

Your Name: \_\_\_\_\_ Your ID #: \_\_\_\_\_

**Question 1.**

Answer each question with either “one sentence” or “a few key words”

1. What are the five basic components of a computer? (missing 3 of 5 = 0 point)

Control      Datapath      Memory      Input      Output

2. Suppose A is a byte address (A=0,1,2...). Explain what it means by “alignment” in memory access (with A) by write down a boolean expression in terms of A. Similarly, explain what it means by “not align” by write down another boolean expression.

Alignment expression: (A%4) = 0

Not-Alignment expression: (A%4) <> 0

3. What are the three basic instruction formats in MIPS? What format do beq and bne (conditional branches) use?

3 formats: R-Type      I-Type      J-Type

beq and bne use format: I-Type

4. Why in MIPS, there is no subtract immediate “subi” instruction?

Because: subi is the same as addi with a negative constant

5. What is a pseudo instruction? Why they exist (hint: what is the principle of instruction design)?

A pseudo instruction is processed by software (compiler, etc.)

It can be decomposed into a sequence of instructions

We differentiate them from regular instructions because we want to keep the instruction set

As simple as possible to simplify hardware design

Your Name: \_\_\_\_\_ Your ID #: \_\_\_\_\_

**Question 2:**

Part a) Convert  $-114_{ten}$  into a 16-bit 2's complement binary number

$64+32+16+2=114$ , so  $114=01110010$ , 2's complement =  $10001110 = -114$

Part b) Given a bit pattern, 1100 0101 on an 8-bit machine

What does it represent assume that it is

A 2's complement integer?  $00111011 = 32+16+8+2+1 = 59$ , so  $11000101 = -59$

A 1's complement integer?  $00111010 = 32+16+8+2 = 58$ , so  $11000101 = -58$

An unsigned integer?  $11000101 = 128 + 64 + 4 + 1 = 197$

Part c) Write the shortest MIPS instruction sequence to accomplish the following: add two 2's complement values as their absolute values. *You are allowed to use only instructions such as* (slt, bne, sub, add). In other words, we are like creating a pseudo instruction "addabs." "addabs \$t1, \$t2, \$t3" is doing  $\$t1 = |\$t2| + |\$t3|$ . Take the absolute value of \$t2, add it to the absolute value of \$t3, and store the result in \$t1. Let's further suppose that this piece of program is stored at address  $4000_{ten}$ . For this question, put "jr \$ra" as the last instruction to indicate "jumping back to return address." Also, you can assume that \$t1, \$t2, and \$t3 are the 3 registers in use and you don't need to worry about making it like a complete procedure. I just want to see the equivalent instruction sequence to "addabs \$t1, \$t2, \$t3."

4000	addabs: slt	\$s1, \$t2, \$zero
4004	beq	\$s1, \$zero, 1
4008	sub	\$t2, \$zero, \$t2
4012	slt	\$s1, \$t3, \$zero
4016	beq	\$s1, \$zero, 1
4020	sub	\$t3, \$zero, \$t3
4024	add	\$t1, \$t2, \$t3
4028	jr	\$ra
4032		
4036		
4040		
....		

The correct answer should use "bne" but using "beq" is fine. This is the shortest.

Your Name: \_\_\_\_\_ Your ID #: \_\_\_\_\_

### Question 3.

Part a) Consider the following C code:

```
int sum(int* a, int* b, int* c)
{ int temp;
  temp = 0;
  for (i=0; i<50; i++) {a[i] = b[i]+c[i]; temp += a[i];}
  printf1(temp);
  return temp;
}
```

Assume that a, b, c are arrays of words and the base address of a is in \$a0, base address of b is in \$a1, and base address of c is in \$a2. We further assume that we can use \$t0 for i and \$t1 for temp. When call the function “printf1,” you can assume that printf1 will take \$v0 as the argument. Moreover, printf1 is nothing but a beginning label of a system MIPS routine staying somewhere in the memory.

**Step 1.** Convert the following 3 lines into a “prologue part” of the MIPS procedure which allows to be called by any other MIPS program from anywhere in the memory.

```
int sum(int* a, int* b, int* c)
{ int temp;
  temp = 0;
```

```
sum: addi    $sp,$sp,-4
      sw     $ra,0($sp)
      add   $t1,$0,$0
```

**Step 2.** Now, convert “for (i=0; i<50; i++) {a[i] = b[i]+c[i]; temp += a[i];} printf1(temp);” into the “body part” of the MIPS procedure.

```
      add   $t0,$0,$0
      addi  $t2,$0,50
loop:  beq   $t0,$t2,done
      lw   $t3,0($a1)
      lw   $t4,0($a2)
      add  $t3,$t3,$t4
      sw   $t3,0($a0)
      add  $t1,$t1,$t3
      addi $a0,$a0,4
      addi $a1,$a1,4
      addi $a2,$a2,4
      addi $t0,$t0,1
      j    loop
done:  add  $v0,$t1,$0
      jal  printf1
```

Your Name: \_\_\_\_\_ Your ID #: \_\_\_\_\_

**Step 3.** Now convert “`return temp;`” into the “epilogue part” of the MIPS procedure.

```
lw    $ra, 0($sp)
add   $v0,$0,$t1
addi  $sp,$sp,4
jr    $ra
```

Part b) Now, consider the main program:

```
void main()
{
    int a[50], b[50], c[50];
    int result;
    ...
    result = sum(a, b, c);
}
```

Assume the *global pointer* \$gp points to a[0]. Convert this piece of C code in another MIPS program which will call the procedure you wrote for Part a).

#main program is just another subroutine

```
main:addi    $sp,$sp,-4
        sw     $ra, 0($sp)
        addi   $a0,$gp,0
        addi   $a1,$gp,200
        addi   $a2,$gp,400
        jal    sum
        ...
        add    $t1, $v0, $0
        lw    $ra, 0($sp)
        addi  $sp,$sp,4
        jr    $ra
```

Your Name: \_\_\_\_\_ Your ID #: \_\_\_\_\_

#### Question 4:

Suppose  $k$ ,  $f$ ,  $a$ ,  $b$  are all integer type in C. Further assume that we have the mapping already as  $f:\$s0$ ,  $a:\$s1$ ,  $b:\$s2$ , and  $k:\$t1$ . Convert the following C code into MIPS.

```
switch (k) {
case 1: f=a+b; break; /* k=0*/
case 2: f=a-b; break; /* k=1*/
case 3: f=a*b; break; /* k=2*/
case 4: f=a/b; break; /* k=3*/
case 5: f=a%b; break; /* k=4, module operation */
default: f=0; break;
}
```

```
        addi $t2,$t1,-1
        bne  $t2,$0,L1
        add  $s0,$s1,$s2
        j    Exit          # case 1
L1:     addi $t2,$t1,-2
        bne  $t2,$0,L2
        sub  $s0,$s1,$s2
        j    Exit          # case 2
L2:     addi $t2,$t1,-3
        bne  $t2,$0,L3
        mult $s1,$s2
        mflo $s0
        j    Exit
L3:     addi $t2,$t1,-4
        div  $s1,$s2
        bne  $t2,$0,L4
        mflo $s0
        j    Exit
L4:     addi $t3,$t1,-5
        bne  $t2,$0,L5
        mfhi $s1
        j    Exit
L5:     add  $s0,$0,$0
Exit:
```