==========

Assignment 12
Group 10

Apurv Kumar Shubham Sharma
14CS10006 14CS30034

==========

*All instructions are of 16 bits.
*Since all the general registers are 8 so they can be represented using only 3 bits

The general instruction format is as follows :-

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |Op-code | Op code-Type | dst(operand 1) | mode for operand 2 | src1(operand 2) | src2(operand 2) |
| D2 (next line for operand 2) [Depends upon the mode of operand 2] |

Here ,
+op-code : The operation type out of 3 - Load and store, ALU, jump instructions and subroutine instructions.
+op-code type : The type of Load and store, ALU, jump instructions and subroutine instructions. For example, for op-code=ALU, add and sub will have different values.
+dst(operand 1) : The destination register or operand 1. It can have only one mode i.e "reg" as per the problem statement.
+mode for operand2 : Addressing mode for operand 2.
+src1(operand 2) : The second operand is the source and can have 6 modes.
+src2(operand 2) : This along with src1 define the operand 2 for our instruction.

+D2 : The immediate/offset value for operand 2 is stored in the next 16 bits. Its use solely depends upon the mode of addressing of the operand 2.

I) Load and store : ld, st with a 16-bit displacement (depending upon the addressing mode).
The dst operand for the load instruction and the src operand for the store instruction is always in register mode.
op-code = 00

A) Load -- General assembly instruction format:
ld dst src
op-code type = 000

>>Load Immediate (li)
++++++++++++++++++++++++++++++++li r5, #100 // r5 ← 100 i.e., r5 ← M[PC]; PC ← PC+2, where M[PC] is
// the second word of the instruction
// (since PC is incremented in the fetch phase)
// -- load immediate

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |

FORMAT: |00 | 000 | dst(operand 1) | 00 | src1(operand 2) | src2(operand 2) |
| IMMEDIATE |

>>Load Register (lr)
++++++++++++++++++++++++++++++++lr r5, r7 // r5 ← r7 -- load register

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |00 | 000 | dst(operand 1) | 01 | src1(operand 2) | X |

>>Load with Base Indexed Addressing (lx)
++++++++++++++++++++++++++++++++lx r5, 10(r1, r7) // r5 ← M[r1+ r7 + 10] i.e.,
// r5 ← M[r1 + r7 +M[PC]]; PC ← PC+2 -- load indexed

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |00 | 000 | dst(operand 1) | 10 | src1(operand 2) | src2(operand 2) |
| D2 (next line for operand 2) |

>>Load with Memory Indirect Addressing(ldn)
++++++++++++++++++++++++++++++++ldn r5, @10(r1, r7) // r5 ← M[M[r1+ r7 + 10]] i.e.,
// r5 ← M[M[r1+ r7 + M[PC]]]; PC ← PC+2
// -- load indirect

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |00 | 000 | dst(operand 1) | 11 | src1(operand 2) | src2(operand 2) |
| D2 (next line for operand 2) |

B) Store -- General assembly instruction format:
st dst src // only two addressing modes are needed
// -- Indexed and Indirect
op-code type = 001

>>Store with Base Indexed Addressing (stx)
++++++++++++++++++++++++++++++++stx -5(r2), r3 // M[r2-5] ← R3 -- base addressing

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |00 | 001 | dst(operand 1) | 00 | src1(operand 2) | src2(operand 2) |
| D2 (next line for operand 2) |

>>Store with Memory Indirect Addressing (sdn)
++++++++++++++++++++++++++++++++stn @-5(r2), r3 // M[M[r2-5]] ← R3 i.e.,
// M[M[r2 + M[PC]]]; PC ← PC+2 -- indirect

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |00 | 001 | dst(operand 1) | 01 | src1(operand 2) | src2(operand 2) |
| D2 (next line for operand 2) |

==========
==========

II) Arithmetic and logical instructions: add, sub, and, or, mns, cmp using two's complement arithmetic
The dst operand for these instructions is always in register mode.

General assembly instruction format:
op-code dst/src 1 src 2

op-code=01

All addressing modes supported for src 2 for two operand instructions.

A) Add
op-code type = 000

>>Add Immediate (addi)
++addi r1, #43 // r1 ← r1 + 43 add immediate

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 000 | dst(operand 1) | 00 | src1(operand 2) | src2(operand 2) |
| IMMEDIATE |
>>Add Register (addr)
++addr r1, r2 // r1 ← r1 + r2 add register

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 000 | dst(operand 1) | 01 | src1(operand 2) | X |

>>Add with Base Indexed Addressing (addx)
++addx r2,10(r1, r7) // r2 ← r2 + M[r1+r7+10]
//r2 ← r2 + M[r1 + r7 +M[PC]]; PC ← PC+2 add indexed

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 000 | dst(operand 1) | 10 | src1(operand 2) | src2(operand 2) |
| D2 (next line for operand 2) |
>>Add with Memory Indirect Addressing (addn)
++addn @-120(r2, r6) // r4 ← r4 + M(M(r2 + r6 -120)) add indirect

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 000 | dst(operand 1) | 11 | src1(operand 2) | src2(operand 2) |
| D2 (next line for operand 2) |

