Name: Cicelia Siu

CS 219.1001 – Assignment #5

Purpose: Become familiar with the MIPS architecture instruction formats, floating point

data representation issues, and multiplication algorithms

Points: 100

Assignment:

1. According to the text (Ch 2), what are the three (3) underlying design principals? [8 pts]
2. Simplicity favors regularity.
3. Smaller is faster.
4. Good design demands good compromises.
5. List the two architectural models and describe the key difference each. Include which is the most commonly used. Note, not in text. [5 pts]
6. Von Neumann architecture (most commonly used) - data and instructions are stored in the same memory
7. Harvard architecture - data and instruction are in separated memory locations

1. List the six possible fields (include the name and meaning) in a MIPS instruction? [8 pts]
2. op - basic operation of the instruction (aka opcode)
3. rs - 1st register source operand
4. rt - 2nd register source operand
5. rd - destination register
6. shamt - shift amount for logicals
7. funct - function (variant in op field)
8. What are the three basic instruction formats? Show the layout/fields for each. [3 pts]
9. r-format

field: op / rs / rt / rd / shamt / funct

bits: 6 / 5 / 5 / 5 / 5 / 6

1. i-format

field: op / rs / rd / address/constant

bits: 6 / 5 / 5 / 16

1. j-format

field: op / address

bits: 6 / 26

address is the relative address

1. Referring to mips.s program from assignment #3
   1. what is the opcode for the “add $s0, $s0, $t5” instruction?

Show result in binary, grouped by field. [3 pts]

0x020d8020

0000 0010 0000 1101 1000 0000 0010 0000

**op | rs | rt | rd | shamt | funct |**

**000000 10000 01101 10000 00000 100000**

* 1. what is the value (in binary and decimal) for rd field and which register (name) is indicated for that field? [2 pt]

**rd -> 100002 -> 1610 -> Register 16 = a saved register aka $s0**

1. Given the hex opcode 0x00954020 what is the instruction (including register names)? Show field values.

[5 pts]

0000 00|00 100|1 0101| 0100 0|000 00|10 0000

op - > 000000 -> add/sub

rs - > 00100 -> 4 -> $4 -> $a0

rt - > 10101 -> 21 -> $21 ->$s5

rd - > 01000 -> 8 -> $8 -> $t0

shamt - > 00000 ->N/A

funct - > 100000 -> 32 -> 0x20 -> add

add $t0, $a0, $s5 = **add $8, $4, $21**

1. Given that x = 0000 0000 0000 0000 0000 0000 0101 10112 and y= 0000 0000 0000 0000 0000 0000 0000 11012 representing two's compliment signed integers, perform the following operations showing all work. [3 pts each]
   1. x + y

carries: 11 111

x = 0000 0000 0000 0000 0000 0000 0101 1011

y = 0000 0000 0000 0000 0000 0000 0000 1101

+----------------------------------------------------------------

**0000 0000 0000 0000 0000 0000 0110 1000 -> aka 104**

* 1. x – y → (x + (-y))

-y -> y = 0000 0000 0000 0000 0000 0000 0000 1101

flip = 1111 1111 1111 1111 1111 1111 1111 0010

add 1= 1

+----------------------------------------------------------------

1111 1111 1111 1111 1111 1111 1111 0011 ->aka -13

carries:

1 1111 1111 1111 1111 1111 1111 111 11

x = 0000 0000 0000 0000 0000 0000 0101 1011

-y = 1111 1111 1111 1111 1111 1111 1111 0011

+----------------------------------------------------------------

**0000 0000 0000 0000 0000 0000 0100 1110 -> aka 78**

1. Given that x = 1111 1111 1111 1111 1001 1011 0011 11102 and

y= 0000 0000 0000 0000 0000 0110 0010 01012 representing two's compliment signed integers, perform the following operations showing all work. [3 pts each]

* 1. x + y

carries:

11 11 111 1

x = 1111 1111 1111 1111 1001 1011 0011 1110 -> -25794

y = 0000 0000 0000 0000 0000 0110 0010 0101 ->1573

+----------------------------------------------------------------

**11111 1111 1111 1111 1010 0001 0110 0011 -> aka -24221**

* 1. x – y → (x + (-y))

-y -> y = 0000 0000 0000 0000 0000 0110 0010 0101

flip = 1111 1111 1111 1111 1111 1001 1101 1010

add 1= 1

+----------------------------------------------------------------

1111 1111 1111 1111 1111 1001 1101 1011 ->aka -1573

carries:

