Computer Science 605.611

Problem Set 4

1. (5) The nine instructions that make up our MIPS core instruction subset are listed in the table below. The table also shows the time required to fetch and execute each of these nine instructions on a certain implementation of the single cycle data path.

|  |  |
| --- | --- |
| Instruction | Time required to fetch and execute |
| add, sub, slt, and, or | 50ns per instruction |
| lw | 100ns |
| sw | 90ns |
| beq | 70ns |
| j | 50ns |

What is the maximum clock rate that can be used for this implementation of the single cycle data path? Express your answer in MHz.

Max = 1/maxtimeperinstruction

lw is takes the longest to fetch and execute

max = 1/100ns = **10 MHz**

2. (3) The single-cycle datapath ALU outputs a 32-bit data result (R31, R30, … , R1, R0) along with a single-bit zero flag (ZERO). Write down a logic expression that gives ZERO as a logic function of the 32-bit result.

ZERO = \_\_\_\_(R31’ + R30’ + … + R1’ + R0’)\_\_\_\_\_\_\_ (NORing each bit results in the ZERO flag)

3. The addu instruction performs the same operation as the add instruction. The only difference is that add triggers an exception if the addition results in a signed overflow, while addu does not. The instruction addu $4, $5, $6 places into register $4 the 32-bit sum of the contents of register $6 plus the contents of register $5. Answer the following questions about the instruction addu $4, $5, $6 :

a) (3) If registers $5 and $6 contain integers that differ in sign, can the result in $4 be incorrect due to an overflow? Explain your answer.

No, due to the way overflow works with unsigned integers and addu instead of add. Unsigned integers already deal with overflow due to the carryout bit while signed integers do not. Since addu deals with unsigned integers it will work even if the signs differ.

b) (3) If an overflow does not occur and registers $5 and $6 both contain a negative integer , what will be the sign of the result in $4?

The sign should still remain negative. Since addu is dealing with unsigned numbers it does not deal with signs at all. Due to this it will simply perform the operation as requested, since both of the integers are negative, if an overflow doesn’t happen, the twos complement representation of the result will also be negative.

c) (3) If there an overflow does occur and registers $5 and $6 bot contain a positive integer, what will be the sign of the result in $4?

The sign of the result in $4 would most likely be incorrect. Since overflow occurs and addu doesn’t actually handle overflow, the answer would most likely be a meaningless one to us. Since the answer wouldn’t make sense and that is where the sign is coming from, more than likely it would be negative instead of positive as it should be.

4. The lw, sw and beq MIPS instructions all use the low 16-bits within the machine instruction together with a register to generate a 32-bit memory address.

a) (3) Identify the bits within the 32-bit sw machine instruction that indicate the register to use to generate the 32-bit memory address? The instruction bits are numbered 0 to 31 from right to left.

The bits that indicate the register to use to generate the memory address are: the 21st-25th bits

b) (3) Explain how the 32-bit lw machine instruction uses the low 16 bits from the instruction to generate the 32-bit memory address.

There are two steps to generating the 32-bit memory address. The first is to sign extend the 16 bits of the instruction that refers to the offset. Then you just need to add bits in position 21-25 to it. This will return the memory address.

c) (3) Explain how the 32-bit beq machine instruction uses the low 16 bits from the instruction to generate the 32-bit branch target address. This was explained in module 1 as well as in appendix A of the textbook.

To generate the 32-bit branch target address it is similar to the last answer. The first step is to take the 16-bit offset field and sign extend it to 32 bits. This is then shifted left by 2 bits and added to the address of the instruction.

d) (3) Explain how the 32-bit j (jump) machine instruction generates the 32-bit address to which the instruction transfers control. This was explained in module 1 as well as in appendix A of the textbook.

The way that the jump instruction generates the 32-bit address to which the instruction transfers control is done in a few steps. The first is looking at the instruction that has come in, bits 31-6 are the target address, the lower 6 bits contain the current program counter. The processor concatenates these two pieces to get the 32-bit target address.

5. Listed below are three of the possible patterns for the 2-bit ALUOp control. Indicate what operation the ALU performs for each of these 2-bit patterns.

a) (3) 01

Branch Equal

b) (3) 10

this could be:  
ADD, SUB, AND, OR, or set-on-less-than

c) (3) 00

store word

6. The format of each MIPS machine instruction defines how many separate fields the 32-bit machine instruction contains as well as the width in bits of each field. The instruction opcode determines the machine instruction format (R-type, I-type or J-type). Specify the number of fields and the width of each field within each machine instruction whose opcode is shown below:

a) (3) 000100

This opcode is for the beq instruction. It contains 4 fields

Field 1: Opcode with 6 bits

Field 2: RS with 5 bits

Field 3: RT with 5 bits

Field 4: Offset with 16 bits

b) (3) 000000

This opcode is for R type instructions that follow this pattern.

