RISC-V EMULATOR:-

It is a command line based emulator. It can be used to run RISC-V program.

This application supports all the RV32I and RV64I instructions. It also supports “M” standard extension for integer multiplication and division.

This interpreter supports the following instructions:

Computational Instruction:-

Arithmetic: add, addw, addi, addiw, sub, subw, mul, mulw, mulh mulhu, mulhsu , lui

Logical: or, ori, and, andi, xor, xori

Compare: slt, slti, sltu, sltui

Shift: sll, sllw, slli, slliw, srl, srlw srli, srliw sra, sraw, srai sraiw

DataTransfer Instruction:- sb, sh, sw, sd, lb, lh, lw, ld, lbu, lbh, lbw

Branch Instructions:- beq, bne, bge, blt, bgeu, bltu, jal, jalr

RISC-V Architecture:-

General-purpose registers x1 through x31 are available for use without any restrictions or special functions assigned by the processor hardware. Register x0 is hardwired to return zero when it is read, and will discard any value written to it. There is no processor flags register in the RISC-V ISA. Some operations that modify flags in other processor architectures instead store their results in a RISC-V register. For example, the signed (slt) and unsigned (sltu) RISC-V comparison instructions subtract two operands and set a destination register to 0 or 1 depending on the sign of the result. A subsequent conditional branch instruction uses the value in that register to determine which code path to take.

Most of the base ISA computational instructions use a three-operand format, in which the first operand is the destination register, the second operand is a source register, and the third operand is either a second source register or an immediate value. This is an example three-operand instruction:

add x1, x2, x3

To avoid introducing instructions that are not strictly necessary, many instructions take on extra duties that are performed by dedicated instructions in other processor architectures. For example, RISC-V has no instruction that simply moves one register to another. Instead, a RISC-V addition instruction adds a source register and an immediate value of zero and stores the result in a destination register, producing the same result. The instruction to transfer register x2 to x1 is therefore add x1, x2, x0 assigning the value (x2 + 0) to x1.

RISC-V Base Instruction Set:-

RISC-V base instructions fall into the categories of computational instructions, control flow instructions, and memory access instructions.

Computational Instructions:-

- Arithmetic Instruction (add, addi, sub):-Perform addition and subtraction. The immediate value in the addi instruction is a 12-bit signed value. The sub instruction subtracts the second source operand from the first. There is no subi instruction because addi can add a negative immediate value.

- Shift Instructions(sll, slli, srl, srli, sra, srai):- Perform logical left and right shifts (sll and srl), and arithmetic right shifts (sra). Logical shifts insert zero bits into vacated locations. Arithmetic right shifts replicate the sign bit into vacated locations. The number of bit positions to shift is taken from the lowest 5 bits of the second source register or from the 5-bit immediate value.

- Logical Instruction(and, andi, or, ori, xor, xori):- Perform the indicated bitwise operation on the two source operands. Immediate operands are 12 bits.

- Compare Instructions(slt, slti, sltu, sltui):- The set if less than instructions set the destination register to 1 if the first source operand is less than the second source operand: This comparison is in terms of two's complement (slt) or unsigned (sltu) operands. Immediate operand values are 12 bits.
- lui:- Load upper immediate. This instruction loads bits 12-31 of the destination register with a 20-bit immediate value. Setting a register to an arbitrary 32-bit immediate value requires two instructions: First, lui sets bits 12-31 to the upper 20 bits of the value. Then addi adds in the lower 12 bits to form the complete 32-bit result. lui has two operands: the destination register and the immediate value.

Control Flow Instructions:-

The conditional branching instructions perform comparisons between two registers and, based on the result, may transfer control within the range of a signed 12-bit address offset from the current PC. Two unconditional jump instructions are available, one of which (jalr) provides access to the entire 32-bit address range.

- beq, bne, blt, bltu, bge, bgeu:- Branch if equal (beq), not equal (bne), less than (blt), less than unsigned (bltu), greater or equal (bge), or greater or equal, unsigned (bgeu). These instructions perform the designated comparison between two registers and, if the condition is satisfied, transfer control to the address offset provided in the 12-bit signed immediate value.
- jal:- Jump and link. Transfer control to the PC-relative address provided in the 20-bit signed immediate value and store the address of the next instruction (the return address) in the destination register.
- jalr:- Jump and link, register. Compute the target address as the sum of the source register and a signed 12- bit immediate value, then jump to that address and store the address of the next instruction in the destination register. When preceded by the auipc instruction, the jalr instruction can perform a PC-relative jump anywhere in the 32-bit address space.

Memory Access Instructions:-

The memory access instructions transfer data between a register and a memory location. The first operand is the register to be loaded or stored. The second is a register containing a memory address. A signed 12-bit immediate value is added to the address in the register to produce the final address for the load or store.

The load instructions perform sign extension for signed values or zero extension for unsigned values. The sign or zero extension operation fills in all 32 bits in the destination register when a smaller data value (a byte or halfword) is loaded. Unsigned loads are specified by a trailing *u* in the mnemonic.

- lb, lbu, lh, lhu, lw:- Load an 8-bit (byte) (lb), a 16-bit (halfword) (lh) or 32-bit (word) (lw) into the destination register. For byte and halfword loads, the instruction will either sign-extend (lb and lh) or zero-extend (lbu and lhu) to fill the 32-bit destination register. For example, the instruction lw x1, 16(x2) loads the word at the address (x2 + 16) into register x1.
- sb, sh, sw:- Store a byte (sb), halfword (sh) or word (sw) to a memory location matching the size of the data value.

Pseudo Instructions:-

The RISC-V architecture has a truly reduced instruction set, lacking several types of instructions present in the instruction sets of other processor architectures. The functions of many of those more familiar instructions can be performed with RISC-V instructions, though perhaps not in an immediately intuitive manner.