1 1111 1111 1111 1111 1111 111 1111 11

x = 1111 1111 1111 1111 1001 1011 0011 1110 -> -25794

y = 1111 1111 1111 1111 1111 1001 1101 1011 -> -1573

+----------------------------------------------------------------

**1111 1111 1111 1111 1001 0101 0001 1001 -> aka -27367**

1. Briefly describe the what is meant by overflow and underflow conditions that might be produced after a floating point operation (for the exponent). [4 pts]

overflow - when a positive exponent becomes too large to fit the exponent field usually happens during arithmetic operations

underflow - when a negative exponent becomes too large to fit the exponent field usually happens during arithmetic operations

from pg 198

1. What are the basic elements/steps required for floating point addition?

Note, answer with short paraphrased description of each step. [4 pts]

1. Align decimals by shifting to match the significand of the smaller exponent to the larger one. Round if needed to represent 4 decimal digits.
2. Add with the decimals aligned.
3. Adjust the sum to the general scientific notation format (normalize) and keep track of the overflow and underflow.
4. Round the number if needed to represent 4 digits. Repeat step 3 if sum isn’t the general scientific notation format.

from pg 203

1. What are the basic elements/steps required for floating point multiplication?

Note, answer with short paraphrased description of each step. [4 pts]

1. Add exponents of the operands together. If the representation is too large subtract bias from the sum.
2. Multiply significands.
3. Adjust the product to the general scientific notation format (normalize). Check for underflow and overflow.
4. Round the number if needed to represent 4 digits.
5. Set the sign to negative if the operands’ signs are different, else, set the sign to positive.

from pg 207

1. Given the below data declarations (which are in a provided main):

.data

# -----

# Single precision

fval1: .float 9.75

fval2: .float 5.5

fval3: .float 12.25

fval4: .float 3.6

fans1: .float 0.0

fans2: .float 0.0

fval: .float 0.001

fsum: .float 0.0

# -----

# Double precision

dval1: .double 9.75

dval2: .double 5.5

dval3: .double 12.25

dval4: .double 3.6

dans1: .double 0.0

dans2: .double 0.0

dval: .double 0.001

dsum: .double 0.0

Write a simple MIPS program to compute the following floating point formulas:

# fans1 = fval1 + fval2

# fans2 = fval3 + fval4

# fsum = 0.0

# do 1000 times

# fsum = fsum + fval

# dans1 = dval1 + dval2

# dans2 = dval3 + dval4

# dsum = 0.0

# do 1000 times

#

dsum = dsum + dval

Show the displayed results of the program (i.e,. console output) and explain. [10 pts]

For this question:

1. Submit a copy of the console output (from the log file).
   1. It is not necessary to include the program source or text segment information in the log file.
2. Submit a copy of the source file.

13) Perform the following multiplication problems using Booths algorithm (see handout). Assume an 8-bit machine. [8 pts each]

1. 5 × 10

multiplicand = 0000 0101

-multiplicand = 1111 1011

multiplier = 0000 1010

|  |  |  |  |
| --- | --- | --- | --- |
| **step** | **action** | **multiplicand** | **product** |
| 0 | initial | 0000 0101 | 0000 0000 0000 1010 0 |
| 1 | 00 -> no op |  | 0000 0000 0000 1010 0 |
|  | rs |  | 0000 0000 0000 0101 0 |
| 2 | 10 -> ls - mc |  | 1111 1011 0000 0101 0 |
|  | rs |  | 1111 1101 1000 0010 1 |
| 3 | 01 -> ls + mc |  | 0000 0010 1000 0010 1 |
|  | rs |  | 0000 0001 0100 0001 0 |
| 4 | 10 -> ls - mc |  | 1111 1100 0100 0001 0 |
|  | rs |  | 1111 1110 0010 0000 1 |
| 5 | 01 -> ls + mc |  | 0000 0011 0010 0000 1 |
|  | rs |  | 0000 0001 1001 0000 0 |
| 6 | 00 -> no op |  | 0000 0001 1001 0000 0 |
|  | rs |  | 0000 0000 1100 1000 0 |
| 7 | 00 -> no op |  | 0000 0000 1100 1000 0 |
|  | rs |  | 0000 0000 0110 0100 0 |
| 8 | oo -> no op |  | 0000 0000 0110 0100 0 |
|  | rs |  | **0000 0000 0011 0010** 0 |
|  |  |  |  |