Opcode: 6 bits

rs: 5 bits

rt: 5 bits

rd: 5 bits

Shift amount: 5 bits

Function code: 5 bits

6 fields including opcode

c) (3) 000010

This is the jump instruction op code:

This has 2 fields

Opcode: 6 bits

Target address: 26 bits

d) (3) 101011

This code corresponds to the LW instruction

Opcode: 6 bits

Rs: 5 bits

Rt: 5 bits

Offset: 16 bits

4 fields total.

7. (5) Identify the control bits within the single-cycle data path that are used to decide whether the beq instruction should transfer control and explain how the transfer is performed.

The Control bits used are: ALUOp, ALUSrc, Branch, Zero

Beq transfers control by:

1. ALU computes result by subtracting value
2. Zero flag is generated
3. Control unit checks branch control bit and zero flag, if both branch and zero flag are set control is transferred
4. Target address is computed and a new program counter value is made from that.

8. (5) The diagram below shows a logic gate implementation of a device that accepts a single control bit C along with 4 data inputs N3 through N0. The device outputs 8 bits X7 through X0.

Chart

Description automatically generated

Complete the table below to show the X outputs generated by the device in response to the C and N inputs shown below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | N3 | N2 | N1 | N0 | X7 | X6 | X5 | X4 | X3 | X2 | X1 | X0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

8. All MIPS opcodes are 6 bits in size and each opcode can be represented as two hex digits.

The circuit shown below is one option for generating the 9 control bits required to implement an instruction on the single-cycle datapath. For each of the instructions listed below show the 2-digit hex number that corresponds to the required 6 input bits (Op5 through Op0) and the 3-digit hex number that corresponds to the 9 output control bits that should be output for each set of 6 input bits.

A diagram of a computer

Description automatically generated

a) (3) beq input= \_04\_ output = \_005\_

b) (3) lw input= 23\_ output = \_0F0\_\_

c) (3) or input= 00\_ output = \_\_122\_

d) (3) add input= \_00\_ output = \_\_122\_\_

9. (7) Some MIPS instructions use the low 16 bits within the machine instruction as a displacement or immediate operand. For each of the instructions listed below, explain whether the instruction sign-extends or zero-extends the low 16 instruction bits.

jr sign-extended

addiu zero-extend

addi sign-extend

ori zero-extend

lui zero-extend

andi zero-extend

j sign-extended

10. Reproduced below is one version of the MIPS single-cycle data path diagram discussed in module 4.

A diagram of a computer

Description automatically generated

For each of the following instructions, indicate whether this data path, as shown, DOES or DOES NOT support the instruction. Explain your answer in each case.

a) (3) and

This DOES support the and instruction. It contains the ALUOp which allows to determine that and is being performed. The other control bits are necessary for AND to work.

b) (3) lw

This DOES support the lw instruction. This is because it has both an Add and sign extended components which are requirements for load word to work.

c) (3) j

This DOES NOT support the instruction. This is because jump requires an extra control signal from the opcode that this data path does not contain.

d) (3) beq

This DOES support the and instruction. This is because it has the branch control code as well as the ability to subtract and zero flag all of which are necessary for beq to work. It also has the PC, another necessary piece for beq to work.

e) (3) jr

This DOES NOT support the instruction. This is because jump requires an extra control signal from the opcode that this data path does not contain.

11. (6) Our “MIPS core instruction subset” is defined as the following group of instructions:

add, sub, and, or, slt, lw, sw, beq & j. List each of the single-cycle datapath control bits required for this MIPS core instruction subset and explain the purpose of each control bit.

RegDst: Determines which register the result of the ALU operation should be written to

Jump: This bit is used for jump instructions, set it to 1 means a jump should occur

Branch: Used for branch instructions, signals whether a branch should be taken. 1 means branch should be taken

MemRead: Enables memory read operations for instructions like lw.

MemtoReg: Control bit indicates whether the data to be written to a register comes from memory or the ALU result

ALUOp: Specifies operation to be performed by ALU (i.e. addition, subtraction…)

MemWrite: Enables memory write operations for instructions like sw

ALUSrc: Selects one of the ALU inputs. Specifies if the second ALU operand should come form immediate field or from the second registers value

RegWrite: Signals whether a write operation should be performed on the destination register.