The RISC-V assembler supports a number of pseudo-instructions, each of which translates to one or more RISC-V instructions providing a type of functionality one might expect in a general-purpose processor instruction set. The following table presents a few of the most useful RISC-V pseudo-instructions:

PSEUDO-INSTRUCTION	RISC-V INSTRUCTION	FUNCTION
Nop	addi x0, x0, 0	No operation
mv rd,rs	addi rd, rs, 0	Copy rs to rd
not rd, rs	ori rd, rs, -1	rd = NOT rs
neg rd, rs	sub rd, x0, rs	rd = -rs
j offset	jal x0, offset	Unconditional jump
jal offset	jal x1, offset	Near function call (20-bit offset)
call offset	auipc x1, offset[31:12] + offset[11] jalr x1, offset[11:0](x1)	Far function call (32-bit offset)
ret	jalr x0, 0(x1)	Return from function
beqz rs, offset	beq rs, x0, offset	Branch if equal to zero
bgez rs, offset	bge rx, x0, offset	Branch if greater than or equal to zero
bltz rs, offset	blt rs, x0, offset	Branch if less than zero
bgt rs, rt, offset	blt rt, rs, offset	Branch if greater than
ble rs, rt, offset	bge rt, rs, offset	Branch if less than or equal
li rd, immed	addi rd, x0, immed	Load 12-bit immediate

RISC-V Extension:-

The instruction set described in this section is named RV32I, which stands for the RISC-V 32-bit integer instruction set. Although the RV32I ISA provides a complete and useful instruction set for many purposes, it lacks several functions and features available in other processors such as x86 and ARM.

The RISC-V extensions provide a mechanism for adding capabilities to the base instruction set in an incremental and compatible manner. Implementors of RISC-V processors can selectively include extensions in a processor design to optimize trade-offs between chip size, system capability, and performance.

M Extension:-

The RISC-V M extension adds integer multiplication and division functionality to the base RV32I instruction set. The following instructions are available in this extension:

- mul:- Multiply two 32-bit registers and store the lower 32 bits of the result in the destination register.
- mulh, mulhu, mulhsu:- Multiply two 32-bit registers and store the upper 32 bits of the result in the destination register. Treat the multiplicands as both signed (mulh), both unsigned (mulhu), or signed rs1 times unsigned rs2 (mulhsu). rs1 is the first source register in the instruction and rs2 is the second.
- div, divu :- Perform division of two 32-bit registers, rounding the result toward zero, on signed (div) or unsigned (divu) operands.
- rem, remu:- Return the remainder corresponding to the result of a div or divu instruction on the operands.

Division by zero does not raise an exception. To detect division by zero, code should test the divisor and branch to an appropriate handler if it is zero.

64-bit RISC-V

The RISC-V introduction to this point has discussed the 32-bit RV32I architecture and instruction set, with extensions. The RV64I instruction set extends RV32I to a 64-bit architecture. As in RV32I, instructions are 32-bits wide. In fact, the RV64I instruction set is almost entirely the same as RV32I except for a few significant differences:

- All of the integer registers are widened to 64 bits.
- Addresses are widened to 64 bits.
- Bit shift counts in instruction opcodes increase in size from 5 to 6 bits.
- Several new instructions are defined to operate on 32-bit values in a manner equivalent to RV32I. These instructions are necessary

because most instructions in RV64I operate on 64-bit values and there are many situations in which it is necessary to operate efficiently on 32-bit values. These word-oriented instructions have an opcode mnemonic suffix of w. The w-suffix instructions produce signed 32-bit results. These 32-bit values are sign-extended (even if they are unsigned values) to fill the 64-bit destination register. In other words, bit 31 of each result is copied into bits 32-63.

The following new instructions are defined in RV64I:

- addw, addiw, subw, sllw, slliw, srlw, srliw, sraw, sraiw:- These instructions perform equivalently to the RV32I instruction with the same mnemonic, minus the `w` suffix. They work with 32-bit operands and produce 32-bit results. The result is sign-extended to 64 bits.
- ld, sd:- Load and store a 64-bit doubleword. These are the 64-bit versions of the lw and sw instructions in the RV32I instruction set.

The remaining RV32I instructions perform the same functions in RV64I, except addresses and registers are 64 bits in length. The same opcodes, both in assembly source code and binary machine code, are used in both instruction sets.

Design of Emulator:-

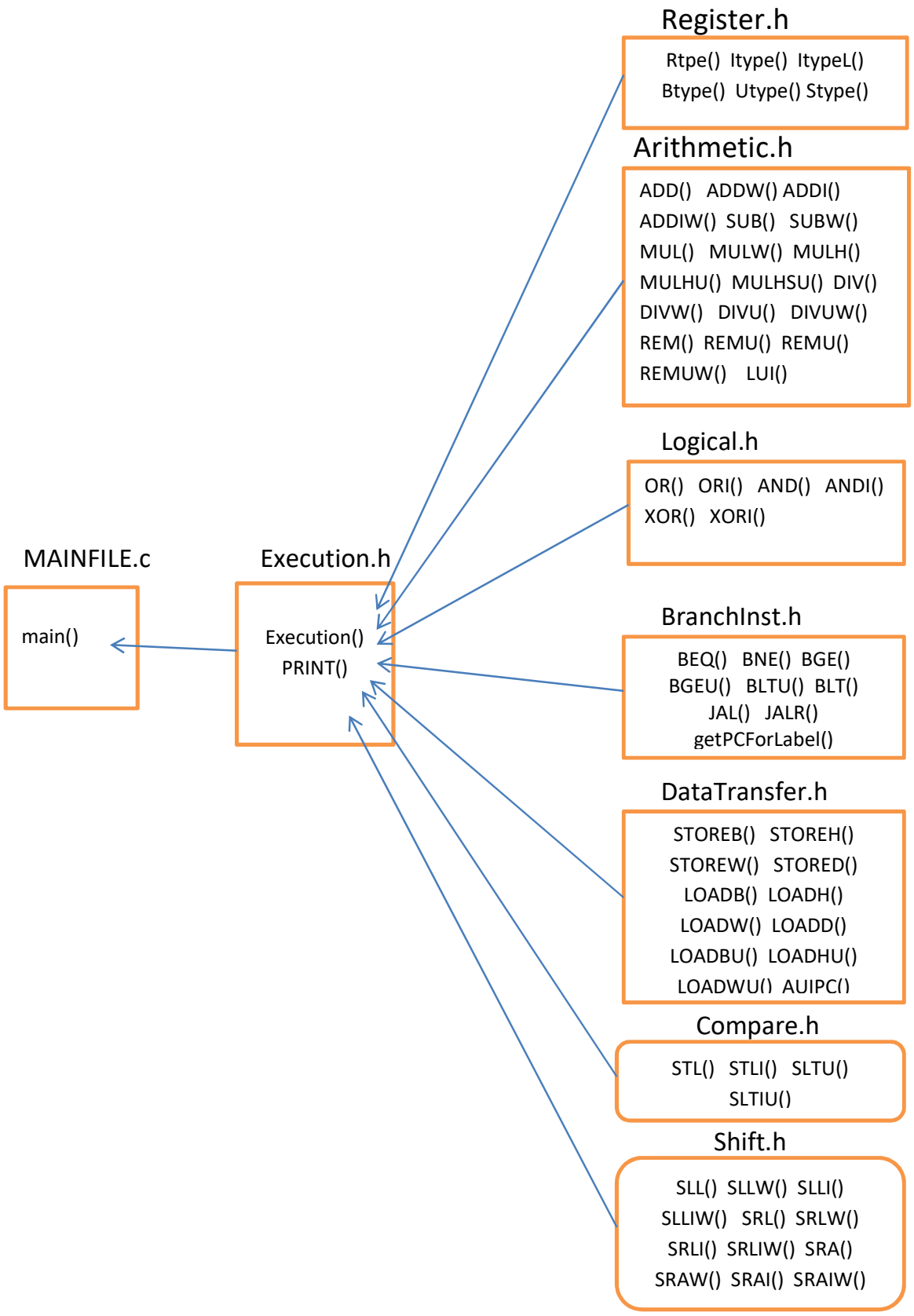
This emulator is designed in multiple file format. Here we have single .c file that contains main() function and eight header files that contains definition of all useful functions.

