Worksheet: RISC-V Machine Language 1

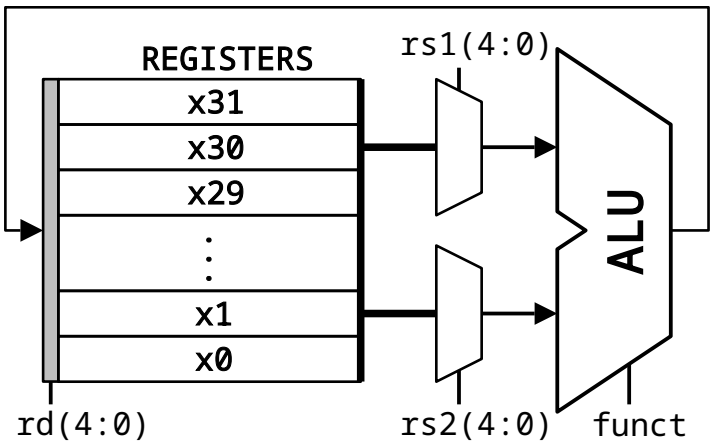
The most common processors today are probably the x86 processors, which are used in most Microsoft Windows-based PCs, and ARM processors, which are found in almost all smart phones and in the newer (non-Intel) Apple Macintosh computers. We will discuss the differences between these processors after we learn more about how processors work.

For our learning, we will use RISC-V (the "V" is the roman numeral for "five", so we pronounce it as "risk five"). Although less common that the two processors mentioned above, I choose to examine this processor rather than the others for two reasons. The first is that its implementation is simpler than x86. The second is that the architecture is free and open-source. (We will also discuss open-source at another time).

There are a number variations of RISC-V. Some variations are based on the width of the integer registers. The two common variants are the 32-bit and 64-bit variants, but there is also the specification for a 128-bit variant that is currently less common. Regardless of the variant, all instructions are 32 bits in width.

The standard RISC-V processors have 32 registers inside the processor. These registers provide storage for numbers that the processor is currently using in computations.

The figure to the right shows a diagram of a RISC-V arithmetic logic unit (ALU), along with 32 registers. (The diagram does not specify the width of each register.)

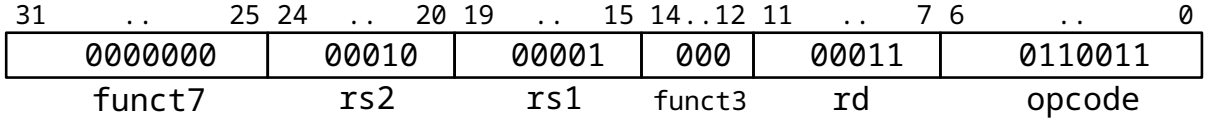


Say, hypothetically, we wish to add register `x1` to register `x2` and store the result in register `x3`. We would need to:

- set `rs1` to binary `00001`
- set `rs2` to binary `00010`
- set `rd` to binary `00011`
- set `funct` to whatever binary value is appropriate for addition.

Ensure you follow the diagram and understand how to come up with these values.

To create the RISC-V ***machine-language instruction***, ***t***he binary values required to carry out this instruction are placed into specific positions as defined by the RISC-V *instruction set* as shown in the diagram below.



The `opcode` tells the processor how to decode the remaining bits of the instruction.

The `opcode` binary value `0110011` tells the processor that the instruction is a register-to-register instruction. This means, as we have just done, the two sources of the computation are both one of the 32 internal registers, and that the result is also to be stored in one of these registers.

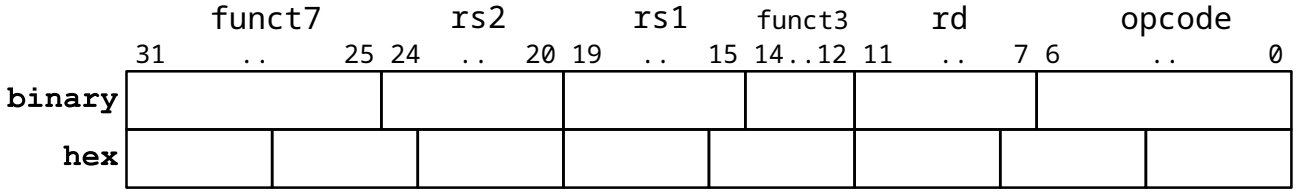
The table to the right shows some of the RISC-V register-to-register operations that can be performed, and the values that `funct7` and `funct3` must be set to in order to perform these operations.

