Kaleb J Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Homework 3
Computer Architecture
CSCI 361, Fall 2014
Due: October 20, 2014 in class

Instructions: Complete the problems enumerated below. No collaboration is permitted on this or any assignment. Correct answers without work shown will not receive full credit. Answers must typed and neatly formatted. You will submit a printed hardcopy of your work at the beginning class on the specified due date. Include your name and email address on each page of your submission. Please include a header containing your name on every page. Late assignments will not be accepted.

Problem 1 [10 pts]

Consider the following MIPS loop:

```
LOOP :    slt $t2 , $0 , $t1  #if $0 is less then $t1, $t2=1, else 0
          beq $t2 , $0 , DONE #if the above set $t2=0 then done else:

          subi $t1 , $t1 , 1  #subtract 1 from t1 store result in t1
          addi $s2 , $s2 , 2  #add 2 to s2 store result in s2
          j LOOP              #return to slt call
DONE :
```

(a) Assume that the register $t1 is initialized to the value 10. What is the value in register $s2 assuming the $s2 is initially zero.

1. t1 = 10	t2 = 1	t1 = 9	s2 = 2
2. t1 = 9	t2 = 1	t1 = 8	s2 = 4
3. t1 = 8	t2 = 1	t1 = 7	s2 = 6
4. t1 = 7	t2 = 1	t1 = 6	s2 = 8
5. t1 = 6	t2 = 1	t1 = 5	s2 = 10
6. t1 = 5	t2 = 1	t1 = 4	s2 = 12
7. t1 = 4	t2 = 1	t1 = 3	s2 = 14
8. t1 = 3	t2 = 1	t1 = 2	s2 = 16
9. t1 = 2	t2 = 1	t1 = 1	s2 = 18
10. t1 = 1	t2 = 1	t1 = 0	**s2 = 20**
11. t1 = 0	t2 = 0	DONE:	

Kaleb J Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

(b) For the loop above and using the values given in part (a), write the equivalent C code routine. Assume that the registers $s1, $s2, $t1, and $t2 are integers A, B, i, and temp, respectively. (since $s1 is not used in the above loop, ignoring "int A")

```
int B = 0;        /* $s2 increment by 2 while $t1 > 0*/
int i = 10;       /* $t1 compare to $0, "-1" if greater */
int temp = 0;     /* $t2 set to 1 if $t1 is greater than $0*/

while (temp < i) {
    B += 2;
    i--;
}
printf("$s2 = %d\n", B);
```

Problem 2 [10 pts]

Suppose the program counter (PC) is set to 0x2000 0000.

(a) Is it possible to use the branch-on-equal (beq) MIPS assembly instruction to set the PC to this same address?

0x 2 0 0 0 0 0 0 0
0010 0000 0000 0000 0000 0000 0000 0000
| 16 bits |

We are allowed 16 bits in an I type instruction so now we would not be able to set the PC to this addr in one "beq" instruction we would need to branch to a location that contained a jump addr and continue in this fashion until we reached the desired memory location.

(b) Is it possible to use the jump (j) MIPS assembly instruction to set the PC to the address as 0x4000 0000?

0x 2 0 0 0 0 0 0 0
0010 00 00 0000 0000 0000 0000 0000 0000
| 26 bits |

This would get us closer but again no we would fall short of the desired memory location.

Kaleb J Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Problem 3 [10 pts]

Write MIPS assembly code that creates the 32-bit constant

0010 0000 0000 0001 0100 1001 0010 0100 (base 2)