.c File:- MAINFILE.c

.h File:- Execution.h, Register.h, Arithmetic.h, Logical.h, BranchInst.h, DataTransfer.h, Shift.h and Compare.h

MAINFILE.c includes Execution.h header file and Execution.h includes rest all header files(Register.h, Arithmetic.h, Logical.h, BranchInst.h, DataTranfer.h, Shift.h, Compare.h).

Structure of this Emulator:-



MAINFILE.c:-

This file contains a main() function that reads your complete file that is given to its argument and stores its all instructions and labels in array of struct data type. It keeps track on which instruction is executing.

It includes Execution.h file and for every new instruction it transfer control to Execution.h header file.

Execution.h:-

It includes Register.h, Arithmetic.h, Logical.h, Shift.h, DataTransfer.h, Compare.h, BranchInst.h header files. This file stores definition of executeInstruction() and print() function.

executeInstruction():-This file stores current instruction in an string and identifies which instruction is going to be executed. If it is unable to recognize which instruction it is it prints line number where it failed and terminate the program.

If it recognizes instruction it transfer control to the specific function that is present in specific header file.

print():-This function is called when print label is encountered. It prints values stored in specific registers.

Register.h:-

This file stores definition of Rtype(), Itype(), ItypeL(), Stype(), Btype(), Utype(), invalidInstruction().

Rtype(): This function is called when instruction format is op rd, rs1, rs2. This function identifies which register is used as destination register and which are as source registers. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

Itype():-This function is called when instruction format is op rd, rs1, imm. This function recognizes which register is used as destination register and which as source register. It also finds decimal value of immediate. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

ItypeL():-This function is called when instruction format is op rd, offset(rs1). This function identifies which register is used as destination register and which as source register. It finds decimal value of offset. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

Stype():-This function is called when instruction format is op rs1, offset(rs2). This function recognizes which registers are used as source registers. It finds decimal value of offset. It takes a string in which instruction is stored and index which points first source register as arguments. Its return type is void.

Btype():-This function is called when instruction format is op rs1, rs2, imm/label. This function recognizes which registers are used as source registers. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is int. It returns index where it is currently pointing in string.

Utype():-This function is called when instruction format is op rd imm/label. This function recognizes which register is used as destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is int. It returns index where it is currently pointing in string.

invalidInstruction():- This function is called when any undefined behavior is encountered.

Arithmetic.h:-

This file stores definition of ADD(), ADDI(), ADDW() ADDIW(), SUB(), SUBW(), MUL(), MULW(), MULH(), MULHU(), MULHSU(), DIV(), DIVW(), DIVW(), DIVUW(), REM(), REMU(), REMW(), REMUW().

ADD():-This function is called when instruction is add rd, rs1, rs2. It performs addition on values present in source registers and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

ADDI():-This function is called when instruction is addi rd, rs1, imm. It performs addition on values present in source register and immediate and stores in

destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

ADDW():-This function is called when instruction is addw rd, rs1, rs2. It performs addition on lower 32-bit values present in source registers and stores its sign-extended reprensentation in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

ADDIW():-This function is called when instruction is addiw rd, rs1, imm. It performs addition on lower 32-bit values present in source register and immediate stores its sign-extended representation in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SUB():-This function is called when instruction is sub rd, rs1, rs2. It subtracts values present in rs2 from values present in rs1 and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SUBW():-This function is called when instruction is sub rd, rs1, rs2. It subtracts lower 32-bit values present in rs2 from lower 32-bit values present in rs1 and stores in its sign-extended representation in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

MUL():-This function is called when instruction is mul rd, rs1, rs2. It performs multiplication on values present in source registers and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

MULW():-This function is called when instruction is mulw rd, rs1, rs2. It performs multiplication on lower 32-bit values present in source registers and stores its sign-extended representation in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

MULH():-This function is called when instruction is add mulh, rs1, rs2. It performs multiplication on values present in source registers and stores its

upper 64-bit in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

MULHU():-This function is called when instruction is add mulhu, rs1, rs2. It performs multiplication on unsigned values present in source registers and stores its upper 64-bit in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

MULHSU():-This function is called when instruction is add mulhsu, rs1, rs2. It performs multiplication on signed values present in first source register and unsigned value present in second source register and stores its upper 64-bit in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

DIV():-This function is called when instruction is div rd, rs1, rs2. It divides value present in rs1 by value present in rs2 and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

DIVW():-This function is called when instruction is divw rd, rs1, rs2. It divides lower 32-bit value present in rs1 by lower 32-bit value present in rs2 and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

DIVU():-This function is called when instruction is divu rd, rs1, rs2. It divides unsigned value present in rs1 by unsigned value present in rs2 and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

DIVUW():-This function is called when instruction is divuw rd, rs1, rs2. It divides lower unsigned 32-bit value present in rs1 by lower unsigned 32-bit value present in rs2 and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

REM():-This function is called when instruction is rem rd, rs1, rs2. It divides value present in rs1 by value present in rs2 and stores remainder in destination

register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

REMW():-This function is called when instruction is remw rd, rs1, rs2. It divides lower 32-bit value present in rs1 by lower 32-bit value present in rs2 and stores remainder in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

REMU():-This function is called when instruction is remu rd, rs1, rs2. It divides unsigned value present in rs1 by unsigned value present in rs2 and stores remainder in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

REMUW():-This function is called when instruction is remuw rd, rs1, rs2. It divides lower unsigned 32-bit value present in rs1 by lower unsigned 32-bit value present in rs2 and stores remainder in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

LUI():-This function is called when instruction is lui rd, imm. It stores 12-31 bit of destination register with 20-bit immediate. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

Logical.h:-

This header file stores definition of OR(), ORI(), AND(), ANDI(), XOR(), XORI().

