**Homework 4 Answer Sheet**

Please state the name, SID and email of each member of your group.

| member | name | SID | email |
| --- | --- | --- | --- |
| #1 | DARMAWAN Ryan Dyson | 57883349 | rdarmawan2-c@my.cityu.edu.hk |
| #2 | GUNAWAN Cayleen | 57903224 | cayleen.g@my.cityu.edu.hk |

Logisim Version: Logisim 2.7.1

OS (Window/MAC/Linux): Windows

1. Do all members make significant contributions to this homework? If not, please specify the details.

Yes!

1. Please explain how many types of instructions are supported in your processor, and explain the format of each type of instructions (e.g., which bits are used as the operation or function code, which bits are used to index the 1st, 2nd or 3rd operand, and which bits are used to store the immediate number). You can draw figures to better explain your answer.

X = 57883349 + 57903224 = 115786573

Y = X % 7 = 115786573 % 7 = 0

| R-Type | I-Type | J-Type | M-Type (memory instruction) |
| --- | --- | --- | --- |
|  |  |  |  |
| o Bits(0~2): write (destination) register  o Bits(3~5): read register 1  o Bits(6~8): read register 2  o Bits(9~11): Opcode  o Bits(12~15): Opcode | o Bits(0~5): Immediate value  o Bits(6~8): read register 1  o Bits(9~11): destination register  o Bits(12~15): Opcode | o Bits(11~0): Target value  o Bits(12~15): Opcode | o Bits(5~0): Reserved (000000, in our case)  o Bits(11~9): Source value  o Bits(11~9): Target value  o Bits(12~15): Opcode |

1. Please explain the format of each instruction (including the format of this instruction and its operation codes, and other information if needed).

**OPcode**

o We only need to accommodate 16 instructions, so we only need 4 bits to accommodate all instruction set.

o Highest immediate value is 10 and lowest immediate value is -1. We use 2’s complement, so we need 6 bits.

o Therefore, immediate numbers for I-type, we need to retrieve the I-type OPcode to 4-bits

o Therefore, for consistent design, we will ignore the last 3-bits

| li | I-type Instruction  opcode: 0000 |
| --- | --- |
| add | R-type Instruction  opcode: 0001xxx  note that, xxx doesnt matter |
| and | R-type Instruction  opcode: 0010xxx  note that, xxx doesnt matter |
| or | R-type Instruction  opcode: 0011xxx  note that, xxx doesnt matter |
| load | M-typeInstruction  opcode: 0100 |
| store | M-type Instruction  opcode: 0101 |
| move | R-type Instruction  opcode: 0110xxx  note that, xxx doesnt matter |
| addi | I-type Instruction  opcode: 0111 |
| andi | I-type Instruction  opcode: 1000 |
| ori | I-type Instruction  opcode: 1001 |
| ble | J-type Instruction  opcode: 1010 |
| bne | J-type Instruction  opcode: 1011 |
| jump | J-type Instruction  opcode: 1100 |
| call | J-type Instruction  opcode: 1101 |
| rtn | J-type Instruction  opcode: 1110 |
| halt | J-type Instruction  opcode: 1111 |

* note that for R-type, xxx doesnt matter in our design, we will use 000. xxx can be later extended to include shamt or funct however, that is not required in our design.

1. Fill the following tables with the machine codes of each instruction of the testing programs:

**Test program 1:**

| instruction | machine code (binary) | machine code (hex) |
| --- | --- | --- |
| li $r1, 1 | 0000 000 000 000001 | 0x0001 |
| li $r2, 2 | 0000 000 001 000010 | 0x0042 |
| li $r3, 10 | 0000 000 010 001010 | 0x008A |
| add $r2, $r1, $r2 | 0001 000 001 000 001 | 0x1041 |
| ble $r2, $r3, -1 | 1010 001 010 111111 | 0xA2BF |
| halt | 1111 000000000000 | 0xF000 |

**Test program 2:**

| instruction | machine code (binary) | machine code (hex) |
| --- | --- | --- |
| li $r1, 6 | 0000 000 000 000110 | 0x0006 |
| li $r2, 5 | 0000 000 001 000101 | 0x0045 |
| andi $r3, $r1, 3 | 1000 000 010 000011 | 0x8083 |
| ori $r4, $r3, 8 | 1001 011 010 001000 | 0x9688 |
| halt | 1111 000000000000 | 0xF000 |

**Test program 3:**

| instruction | machine code (binary) | machine code (hex) |
| --- | --- | --- |
| li $r1, 6 | 0000 000 000 000110 | 0x0006 |
| li $r2, 5 | 0000 000 001 000101 | 0x0045 |
| and $r3, $r1, $r2 | 0010 000 000 001 010 | 0x200A |
| li $r8, 0 | 0000 000 111 000000 | 0x01C0 |
| store $r3, $r8 | 0101 111 010 000000 | 0x5E80 |
| or $r4, $r1, $r2 | 0011 000 001 000 011 | 0x3043 |
| li $r8, 1 | 0000 000 111 000001 | 0x01C1 |
| store $r4, $r8 | 0101 111 011 000000 | 0x5EC0 |
| li $r8, 1 | 0000 000 111 000001 | 0x01C1 |
| load $r7, $r8 | 0100 111 110 000000 | 0x4F80 |
| halt | 1111 000000000000 | 0xF000 |

**Test program 4:**

| instruction | machine code (binary) | machine code (hex) |
| --- | --- | --- |
| li $r1, 6 | 0000 000 000 000110 | 0x0006 |
| li $r2, 4 | 0000 000 001 000100 | 0x0044 |
| call 7 | 1101 000000000111 | 0xD007 |
| move $r4, $r3 | 0110 000 000 010 011 | 0x6013 |
| li $r1, 7 | 0000 000 000 000111 | 0x0007 |
| li $r2, 8 | 0000 000 001 001000 | 0x0048 |
| call 3 | 1101 000000000011 | 0xD003 |
| move $r5, $r3 | 0110 000 000 010 100 | 0x6014 |
| jump 3 | 1100 000000000011 | 0xC003 |
| add $r3, $r1, $r2 | 0001 000 001 000 010 | 0x1042 |
| rtn | 1110 000000000000 | 0xE000 |
| halt | 1111 000000000000 | 0xF000 |