Brandon Marcoux
EEL 4768 Spring 2017
Abichar

Homework #3

Question 1: Translate the code below to MIPS64 assembly language.

Addresses: (a @ 400) (b @ 408) (result @ 416)

```
long int a, b;      // 64-bit
double result;      // 64-bit
...
result = (double) (a+b) / 12.22;
```

```
LD        R1, 400(R0)    # load double word (64-bit) into R1 from address 400 (a).
LD        R2, 408(R0)    # load b into R2
DADD      R3, R1, R2     # add a and b using DADD
DMTC1     R3, F0         # move the addition result double word into coprocessor 1 (this is
                         # the FPR)
CVT.D.L   F0, F0         # convert to double floating point format
LI.D      F1, 12.22      # load immediate as double precision
DIV.D     F2, F0, F1     # divide division result by 12.22
S.D       F2, 416(R0)    # store result to memory.
```

Question 2: Translate this code to MIPS64 assembly language.

Addresses: (a @ 400) (b @ 408) (result @ 416)

```
float a, b, result;     // 32-bit
...
result = (a+b) / 12.22;
```

```
L.S       F0, 400(R0)    # load single precision float from address 400 (a)
L.S       F1, 408(R0)    # load b into F1
LI.S      F2, 12.22      # load 12.22 as a single precision into F2

ADD.S     F3, F0, F1     # floating point single precision add a and b
DIV.S     F4, F3, F2     # floating point single precision divide (a+b) / 12.22
S.S       F4, 416(R0)    # store the division result into the memory with single precision
```

Question 3: Translate this code to MIPS64 assembly language.
Addresses: (a @ 400) (b @ 408) (result @ 416)

```
long int a, b, result;      // 64-bit
…
if( a<b || a==33 )
  result = 1;
else result = 0;
```

```
LD      R1, 400(R0)     # load a into R1 from memory as a double
LD      R2, 408(R0)     # load b into R2
LDI     R3, 33          # load 33 int R3

SLT     R4, R1, R2      # set R4 to 1 if a<b, 0 otherwise
BNE     R4, R0, True    # if R4 is 1 branch to true
BEQ     R3, R1, True    # branch if a is equal to 33 to label true

# if the pc makes it here then it has not branched and neither condition is met
SD      R0, 416(R0)     # store zero as result in the memory
J       End             # terminate

True:
DADDI   R5, R0, 1       # store 1 in R5 by adding an immediate double word to 0
SD      R5, 416(R0)     # store 1 as result into the memory

End:
```

Question 4: Translate this code to MIPS64 assembly language.
Addresses: (a @ 400) (b @ 408) (result @ 416)

```
float a, b, result;      // 32-bit
…
if( a<b || a==33 )
  result = 1;
else result = 0;
```

```
L.S     F1, 400(R0)     # store a as single precision in F1
L.S     F2, 408(R0)     # store b
LI.S    F3, 33          # load single precision immediate 33

C.LT.S  F1, F2          # compare single precision F1 < F2 ?
BC1T    True            # branch to true if a < b
C.EQ.S  F1, F3          # a == 33 ?
BC1T    True            # branch to true if a == 33
```

```
# if we have made it to this point in the code neither condition is met, store result as 0
LI.S        F4, 0          # load 0 as an immediate for storage
S.S         F4, 416(R0)    # store 0 at result memory
J           End            # terminate

True:
LI.S        F4, 1          # load 1 as an immediate for storage
S.S         F4, 416(R0)    # store 1 at result memory

End:
```

Question 5: The memory contains an array of 100 double-precision floating-point numbers. The start address of the array is in register R1.
Write a MIPS64 code that loops over the array, finds all the negative values and changes them to zero. Count how many values have been changed to zero and place the answer in register R31.

```
DADD        R31, R0, R0    # make sure accumulator R31 is 0 before we start counting
DADD        R2, R0, 100    # set loop counter to 100 before looping
LI.D        F1, 0          # load 0 into F1 with double precision

Loop:
L.D         F2, 0(R1)      # load a value from the array
C.LT.D      F2, F1         # element < 0 ?
BC1F        False          # element is greater than 0

# element is less than one
S.D         F1, 0(R1)      # change negative element to 0
DADDI       R31, R31, 1    # increment R31

False:                     # element is not less than one, leave it alone

DADDI       R1, R1, 8      # increment array pointer to next element
DADDI       R2, R2, -1     # decrement counter
BGEZ        R2, Loop       # continue in loop while R2 is greater than 0
```