OR():-This function is called when instruction is or rd, rs1, rs2. It performs or operation on values present in source registers and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

ORI():-This function is called when instruction is ori rd, rs1, imm. It performs or operation on values present in source register and immediate and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

AND():-This function is called when instruction is and rd, rs1, rs2. It performs and operation on values present in source registers and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

ANDI():-This function is called when instruction is andi rd, rs1, imm. It performs and operation on values present in source register and immediate and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

XOR():-This function is called when instruction is xor rd, rs1, rs2. It performs xor operation on values present in source registers and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

XORI():-This function is called when instruction is xori rd, rs1, imm. It performs xor operation on values present in source register and immediate and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

Shift.h:-

This header file stores definition of SLL(), SLLW(), SLLI(), SLLIW(), SRA(), SRAW(), SRAI(), SRAIW(), SRL(), SRLW(), SRLI(), SRLIW().

SLL():-This function is called when instruction is sll rd, rs1, rs2. It shifts left the value present in rs1 by value present in rs2 and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SLLW():-This function is called when instruction is sllw rd, rs1, rs2. It shifts left lower 32-bit value present in rs1 by lower 32-bit value present in rs2 and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SLLI():-This function is called when instruction is slli rd, rs1, imm. It shifts left the value present in rs1 by value of immediate and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SLLIW():-This function is called when instruction is slliw rd, rs1, rs2. It shifts left lower 32-bit value present in rs1 by value ifimmediate and stores in destination register. It takes a string in which instruction is stored and ind index which points destination register as arguments. Its return type is void.

SRA():-This function is called when instruction is sra rd, rs1, rs2. It shifts right the value present in rs1 by value present in rs2 arithmetically and stores result in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SRAW():-This function is called when instruction is sraw rd, rs1, rs2. It shifts right lower 32-bit value present in rs1 by lower 32-bit value present in rs2 arithmetically and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SRAI():-This function is called when instruction is srai rd, rs1, imm. It shifts right value present in rs1 by value of immediate arithmetically and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SRAIW():-This function is called when instruction is sraiw rd, rs1, rs2. It shifts right arithmetically lower 32-bit value present in rs1 by value of immediate and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SRL():-This function is called when instruction is srl rd, rs1, rs2. It shifts right value present in rs1 by value present in rs2 logically and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SRLW():-This function is called when instruction is srlw rd, rs1, rs2. It shifts right lower 32-bit value present in rs1 by lower 32-bit value present in rs2 logically and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SRLI():-This function is called when instruction is srli rd, rs1, imm. It shifts right value present in rs1 by value of immediate logically and stores in destination

register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SRLIW():-This function is called when instruction is srliw rd, rs1, rs2. It shifts right lower 32-bit value present in rs1 by value of immediate logically and stores in destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

Compare.h:-

This header file stores definition of SLT(), SLTI(), SLTU(), SLTUI().

SLT():-This function is called when instruction is slt rd, rs1, rs2. It compares content of both source registers. If content of rs1 is less than content of rs2 it sets the destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SLTI():-This function is called when instruction is slti rd, rs1, imm. It compares content of rs1 and immediate. If content of rs1 is less than immediate it sets the destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SLTU():-This function is called when instruction is sltu rd, rs1, rs2. It compares unsigned values stored in both source registers. If unsigned value of rs1 is less than unsigned value of rs2 it sets the destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

SLTUI():-This function is called when instruction is sltui rd, rs1, rs2. It compares unsigned value of rs1 and immediate. If unsigned content of rs1 is less than immediate it sets the destination register. It takes a string in which instruction is stored and index which points destination register as arguments. Its return type is void.

DataTransfer.h:-

This header file stores definition of STOREB(), STOREH(), STOREW(), STORED(), LOADB(), LOADH(), LOADW(), LOADD(), LOADBU(), LOADHU(), LOADWU().

STOREB():-This function is called when instruction is sb rs1, offset(rs2). It stores lower 8-bit of rs1 to the memory location [rs2]+offset. It takes a string in which instruction is stored and index which points to first source register as arguments. Its return type is void.

STOREH():-This function is called when instruction is sh rs1, offset(rs2). It stores lower 16-bit of rs1 to the memory location [rs2]+offset. It takes a string in which instruction is stored and index which points to first source register as arguments. Its return type is void.

STOREW():-This function is called when instruction is sw rs1, offset(rs2). It stores lower 32-bit of rs1 to the memory location [rs2]+offset. It takes a string in which instruction is stored and index which points to first source register as arguments. Its return type is void.

STORED():-This function is called when instruction is sd rs1, offset(rs2). It stores content of rs1 to the memory location [rs2]+offset. It takes a string in which instruction is stored and index which points to first source register as arguments. Its return type is void.

LOADB():-This function is called when instruction is lb rd, offset(rs1). It load a byte from memory location [rs1]+offset to register rd. It takes a string in which instruction is stored and index which points to destination register as arguments. Its return type is void.

LOADH():-This function is called when instruction is lh rd, offset(rs1). It load a halfword from memory location [rs1]+offset to register rd. It takes a string in which instruction is stored and index which points to destination register as arguments. Its return type is void.

LOADW():-This function is called when instruction is lw rd, offset(rs1). It load a word from memory location [rs1]+offset to register rd. It takes a string in which instruction is stored and index which points to destination register as arguments. Its return type is void.

LOADD():-This function is called when instruction is lb rd, offset(rs1). It load a doubleword from memory location [rs1]+offset to register rd. It takes a string in which instruction is stored and index which points to destination register as arguments. Its return type is void.

LOADBU():-This function is called when instruction is lbu rd, offset(rs1). It load unsigned value of byte from memory location [rs1]+offset to register rd. It takes a string in which instruction is stored and index which points to destination register as arguments. Its return type is void.

LOADHU():-This function is called when instruction is lhu rd, offset(rs1). It load unsigned value of halfword from memory location [rs1]+offset to register rd. It takes a string in which instruction is stored and index which points to destination register as arguments. Its return type is void.

LOADWU():-This function is called when instruction is lwu rd, offset(rs1). It load unsigned value of word from memory location [rs1]+offset to register rd. It takes a string in which instruction is stored and index which points to destination register as arguments. Its return type is void.

