

# CS2214 COMPUTER ARCHITECTURE & ORGANIZATION FALL 2009

## EXAM I ANSWERS

1) a) The table showing the values of registers and memory locations used by the code is as follows :

| Instruction     | PC     | R8       | R9                 | R10 | M[10000000] | M[10004000]        | Memory Accesses |
|-----------------|--------|----------|--------------------|-----|-------------|--------------------|-----------------|
| Initial         | 400000 | 10000000 | ?                  | ?   | 7           | ?                  | -----           |
| LW R9, 0(R8)    | 400004 | NS       | 7                  | NS  | NS          | NS                 | 2 : IF & DR     |
| SLT R10, R9, R0 | 400008 | NS       | NS                 | 0   | NS          | NS                 | 1 : IF          |
| BEQ R10, R0, 2  | 400014 | NS       | NS                 | NS  | NS          | NS                 | 1 : IF          |
| SLL R9, R9, 1   | 400018 | NS       | (14) <sub>10</sub> | NS  | NS          | NS                 | 1 : IF          |
| SW R9, 4000(R8) | 40001C | NS       | NS                 | NS  | NS          | (14) <sub>10</sub> | 2 : IF & DW     |

This code takes the absolute value of a memory location and stores back to the same location. Then, it multiplies this positive value by 2 and stores to another location.

b) The first five instructions take the absolute value of a memory location and store back to the same memory location. This is the **ABSM** instruction or “Absolute value Memory” instruction :

Syntax : ABSM Disp(Rs)

Semantics :  $M[Rs + Disp^+] \leftarrow |M[Rs + Disp^+]|$

Format, etc. :

- It uses the **I** format since “Disp” is used. Rt is not used
- We make **three** memory access for the new instruction :
  - One to fetch the instruction
  - One to read the memory location for data
  - One to write the data to the memory if necessary (if the data read was negative)

The original code is rewritten with the ABSM below :

|               |             |              |  |
|---------------|-------------|--------------|--|
| <b>400000</b> | <b>ABSM</b> | <b>0(R8)</b> | <b># Take the absolute value of the data element in memory</b> |
| 400004        | LW          | R9, 0(R8)    | # Read the same memory location which is positive now          |
| 400008        | SLL         | R9, R9, 1    | # Multiply the number by 2                                     |
| 40000C        | SW          | R9, 4000(R8) | # Store the result in memory location 10004000                 |

The ABSM instruction saves **four** original instructions. But, we **cannot** avoid a read from the memory after the ABSM. This is a drawback of CISC A/L instructions that access the memory : If a memory location is operated on multiple times, multiple memory accesses are needed. In the case of RISC, we read the memory once, operate on the data multiple times and eventually write back to the memory, reducing the number of memory accesses.

2) a) The missing instruction should be a BEQ instruction to **skip** the ORI if the rightmost bit should stay 0 :

|        |     |             |  |
|--------|-----|-------------|--|
| 400000 | SLT | R9, R8, R0  | # R8 is less than R0, store 1 on R9, since F0F0F0F0 < 0                |
| 400004 | SLL | R10, R8, 1  | # Shift left R8 by 1 and place a 0 on the rightmost bit                |
| 400008 | BEQ | R9, R0, 1   | # Since R9 is not equal to R0, do <b>not</b> skip the next instruction |
| 40000C | ORI | R10, R10, 1 | # Place a one on the rightmost bit of R10 since R8 is negative         |

b) The execution table is as follows :

| Instruction     | PC     | R8       | R9 | R10      | Mem. Acc. |
|-----------------|--------|----------|----|----------|-----------|
| Initial         | 400000 | F0F0F0F0 | ?  | ?        | -----     |
| SLT R9, R8, R0  | 400004 | NS       | 1  | NS       | 1 : IF    |
| SLL R10, R8, 1  | 400008 | NS       | NS | E1E1E1E0 | 1 : IF    |
| BEQ R9, R0, 1   | 40000C | NS       | NS | NS       | 1 : IF    |
| ORI R10, R10, 1 | 400010 | NS       | NS | E1E1E1E1 | 1 : IF    |

c) Format, etc. :

