Keywords in Assembly Language

Subject: Computer Architecture Lab

1. **ADD** (Addition): The ADD instruction adds together its two operands, storing the result in its first operand.
2. **ADC** (Addition with Carry): The ADC instruction adds two operands along with the carry instruction. This is useful for numbers larger than a single 32 bit word.
3. **AND** (Bitwise AND): Performs a logical AND of each bit in the values specified by the two operands and stores the result in the second operand.
4. **B** (Branch): The B instruction branches to an instruction specified by the branch target address.
5. **BOUND** (Check array index against bounds): Ensures that a signed array index value falls within the upper and lower bounds of a block of memory.
6. **CBTW** (Convert Byte to Word): This instruction converts the signed byte in to a signed word.
7. **CMP** (Compare): The CMP instruction subtracts two operands but does not store the result. It is used for comparison between two operands.
8. **CWTD** (Convert Signed Word to Signed Double Word): The CWTD converts the signed word in to a signed double word.
9. **CWTL** (Convert Word to Long): This instruction converts the signed word in to a signed long.
10. **DEC** (Decrement): The DEC instruction subtracts 1 from the operand. Does not change the carry flag.
11. **DIV** (Division): The DIV instruction divides a register value by another register value or memory byte, word, or long.
12. **EOR** (Bitwise XOR): The EOR instruction executes a bitwise exclusive OR operation between two registers and stores the result in a destination register.
13. **IN / INS** (Input from Port): This instruction transfers a byte, word, or long from the immediate port into the byte, word, or long memory address pointed by the register.
14. **INC** (Increment): The INC operator adds 1 to the operand and does not change the carry flag.
15. **IDIV** (Signed Division): The IDIV instruction executes signed division.

16. **IMUL** (Signed Multiplication): The IMUL instruction executes signed multiplication.
17. **OUT / OUTS** (Output from Port): This instruction transfers a byte, word, or long from the memory address pointed to by the content of the register to the immediate port address.
18. **LDR** (Load Register): The LDR instruction is used to load something from memory into a register.
19. **LSL** (Logical Shift Left): The LSL instruction shifts the bits in given register to the left by a shift value.
20. **LSR** (Logical Shift Right): The LSR instruction shifts the bits in given register to the right by a shift value.
21. **MOV** (Move): The MOV instruction copies data from a source to a destination.
22. **MUL** (Multiplication): The MUL instruction executes a unsigned multiply of a byte, word, or long operand.
23. **NEG** (Two’s Complement Negation): The NEG instruction replace the value of byte, word or long with it’s two’s complement.
24. **NOT** (One’s Complement Negation): Inverts each bit value of the byte, word, or long; that is, every 1 becomes a 0 and every 0 becomes a 1.
25. **ORR** (Bitwise OR): The ORR instruction executes a bitwise OR operation between two registers and stores the result in a destination register.
26. **SBB** (Subtraction with Borrow): Subtracts (operand1 and the carry flag) from operand2 and stores the result in operand2.
27. **SBC** (Subtraction with Carry): The SBC instruction subtracts two operands along with the carry instruction.
28. **STR** (Store): The STR instruction stores a register value into memory.
29. **SUB** (Subtraction): The SUB instruction subtracts its two operands, storing the result in its first operand.
30. **TEST** (Logical Comparison): Performs a bit-wise logical AND of the two operands. The result of a bit-wise logical AND is 1 if the value of that bit in both operands is 1; otherwise, the result is 0.