BranchInst.h:-

This header file stores definition of BEQ(), BNE(), BGE(), BGEU(), BLT(), BLTU(), JAL(), JALR().

BEQ():-This function is called when instruction is beq rs1, rs2, imm/label. It compares content of rs1 and rs2 if both are equal it sets pc to respective label or pc+immediate. It takes a string in which instruction is stored and index which points to first source register as arguments. Its return type is void.

BNE():-This function is called when instruction is bne rs1, rs2, imm/label. It compares content of rs1 and rs2 if both are not equal it sets pc to respective label or pc+immediate. It takes a string in which instruction is stored and index which points to first source register as arguments. Its return type is void.

BGE():-This function is called when instruction is bge rs1, rs2, imm/label. It compares content of rs1 and rs2 if content of rs1 is greater than or equal to content of rs1 it sets pc to respective label or pc+immediate. It takes a string in which instruction is stored and index which points to first source register as arguments. Its return type is void.

BGEU():-This function is called when instruction is bgeu rs1, rs2, imm/label. It compares content of rs1 and rs2 if unsigned value of rs1 greater than or equal to unsigned value of rs2 it sets pc to respective label or pc+immediate. It takes

a string in which instruction is stored and index which points to first source register as arguments. Its return type is void.

BLT():-This function is called when instruction is blt rs1, rs2, imm/label. It compares content of rs1 and rs2 if content of rs1 is less content of rs2 it sets pc to respective label or pc+immediate. It takes a string in which instruction is stored and index which points to first source register as arguments. Its return type is void.

JAL():-This function is called when instruction is jal rd, imm/label. It stores location of next instruction in rd sets pc to respective label or pc+immediate. It takes a string in which instruction is stored and index which points to destination register as arguments. Its return type is void.

JALR():-This function is called when instruction is jalr rd, offset(rs1). It stores location of next instruction in rd sets pc to ([rs1]+offset)/4. It takes a string in which instruction is stored and index which points to destination register as arguments. Its return type is void.

Execution of Arithmetic Instruction:-

Different formats of Arithmetic instruction

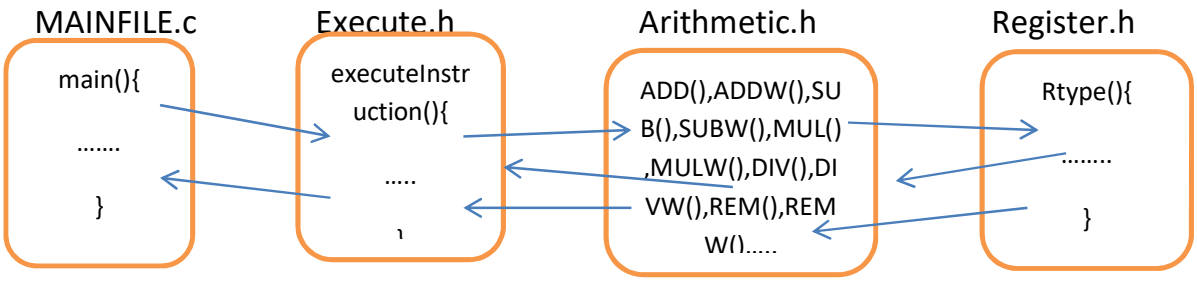
op rd, rs1, rs2
op rd, rs1, imm
op rd, imm

When instruction is in format op rd, rs1, rs2.

add rd, rs1, rs2
addw rd, rs1, rs2
sub rd, rs1, rs2
subw rd, rs1, rs2

mul rd, rs1, rs2
mulw rd, rs1, rs2
div rd, rs1, rs2
divu rd, rs1, rs2
divw rd, rs1, rs2
divuw rd, rs1, rs2
rem rd, rs1, rs2
remw rd, rs1, rs2
remu rd, rs1, rs2
remuw rd, rs1, rs2

In process of execution of program when pc points one of these instruction control is transferred from main() function(resides in MAINFILE.c) to executeInstruction()function(resides in Execution.h header file). Where it identifies which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in Arithmetic.h header file where control is transferred to Rtype() function(resides in Register.h header file). Rtype() function identifies which register is used as destination register and which are source registers and after that it returns control to Arithmetic.h. Here, appropriate operation is performed on content of source register and result is stored in destination register. Now, control is returned to executeInstruction() function and from here control is returned to main() function where pc is increased to point next instruction

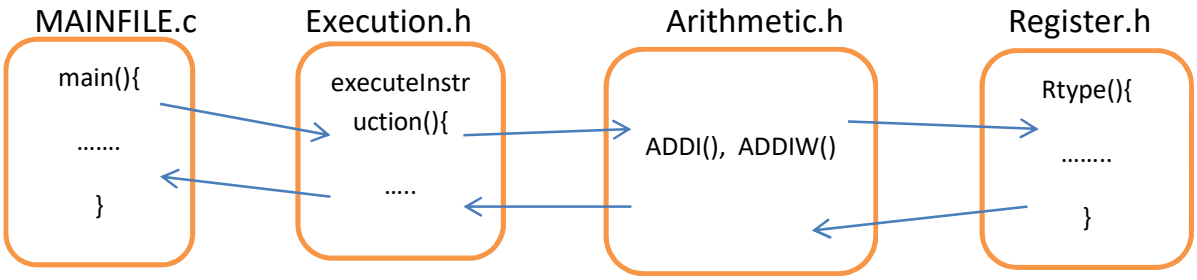


Control Flow of Execution of Arithmetic Instruction(op rd, rs1, rs2)

When instruction is in format op rd, rs1, imm.

addi rd, rs1, imm
addiw rd, rs1, imm

In process of execution of program when pc points one of these instruction control is transferred from main() function(resides in MAINFILE.c) to executeInstruction()function(resides in Execution.h header file). Where it identifies which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in Arithmetic.h header file where control is transferred to Itype() function(resides in Register.h header file). Itype() function recognizes which register is used as destination register and which is as source register. It finds value of immediate and also ensures that immediate is 12-bit value and after that it returns control to Arithmetic.h. Here, appropriate operation is performed on content of source register and immediate and result is stored in destination register. Now, control is returned to executeInstruction() function and from here control is returned to main() function where pc is increased to point next instruction.