- It uses the **R** format since we do **not** need a Disp, Imm or Offset. Also, this is an A/L instruction. Rd and Shamt are not used.
- It has **three** arguments.
  - Rt is a destination register explicitly specified by the instruction : Register addressing mode
  - Rs is a source register explicitly specified by the instruction : Register addressing mode
  - Number "1" is an implied data element : Implied addressing mode
- We make **one** memory access for the new instruction :
  - One to fetch the instruction

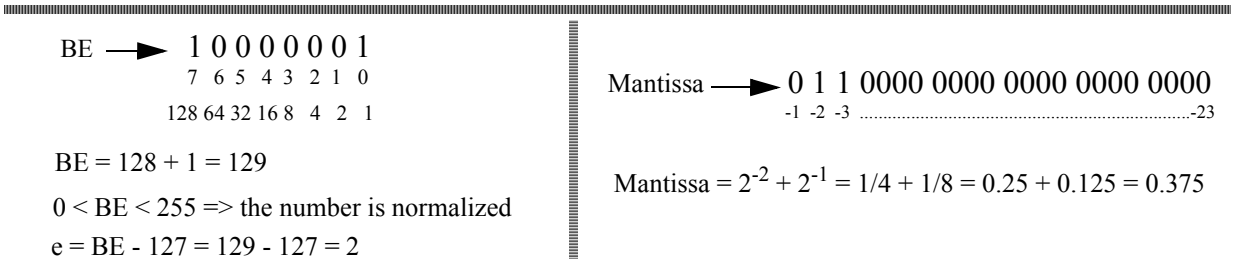
d) The table showing the values of registers and memory locations used by the code is as follows :

| Instruction                       | PC     | R8 | R9                 | R10 | Mem. Acc. | Instruction                       | PC     | R8 | R9                 | R10 | Mem. Acc. |
|-----------------------------------|--------|----|--------------------|-----|-----------|-----------------------------------|--------|----|--------------------|-----|-----------|
| Initial                           | 400000 | 4  | ?                  | 3   | -----     | ADDI R10, R10, (-1) <sub>10</sub> | 40000C | NS | NS                 | 1   | 1 : IF    |
| ADD R9, R8, R0                    | 400004 | NS | 4                  | NS  | 1 : IF    | BNE R10, R0, (-3) <sub>10</sub>   | 400004 | NS | NW                 | NS  | 1 : IF    |
| RL1 R9, R9                        | 400008 | NS | 8                  | NS  | 1 : IF    | RL1 R9, R9                        | 400008 | NS | (32) <sub>10</sub> | NS  | 1 : IF    |
| ADDI R10, R10, (-1) <sub>10</sub> | 40000C | NS | NS                 | 2   | 1 : IF    | ADDI R10, R10, (-1) <sub>10</sub> | 40000C | NS | NS                 | 0   | 1 : IF    |
| BNE R10, R0, (-3) <sub>10</sub>   | 400004 | NS | NS                 | NS  | 1 : IF    | BNE R10, R0, (-3) <sub>10</sub>   | 400010 | NS | NS                 | NS  | 1 : IF    |
| RL1 R9, R9                        | 400008 | NS | (16) <sub>10</sub> | NS  | 1 : IF    |                                   |        |    |                    |     |           |

This code rotates a register by as many times as another register indicates. In this example, it rotates register R8 by R10 times and stores in R9.

3) The IEEE-754 single-precision floating-point number :

The number is a **negative** number, since the **sign** bit of the FP number is **one** →  $\underbrace{1}_{\text{Sign}}$   $\underbrace{100\ 0000\ 1}_{\text{BE}}$   $\underbrace{011\ 0000\ 0000\ 0000\ 0000\ 0000}_{\text{Mantissa}}$



$$(1\ 10000001\ 011000000000000000000000)_{\text{IEEE-754}} = (-1.375 \times 2^2)_{10} = (1.375 * 4)_{10} = (-5.5)_{10}$$