and stores that value to register $t1.

```
addi $t1, $0, 1    # start: 0000 0000 0000 0000 0000 0000 0000 0001
sll $t1, $t1, 13   #result: 0000 0000 0000 0000 0010 0000 0000 0000
addi $t1, $t1, 1   #result: 0000 0000 0000 0000 0010 0000 0000 0001
sll $t1, $t1, 2    #result: 0000 0000 0000 0000 1000 0000 0000 0100
addi $t1, $t1, 1   #result: 0000 0000 0000 0000 1000 0000 0000 0101
sll $t1, $t1, 3    #result: 0000 0000 0000 0100 0000 0000 0010 1000
addi $t1, $t1, 1   #result: 0000 0000 0000 0100 0000 0000 0010 1001
sll $t1, $t1, 3    #result: 0000 0000 0010 0000 0000 0001 0100 1000
addi $t1, $t1, 1   #result: 0000 0000 0010 0000 0000 0001 0100 1001
sll $t1, $t1, 3    #result: 0000 0001 0000 0000 0000 1010 0100 1000
addi $t1, $t1, 1   #result: 0000 0001 0000 0000 0000 1010 0100 1001
sll $t1, $t1, 3    #result: 0000 1000 0000 0000 0101 0010 0100 1000
addi $t1, $t1, 1   #result: 0000 1000 0000 0000 0101 0010 0100 1001

sll $t1, $t1, 2    #result: 0010 0000 0000 0001 0100 1001 0010 0100
```

Problem 4 [10 pts]

Provide a minimal set of MIPS instructions that may be used to implement the following pseudoinstructions:

(a) not $t1 , $t2 # bit - wise invert

```
nor $t0, $t1, $t2
```

(b) move $t1 , $t2

```
lw $t0, 0($t1)
sw $t0, 0($t2)
```

(c) blt $t6 , $t7 , label # branch if less than

(this is not specific at all, as such it must be open to interpretation and therefore I am assuming we are checking if $t6 is less than $t7 and if it is less then we are going to branch to “label”)

```
slt $t0 , $t6 , $t7      #if $t6 is less then $t7, $t0=1, else 0
beq $t0 , $0 , LABEL     #if the above set $t0=0 then branch
… some other stuff ...
LABEL:
```

Kaleb J Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Problem 5 [10 pts]

The follow instruction is not included in the MIPS instruction set:
rpt $t2 , LOOP # if (R [rs] >0) R [rs]= R [rs] -1 , PC = PC +4+ BranchAddr

where LOOP is some label.

(a) If this instruction were to be implemented in the MIPS instruction set, what is the most appropriate instruction format?

Considering we only have 1 opcode (6 bits), 1 register (5 bits), a constant (5 bits) and an addr (all remaining bits) I would say an **I type instruction** would do nicely here!

(b) What is the shortest sequence of MIPS instructions that performs the same operation?

```
LOOP:
     slt $t0, $0, $t2    #if $t2 is greater than $0, $t0=1, else 0
     beq $t0, $0, TARGET #if above set $t0=0 branch to TARGET else:
     subi $t2, $t2, 1
     j LOOP
TARGET:
```

NEXT PAGE

Kaleb J Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Problem 6 [20 pts]

Translate the following C code to MIPS assembly code. Use a minimum number of instructions.

```
for ( i = 0; i < a ; i ++)
     for ( j = 0; j < b ; j ++)
          D [4 * j ] = i + j ;
```

Assume that the values of a, b, i, and j are in registers $s0, $s1, $t0, and $t1, respectively. Also, assume that register $s2 holds the base address of the array D.

a = $s0	b = $s1	compare1 = $t2	compare2 = $t3	temporary array location = $t4
j = $t1	i = $t0	result of i+j = $t5	move register = $t6	

```
LOOP1:
     slt $t2, $s0, $t0   #if $t0 is greater than $s0, $t2=1, else 0
     beq $t2, $0, DEST   #if above set $t2=0 branch to DEST else:
     LOOP2:
          slt $t3, $s1, $t1   #if $t1 > $s1, $t3=1, else 0
          beq $t3, $0, LOOP1  #if above set $t3=0 branch LOOP1 else:
          lw $t6, 0($t1)      #save j in $t6 so we don't change j
          sll $t4, $t6, 2     #use $t6 to multiply j by 4, store $t4
          add $t4, $s2, $t1   #add (4*j) to addr D, store back in $t4
          add $t5, $t0, $t1   #add i and j
          lw  $t6, 0($t5)     #load result of i + j into $t6
          sw  $t6, 0($t4)     #store result of i + j in D[4 * j]($t4)
          j LOOP2             #jump back to beginning of j-Loop
DEST:
     … do whatever after done looping …
```

NEXT PAGE

Kaleb J Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Problem 7 [20 pts]

Translate function f into MIPS assembly language:

```
int f ( int a , int b , int c , int d ){
     return func ( func (a , b ) , c + d );
}
```

As needed, use registers $t0 through $t7, beginning with the lower-numbered registers first. Assume there exists a function func with declaration:

```
int func ( int a , int b );
```

You do not need to write function func but should call it, using label func.