1. 6 × 7

multiplicand = 0000 0110

-multiplicand = 1111 1010

multiplier = 0000 0111

|  |  |  |  |
| --- | --- | --- | --- |
| **step** | **action** | **multiplicand** | **product** |
| 0 | initial | 0000 0110 | 0000 0000 0000 0111 0 |
| 1 | 10 -> ls - mc |  | 1111 1010 0000 0111 0 |
|  | rs |  | 1111 1101 0000 0011 1 |
| 2 | 11 -> no op |  | 1111 1101 0000 0011 1 |
|  | rs |  | 1111 1110 1000 0001 1 |
| 3 | 11 -> no op |  | 1111 1110 1000 0001 1 |
|  | rs |  | 1111 1111 0100 0000 1 |
| 4 | 01 -> ls+mc |  | 0000 0101 0100 0000 1 |
|  | rs |  | 0000 0010 1010 0000 0 |
| 5 | 00 -> no op |  | 0000 0010 1010 0000 0 |
|  | rs |  | 0000 0001 0101 0000 0 |
| 6 | 00 -> no op |  | 0000 0001 0101 0000 0 |
|  | rs |  | 0000 0000 1010 1000 0 |
| 7 | 00 -> no op |  | 0000 0000 1010 1000 0 |
|  | rs |  | 0000 0000 0101 0100 0 |
| 8 | 00 -> no op |  | 0000 0000 0101 0100 0 |
|  | rs |  | **0000 0000 0010 1010** 0 |
|  |  |  |  |

1. 17 × -6

multiplicand = 0001 0001

-multiplicand = 1110 1111

multiplier = 1111 1010

|  |  |  |  |
| --- | --- | --- | --- |
| **step** | **action** | **multiplicand** | **product** |
| 0 | initial | 0001 0001 | 0000 0000 1111 1010 0 |
| 1 | 00 -> no op |  | 0000 0000 1111 1010 0 |
|  | rs |  | 0000 0000 0111 1101 0 |
| 2 | 10 -> ls - mc |  | 1110 1111 0111 1101 0 |
|  | rs |  | 1111 0111 1011 1110 1 |
| 3 | 01 -> ls+mc |  | 0000 1000 1011 1110 1 |
|  | rs |  | 0000 0100 0101 1111 0 |
| 4 | 10 -> ls- mc |  | 1111 0011 0101 1111 0 |
|  | rs |  | 1111 1001 1010 1111 1 |
| 5 | 11 -> no op |  | 1111 1001 1010 1111 1 |
|  | rs |  | 1111 1100 1101 0111 1 |
| 6 | 11 -> no op |  | 1111 1100 1101 0111 1 |
|  | rs |  | 1111 1110 0110 1011 1 |
| 7 | 11 -> no op |  | 1111 1110 0110 1011 1 |
|  | rs |  | 1111 1111 0011 0101 1 |
| 8 | 11 -> no op |  | 1111 1111 0011 0101 1 |
|  | rs |  | **1111 1111 1001 1010** 1 |
|  |  |  |  |

1. -14 × 4

multiplicand = 1111 0010

-multiplicand = 0000 1110

multiplier = 0000 0100

|  |  |  |  |
| --- | --- | --- | --- |
| **step** | **action** | **multiplicand** | **product** |
| 0 | initial | 11111 0010 | 0000 0000 0000 0100 0 |
| 1 | 00 -> no op |  | 0000 0000 0000 0100 0 |
|  | rs |  | 0000 0000 0000 0010 0 |
| 2 | 00 -> no op |  | 0000 0000 0000 0010 0 |
|  | rs |  | 0000 0000 0000 0001 0 |
| 3 | 10 -> ls - mc |  | 0000 1110 0000 0001 0 |
|  | rs |  | 0000 0111 0000 0000 1 |
| 4 | 01 -> ls + mc |  | 1111 1001 0000 0000 1 |
|  | rs |  | 1111 1100 1000 0000 0 |
| 5 | 00 -> no op |  | 1111 1100 1000 0000 0 |
|  | rs |  | 1111 1110 0100 0000 0 |
| 6 | 00 -> no op |  | 1111 1110 0100 0000 0 |
|  | rs |  | 1111 1111 0010 0000 0 |
| 7 | 00 -> no op |  | 1111 1111 0010 0000 0 |
|  | rs |  | 1111 1111 1001 0000 0 |
| 8 | 00 -> no op |  | 1111 1111 1001 0000 0 |
|  | rs |  | **1111 1111 1100 1000** 0 |
|  |  |  |  |