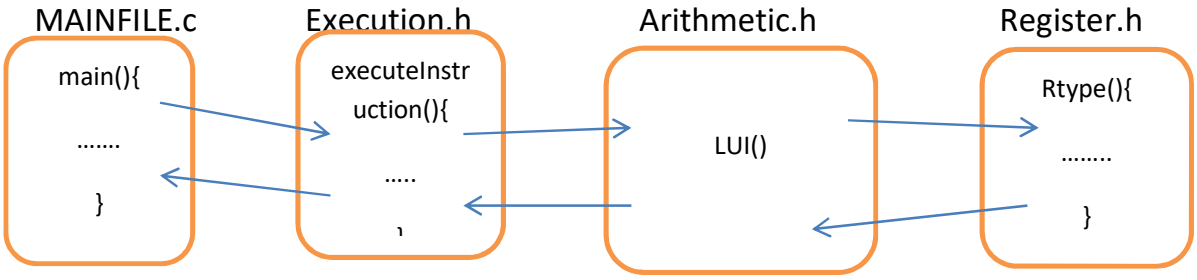
Control Flow of Execution of Arithmetic Instruction(op rd, rs1, imm)

When instruction is in format op rd, imm.

lui rd, imm

In process of execution of program when pc points this instruction control is transferred from main() function(resides in MAINFILE.c) to executeInstruction()function(resides in Execution.h header file). Where it recognizes which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in Arithmetic.h header file where control is transferred to Utype() function(resides in Register.h header file). Utype() function identifies which register is used as destination register and finds value of immediate and also ensures that immediate is 20-bit and after that it returns control to

Arithmetic.h. Here, appropriate operation is performed. Now, control is returned to executeInstruction() function and from here control is returned to main() function where pc is increased to point next instruction.



Control Flow of Execution of Arithmetic Instruction(op rd, imm)

Execution of Logical Instruction:-

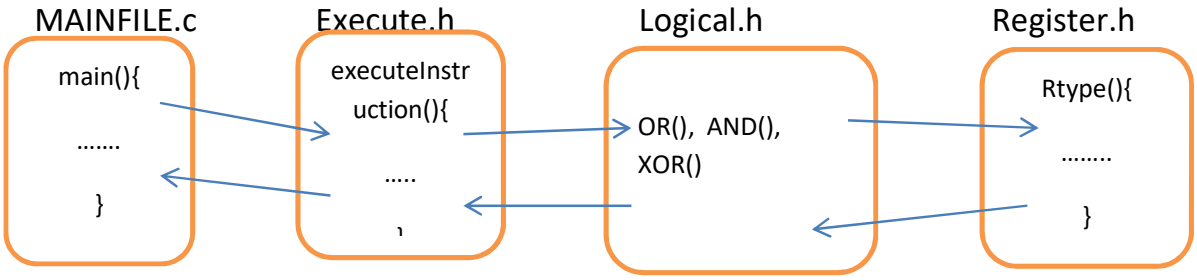
Different formats of Logical instruction

op rd, rs1, rs2
op rd, rs1, imm

When instruction is in format op rd, rs1, rs2.

or rd, rs1, rs2
and rd, rs1, rs2
xor rd, rs1, rs2

In process of execution of program when pc points one of these instruction control is transferred from main() function(resides in MAINFILE.c) to executeInstruction()function(resides in Execution.h header file). Where it identifies which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in Logical.h header file where control is transferred to Rtype() function(resides in Register.h header file). Rtype() function recognizes which register is used as destination register and which are source registers and after that it returns control to Arithmetic.h. Here, appropriate operation is performed on content of source register and result is stored in destination register. Now, control is returned to executeInstruction() function and from here control is returned to main() function where pc is increased to point next instruction.

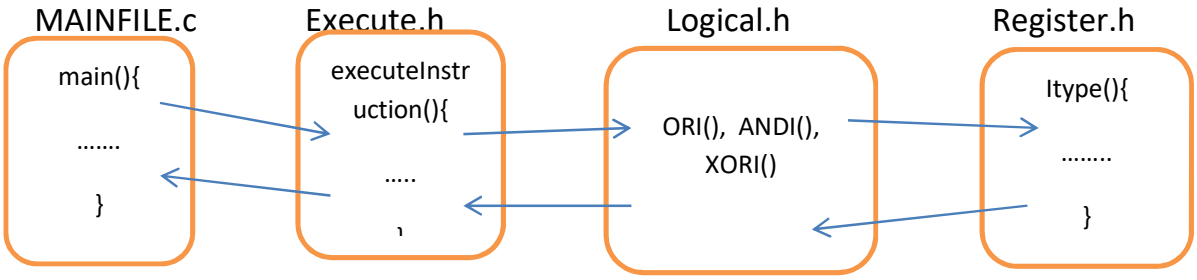


Control Flow of Execution of Logical Instruction(op rd, rs1, rs2)

When instruction is in format op rd, rs1, imm.

or rd, rs1, imm
andi rd, rs1, imm
xori rd, rs1, imm

In process of execution of program when pc points one of these instruction control is transferred from main() function(resides in MAINFILE.c) to executeInstruction()function(resides in Execution.h header file). Where it identifies which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in Logical.h header file where control is transferred to Itype() function(resides in Register.h header file). Itype() function recognizes which register is used as destination register and which is as source register. It finds value of immediate and also ensures that immediate is 12-bit value and after that it returns control to Logical.h. Here, appropriate operation is performed on content of source register and immediate and result is stored in destination register. Now, control is returned to executeInstruction() function and from here control is returned to main() function where pc is increased to point next instruction.



Control Flow of Execution of Logical Instruction(op rd, rs1, imm)

Execution of Compare Instruction:-

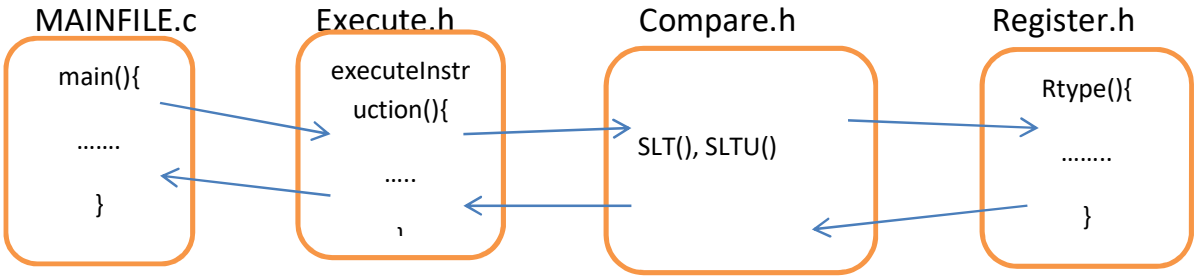
Different formats of Arithmetic instruction

op rd, rs1, rs2
op rd, rs1, imm

When instruction is in format op rd, rs1, rs2.

slt rd, rs1, rs2
sltu rd, rs1, rs2

In process of execution of program when pc points one of these instruction control is transferred from main() function(resides in MAINFILE.c) to executeInstruction()function(resides in Execution.h header file). Where it identifies which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in Compare.h header file where control is transferred to Rtype() function(resides in Register.h header file). Rtype() function recognizes which register is used as destination register and which are source registers and after that it returns control to Compare.h. Here, appropriate operation is performed on content of source register and result is stored in destination register. Now, control is returned to executeInstruction() function and from here control is returned to main() function where pc is increased to point next instruction.