B) Subtract
op-code type = 001

>>Subtract Immediate (subi)
++subi r1, #43 // r1 ← r1 - 43 sub immediate

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 001 | dst(operand 1) | 00 | src1(operand 2) | src2(operand 2) |
| IMMEDIATE |
>>Subtract Register (subr)
++subr r1, r2 // r1 ← r1 - r2 sub register

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 001 | dst(operand 1) | 01 | src1(operand 2) | X |
>>Subtract with Base Indexed Addressing (subx)
++subx r2,10(r1, r7) // r2 ← r2 - M[r1+r7+10]
//r2 ← r2 + M[r1 + r7 +M[PC]]; PC ← PC - 2 sub indexed

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 001 | dst(operand 1) | 10 | src1(operand 2) | src2(operand 2) |
| D2 (next line for operand 2) |

>>Subtract with Memory Indirect Addressing (subn)
++subn @-120(r2, r6) // r4 ← r4 - M(M(r2 +
r6 -120)) sub indirect
BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 001 | dst(operand 1) | 11 | src1(operand 2) | src2(operand 2) |
| D2 (next line for operand 2) |

C) Comparison
op-code type = 010
++mnsi r1, #43 // compare r1 - 43 compare
immediate

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 001 | dst(operand 1) | 00 | src1(operand 2) | src2(operand 2) |
| IMMEDIATE |

++mnsr r1, r2 // compare r1 - r2 compare
register

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 001 | dst(operand 1) | 01 | src1(operand 2) | X |

++mnsx r2,10(r1, r7) // compare r2 -
M[r1+r7+10]
//compare r2 + M[r1 + r7 +M[PC]]; PC -> PC - 2 compare indexed

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 001 | dst(operand 1) | 10 | src1(operand 2) | src2(operand 2) |
| D2 (next line for operand 2) |

++mnsn @-120(r2, r6) // compare r4 -
M(M(r2 + r6 -120)) compare indirect
BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |01 | 001 | dst(operand 1) | 11 | src1(operand 2) | src2(operand 2) |

| D2 (next line for operand 2) |

D) Complement
op-code type = 011

++cmp r3 // only one register operand instruction
// src2 is don't care
// r3 ← 2's complement of r3
//Comparison is accomplished by subtraction. All these instructions set the status flags.

BITS (inclusive):	15-14	13-11	10-8	7-6	5-3	2-0
FORMAT:	01	011	dst(operand 1)	X	X	X

==========

III) Jump instructions:
op-code = 10

General assembly instruction format: op-code dst
It's a two word instruction where the second word contains a signed displacement relative to the PC. (So only one addressing mode allowed.) Interpretation:
j addr // if (true) PC ← PC + M[PC]
jnm addr // if (not S) PC ← PC + M[PC], where "not S"
// indicates that the result of last ALU
// operation was not negative.
//The rest are similar.

/* A) j (jump unconditionally)
op-code type = 000
Can be inserted in subroutine op-code....or jump zero with mnsr r1-r1 can be used
*/

B) jz (jump on zero (flag set)
op-code type = 001

BITS (inclusive):	15-14	13-11	10-8	7-6	5-3	2-0
FORMAT:	10	001	X	X	X	X
	PC relative memory address					

C) jnz (jump on not zero)
op-code type = 010

BITS (inclusive):	15-14	13-11	10-8	7-6	5-3	2-0
FORMAT:	10	010	X	X	X	X
	PC relative memory address					

D) jc (jump on carry)

op-code type = 011

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |10 | 011 | X | X | X | X |
| PC relative memory address |

E) jnc (jump on not carry)
op-code type = 100

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |10 | 100 | X | X | X | X |
| PC relative memory address |

F) jv (jump on overflow)
op-code type = 101

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |10 | 101 | X | X | X | X |
| PC relative memory address |

G) jnv (jump on not overflow)
op-code type = 110

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |10 | 110 | X | X | X | X |
| PC relative memory address |

H) jm (jump on minus (sign flag set))
op-code type = 111

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |10 | 111 | X | X | X | X |
| PC relative memory address |

I) jnm (jump on not minus)
op-code type = 000

BITS (inclusive): | 15-14 | 13-11 | 10-8 | 7-6 | 5-3 | 2-0 |
FORMAT: |10 | 000 | X | X | X | X |
| PC relative memory address |

==========
==========
==========
==========

IV) Subroutine call and return: jal (jump and link), jr (jump to return)
Assembly instruction format: jal link register subroutine address
op-code = 11

Interpretation:
A) jal r5, sub // r5 ← PC+2; PC ← PC + M[PC], where M[PC] (the
// second word) contains the address of the first
// instruction of the subroutine "sub" relative
// to the PC -- better is to permit it to be base
// register relative providing for longer jumps --
// but for simplicity let's have PC relative.
op-code type = 001

BITS (inclusive):	15-14	13-11	10-8	7-6	5-3	2-0
FORMAT:	11	001	dst(link)	X	X	X
	PC relative memory address					

B) Return instruction: Single operand
Assembly instruction format: jr link register
Interpretation:
jr r5 // PC ← r5
op-code type = 010

BITS (inclusive):	15-14	13-11	10-8	7-6	5-3	2-0
FORMAT:	11	010	dst(link)	X	X	X

==