Int a	Int b	Int c	Int d	Func: int a	Func: int b
$a0	$a1	$a2	$a3	$t4	$t5
F: int a	F: int b	F: int c	F: int d	Return val func:	Return:
$t0	$t1	$t2	$t3	$t6	$v0
F: c+d				Branch condition:	
$t7				$t8	

f:

```
     sw $a0, 0($t0)       #load arg "a" in $t0 for manipulation
     sw $a1, 0($t1)       #load arg "b" in $t1 for manipulation
     sw $a2, 0($t2)       #load arg "c" in $t2 for manipulation
     sw $a3, 0($t3)       #load arg "d" in $t3 for manipulation
     add $t7, $t2, $t3    #add "c+d" store in $t7
     sw $a0, 0($t4)       #prepare a copy of "a" for "func"
     sw $a1, 0($t5)       #prepare a copy of "b" for "func"
     j func               #jump to "func", $t6 updated, continue
temp1:
     sw $t6, 0($t4)       #arg func(a) is now result of func(a,b)
     sw $t7, 0($t5)       #arg func(b) is now result of "c+d"
     j func               #jump "func", $t6 is updated again, continue
```

func:

...do something with $t4 and $t5....
...update $t6 as $t4 and $t5 are manipulated...

```
sw $v0, 0($t6)       #save all our hard work in $v0.
```

...I would have a slt and beq in here where if we have performed this twice, jump to somewhere other than "temp1"...

j temp1

Kaleb J Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Problem 8 [10 pts]

(a) Assume 185 and 122 are unsigned 8-bit decimal integers. Calculate 185+122 and show your work. Is there overflow, underflow, or neither?

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
Arg 1	1	0	1	1	1	0	0	1
Arg 2	0	1	1	1	1	0	1	0
Result	0	0	1	1	0	0	1	1

1 <-- fell off the end

we lost a 1, it fell off the end when carrying so we have overflow.

(b) Assume 185 and 122 are unsigned 8-bit decimal integers. Calculate 185−122 and show your work. Is there overflow, underflow, or neither?

1-0 = borrowed from the left
/0 = borrowing from this number

	/0	1-0 /0	/0 1-0	/0 1-0	/0 1-0	1-0 /0	1-0	
Arg 1	1	0	1	1	1	0	0	1
Arg 2	0	1	1	1	1	0	1	0
Result	0	0	1	1	1	1	1	1

Our result is: 32 + 16 + 8 + 4 + 2 + 1 = **63** and 185 – 122 = 63. Furthermore we had no overflow or underflow.

(c) Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate 185+122 and show your work. Is there overflow, underflow, or neither?

Cannot show our work. 185 is too large of a number to be stored in 7 bits.
Sign magnitude allocates the most significant bit to be a sign representation therefore the largest number we can store in 8 bits is 64+32+16+8+4+2+1 = 127 See table Below:

+/-	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0/1	64	32	16	8	4	2	1

In this case I guess we could say there's OVERFLOW as we had overflow just in attempting to calculate the value...

PART “d” ON LAST PAGE

Kaleb J Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

(d) Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate 185−122 and show your work. Is there overflow, underflow, or neither?

Cannot show our work. 185 is too large of a number to be stored in 7 bits.
Sign magnitude allocates the most significant bit to be a sign representation therefore the largest number we can store in 8 bits is 64+32+16+8+4+2+1 = 127 See table Below:

+/-	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0/1	64	32	16	8	4	2	1

In this case I guess we could say there's OVERFLOW as we had overflow just in attempting to calculate the value...