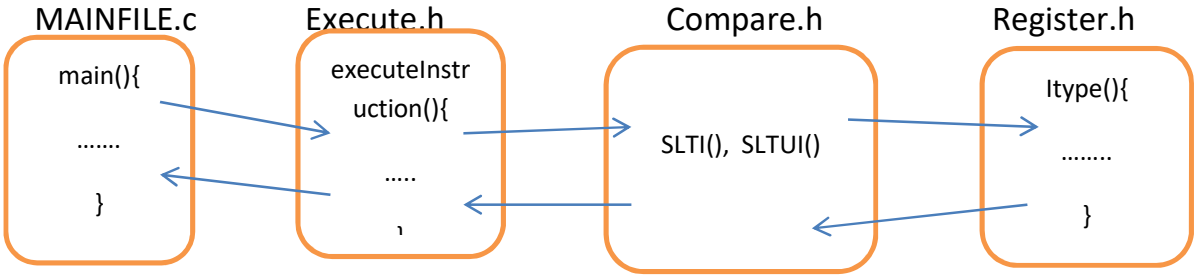


Control Flow of Execution of Compare Instruction(op rd, rs1, rs2)

When instruction is in format op rd, rs1, imm.

slti rd, rs1, imm
sltui rd, rs1, imm

In process of execution of program when pc points one of these instruction control is transferred from main() function(resides in MAINFILE.c) to executeInstruction()function(resides in Execution.h header file). Where it identifies which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in Compare.h header file where control is transferred to Itype() function(resides in Register.h header file). Itype() function recognizes which register is used as destination register and which is as source register finds value of immediate and also ensures that immediate is 12-bit value and after that it returns control to Logical.h. Here, appropriate operation is performed on content of source register and immediate and result is stored in desination register. Now, control is returned to executeInstruction() function and from here control is returned to main() function where pc is increased to point next instruction



Control Flow of Execution of Compare Instruction(op rd, rs1, imm)

Execution of Shift Instruction:-

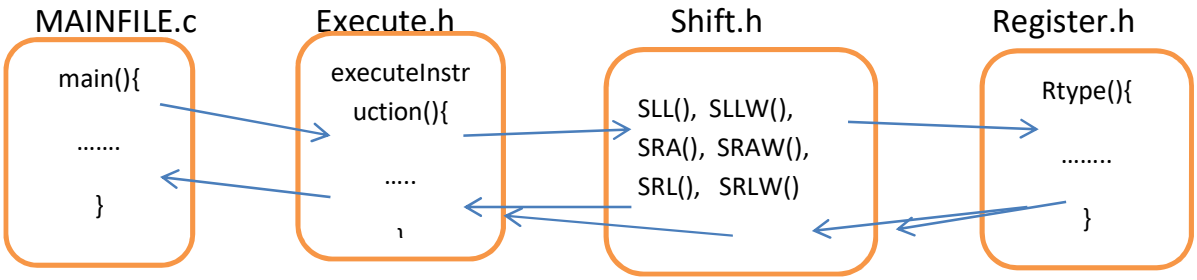
Different formats of Shift instruction:-

op rd, rs1, rs2
op rd, rs1, imm

When instruction is in format op rd, rs1, rs2.

sll rd, rs1, rs2
sllw rd, rs1, rs2
sra rd, rs1, rs2
sraw rd, rs1, rs2
srl rd, rs1, rs2
srlw rd, rs1, rs2

In process of execution of program when pc points one of these instruction control is transferred from main() function(resides in MAINFILE.c) to executeInstruction()function(resides in Execution.h header file). Where it identifies which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in Shift.h header file where control is transferred to Rtype() function(resides in Register.h header file). Rtype() function recognizes which register is used as destination register and which are source registers and after that it returns control to Shift.h. Here, appropriate operation is performed on content of source register and result is stored in destination register. Now, control is returned to executeInstruction() function and from here control is returned to main() function where pc is increased to point next instruction.

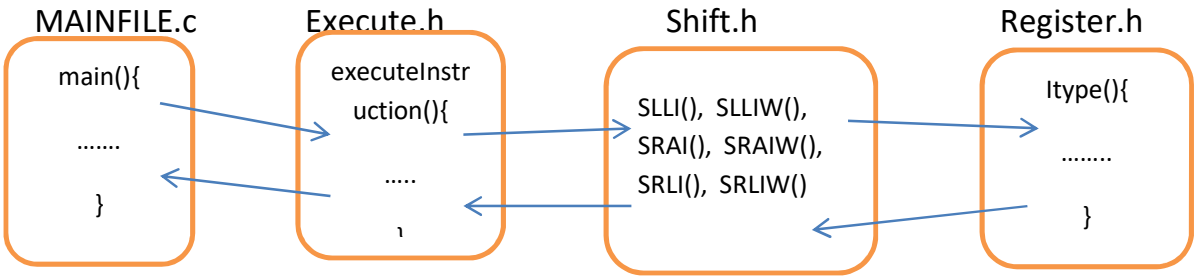


Control Flow of Execution of Shift Instruction(op rd, rs1, rs2)

When instruction is in format op rd, rs1, imm.

slli rd, rs1, rs2
slliw rd, rs1, rs2
srai rd, rs1, rs2
sraiw rd, rs1, rs2
srli rd, rs1, rs2
srliw rd, rs1, rs2

In process of execution of program when pc points one of these instruction control is transferred from main() function(resides in MAINFILE.c) to executeInstruction()function(resides in Execution.h header file). Where it identifies which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in Shift.h header file where control is transferred to Itype() function(resides in Register.h header file). Itype() function recognizes which register is used as destination register and which is as source register finds value of immediate and also ensures that immediate is 12-bit value and after that it returns control to Shift.h. Here, appropriate operation is performed on content of source register and immediate and result is stored in destination register. Now, control is returned to executeInstruction() function and from here control is returned to main() function where pc is increased to point next instruction.



