

Assignment for Laboratory 7

Computer Science 240

Due: Wednesday, before lab

You will be spending the next several labs implementing and experimenting with a simple processor. This machine is very similar to the MIPS machine that you are learning about 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 specified by the opcode 0101?
4. Give the 16-bit binary representation of the following instruction (assume the operands are in hexadecimal notation):

ADD 2 3 4

What would be the effect of executing this instruction?

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

8: BEQ 5 6 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 only addressing mode used by arithmetic and logic operations is register direct. The only addressing mode used by data transfer and branching instructions is register indirect.

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 a 16-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 Rs,Rt,offset	Rt loaded with word from Data Memory at address(Rs + offset)	0000	0-15	0-15	offset
SW Rs,Rt,offset	Data Memory address(Rs + offset) stored with word from Rt	0001	0-15	0-15	offset
ADD Rs,Rt,Rd	$R_d := R_s + R_t$	0010	0-15	0-15	0-15
SUB Rs,Rt,Rd	$R_d := R_s - R_t$	0011	0-15	0-15	0-15
AND Rs,Rt,Rd	$R_d := R_s \text{ AND } R_t$	0100	0-15	0-15	0-15
OR Rs,Rt,Rd	$R_d := R_s \text{ OR } R_t$	0101	0-15	0-15	0-15
SLT Rs,Rt,Rd	If $R_s < R_t$ then Rd:=1 else Rd := 0	0110	0-15	0-15	0-15
BEQ Rs,Rt,off	If $R_s = R_t$ then pc:=pc+2+(offset*2) else pc:=pc+2	0111	0-15	0-15	offset
JMP addr	Jump to abs. addr	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_d , the middle four R_s and the low four R_t 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 the number of words specified by the offset value.

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 in a substantially different way. Unlike the branch offset, the jump offset is not interpreted as a 2's complement number. The jump address is calculated as follows:

$$\begin{array}{r} \underline{15-13} \quad \underline{12..1} \quad \underline{0} \\ PC_{15-13} \quad 12\text{-bit} \quad 0 \\ \text{offset} \end{array}$$

The 16-bit absolute address is formed by first multiplying the offset value by 2, resulting in a 13-bit number. Then, the most significant 3 bits from the program counter is pre-pended to the result to produce a 16-bit number. The JMP operation is achieved by setting the program counter to this value. Note that the jump operation cannot reach all memory locations.