Operation	funct7	funct3	assembly
addition	`0000000`	`000`	`add`
subtraction	`0100000`	`000`	`sub`
shift left	`0000000`	`001`	`sll`
bitwise XOR	`0000000`	`100`	`xor`
shift right (logical)	`0000000`	`101`	`srl`
shift right (arithmetic)	`0100000`	`101`	`sra`
bitwise OR	`0000000`	`110`	`or`
Bitwise AND	`0000000`	`111`	`and`

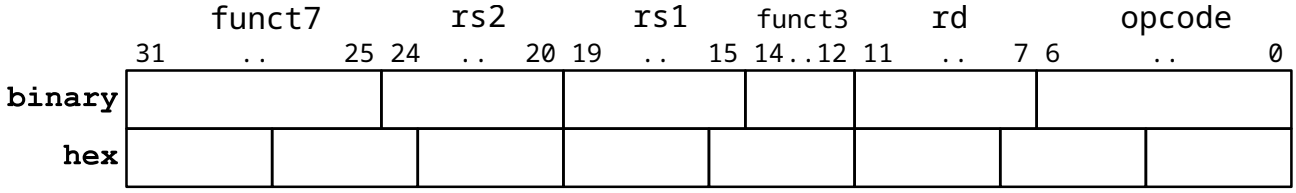
Worksheet: RISC-V Machine Language 1

Fill in the boxes according to the instructions. All instructions are register-to-register instructions, so the `opcode` for them will be `0b0110011`. In RISC-V assembly, the 32 general purpose register labels use the letter `x` followed by a number between `0` and `31`. The destination register comes after the instruction, followed by `rs1` and `rs2`. The instruction, destination, and operands are separated by a space. Thus, the order of a RISC-V assembly instruction is: `<instruction> <rd>, <rs1>, <rs2>`. An example is given in question 1.

1. Fill out the table below for the RISC-V machine language instruction that subtracts register `x25` from register `x9`, and stores the result in register `x3`. In assembly, this instruction is: `sub x3, x9, x25`; in a higher language, it might be: `x3 = x9 - x25`.



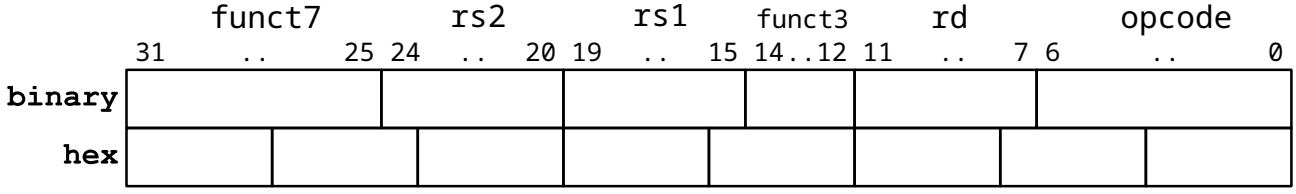
2a. Fill out the table below for the RISC-V machine language instruction that performs an arithmetic shift right of register `x7` by the amount given in register `x1`, and stores the result in register `x6`.



2b. Write the assembly code for the above instruction.

2c Write a high-level language for the instruction.

3a. Fill out the table below for the RISC-V machine language instruction that performs a bitwise `AND` of register `x28` (in `rs1`) and register `x15` (in `rs2`), and stores the result in register `x25`.



3b. Write the assembly code for the above instruction.

3c. Write a high-level language for the instruction.

After you have completed the above, use the ***Interactive RISC-V ALU*** online to verify your answers.

Challenge: use the online ***Interactive RISC-V ALU*** as initialized on load and write a single assembly language instruction that will save the value `0xFFFF_FFFF` to register `x1`.