Control Flow of Execution of Shift Instruction(op rd, rs1, imm)

Execution of DataTransfer Instruction:-

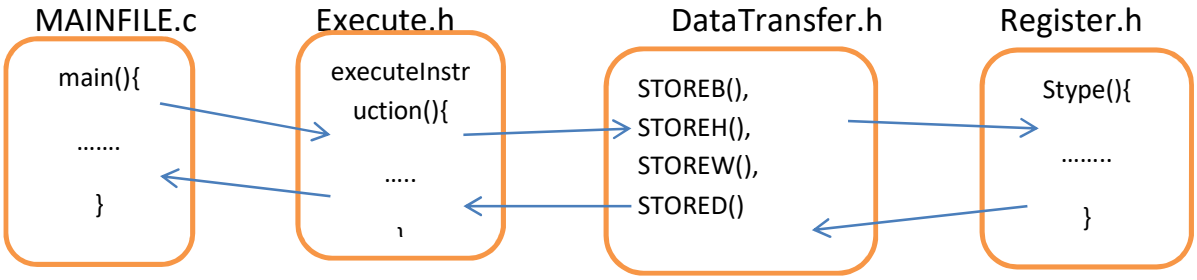
Different formats of DataTransfer instruction:-

op rs1, imm(rs2)
op rd, imm(rs1)

When instruction is in format op rs1, imm(rs2).

sb rs1, imm(rs2)
sh rs1, imm(rs2)
sw rs1, imm(rs2)
sd rs1, imm(rs2)

In process of execution of program when pc points one of these instruction control is transferred from main() function(resides in MAINFILE.c) to executeInstruction()function(resides in Execution.h header file). Where it identifies which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in DataTransfer.h header file where control is transferred to Stype() function(resides in Register.h header file). Stype() function recognizes which registers are used as source registers. It also finds value of immediate and also ensures that it can't be more than 12-bit value and after that it returns control to DataTransfer.h. Here, appropriate operation is performed. Now, control is returned to executeInstruction() function and from here control is returned to main() function where pc is increased to point next instruction.

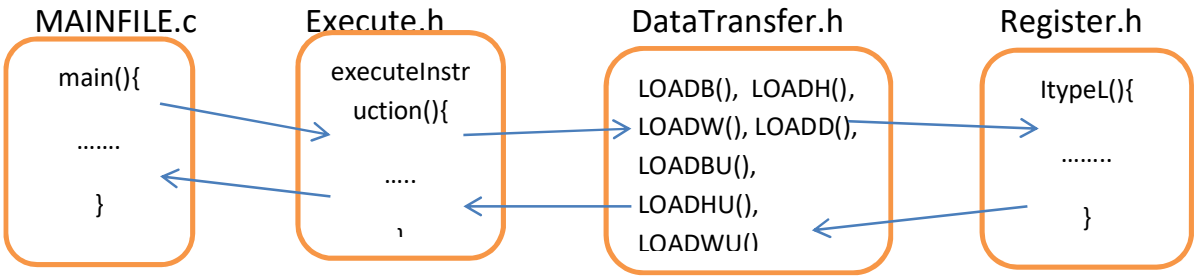


Control Flow of Execution of DataTransfer Instruction(op rd, rs1, rs2)

When instruction is in format op rd, imm(rs1).

lb rd, imm(rs1)
lh rd, imm(rs1)
lw rd, imm(rs1)
ld rd, imm(rs1)
lbu rd, imm(rs1)
lhu rd, imm(rs1)
lwu rd, imm(rs1)

In process of execution of program when pc points one of these instruction control is transferred from main() function(resides in MAINFILE.c) to executeInstruction()function(resides in Execution.h header file). Where it identifies which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in DataTransfer.h header file where control is transferred to ItypeL() function(resides in Register.h header file). ItypeL() function recognizes which register is used as destination register and which is as source register. It finds value of immediate and also ensures that immediate is 12-bit value and after that it returns control to DataTransfer.h. Here, appropriate operation is performed on content of source register and immediate and result is stored in destination register. Now, control is returned to executeInstruction() function and from here control is returned to main() function where pc is increased to point next instruction.



Control Flow of Execution of DataTransfer Instruction(op rd, imm(rs1))

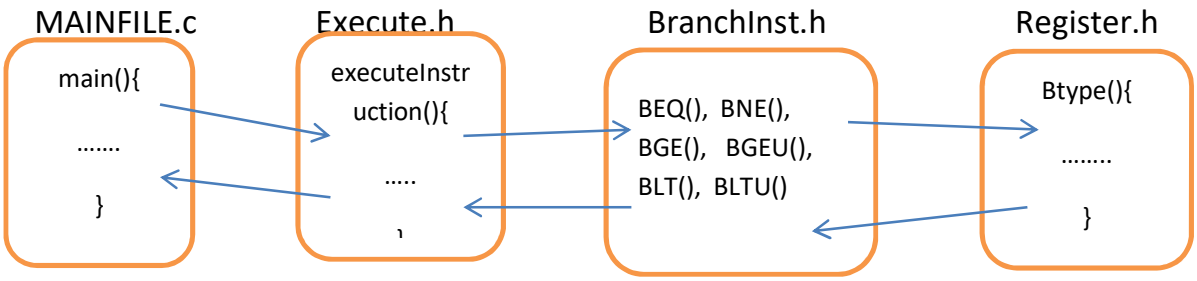
Execution of Branch Instruction:-

When instruction is in format op rs1, rs2, imm/label.

beq rs1, rs2, imm/label
bne rs1, rs2, imm/label
bge rs1, rs2, imm/label
bgeu rs1, rs2, imm/label
blt rs1, rs2, imm/label
bltu rs1, rs2, imm/label

In process of execution of program when pc points one of these instruction control is transferred from main() function(resides in MAINFILE.c) to

executeInstruction()function(resides in Execution.h header file). Where it identifies which operation is going to be performed and then transfers control from executeInstruction() function to corresponding function that resides in BranchInst.h header file where control is transferred to Btype() function(resides in Register.h header file). Btype() function recognizes which registers are used as source registers. It also finds value of immediate and also ensures that it can't be more than 12-bit value and after that it returns control to BranchInst.h. Here, appropriate operation is performed. Now, control is returned to executeInstruction() function and from here control is returned to main() function.



Control Flow of Execution of Branch Instruction(op rs1, rs2, imm/label)