	0111	0-15	0-15	offset
JMP offset	Jump to abs. addr = offset*2	1000	---12 bit offset-----		

The high four bits always specify the operation while the low 12 bits specify registers and offsets, depending on the instruction type.

Mathematical operations (ADD, SUB, AND, OR, SLT) operate only on registers.

Data transfer and branching operations (LW, SW, BEQ) operate on two registers and an absolute offset value.

The jump operation (JMP) operates on a single 12-bit offset.

Mathematical and logical operations treat the low 12 bits as register identifiers. The high four bits represent R_a, the middle four R_t and the low four R_d as specified in the previous table.

Addition, subtraction and set-greater-than treat the contents of R_s and R_t as 16 bit, two's complement numbers. Logical operations (AND,OR) treat R_s, R_t and R_d as unsigned, 16 bit values.

Load and store operations use the low 12 bits to specify memory address, source/destination register and offset respectively. R_s specifies the register containing a base address. R_d specifies the offset and R_t specifies the destination (or source) for data being read (or stored). Note that the only addressing mode is register indirect.

The branch equal (BEQ) operator uses the low 12 bits to specify the registers for comparison and the branch offset. R_s and R_t specify registers whose values are to be compared. If they are equal, the program counter is incremented by 2, and then incremented by (2 * offset).

Offsets for loading, storing and branching are 4 bit, 2's complement numbers that specify offsets as words. Be cautious as you add and subtract offsets to get new program counter values. The length of the offset limits how far a program can branch using the BEQ command.

The jump operation uses the low 12 bits to specify a single offset value (this value should be in the range of 0 to 127, to guarantee a legal address within the range of 0 to 254. Unlike the branch offset, the jump offset is not interpreted as a 2's complement number. An 8-bit absolute address is formed by multiplying the offset value by 2 (shift left). The JMP operation is achieved by setting the program counter to this address.