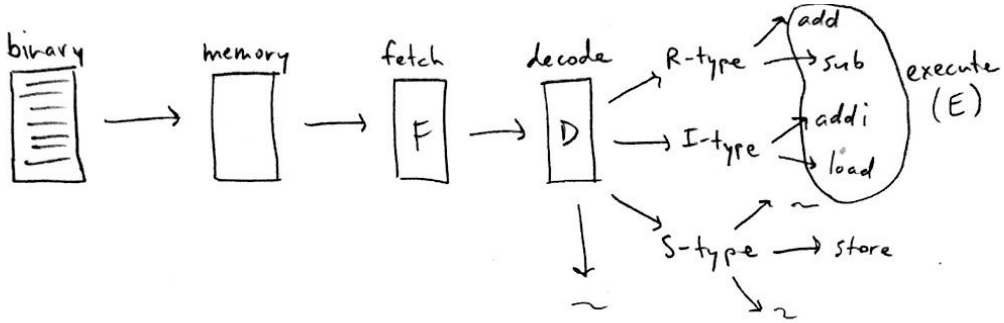

binary file (feeds in instructions to run)

*not provided
- we write this, could use C code and cross compile or get online pre-compiled binaries, or just get it from someone else (doesn't matter where we get it as long as we cite it)

2D array - memory
2D array - registers
} provided

When you run the simulator, it takes the binary file and puts all the instructions in memory.

Program is structured around instruction stages : F D X M W
- Each letter needs its own function (helps w/ pipelining)

(F.) Fetch the instruction from memory, using address (provided by the program counter (PC)).
32-bit instructions ← fetch these!
memory-read (PC address?, pointer for getting the instruction?, 32 bits)

↓

(D.) Mask opcode (huge case comparison)

↓

(X.) Should call one ~~of~~ several functions that say what to do with the data.

↓

(M, W) Specific to the operation

Handy RISC-V instruction guide:
http://inst.eecs.berkeley.edu/~cs61c/fa17/img/riscvcard.pdf

Assignment Requirements:

Instructions we need to support:

- add, addw
- addi, addiw
- and
- andi
- auipc
- beq
- bge
- bgeu
- blt
- bltu
- bne
- jal
- jalr
- lb
- lbu
- ld
- lh
- lhu
- lui
- lw
- lwu
- or
- ori
- sb
- sd
- sh
- sll, sllw
- slli, slliw
- slt
- slti
- sltiu
- sltu
- sra, sraw

- srai, sraiw
- srl, srlw
- srli, srliw
- sub, subw
- sw
- xor
- xori
- mul, mulw
- mulh
- mulhu
- mulhsu
- div, divw
- divu, divuw
- rem, remw
- remu, remuw

Other Deliverables:

- README file (ASCII file): a top-level description of the program and includes and notes that may be helpful to the graders
- Makefile: our makefile should build our program on the UNIX timeshare.

How to decode each instruction:

Reference page 120 in the textbook.

Questions:

Question 1: For this assignment, should we assume that every program counter refers to an instruction, and not a piece of data or something else?

Question 2: Should we use #defines for every instruction, or are they already predefined?

Every type of instruction is decoded by first reading the last 7 bits of the instruction, called the *opcode.*

We will be considering 6 types of instructions: R-type, I-type, S-type, SB-type, UJ-type, and U-type instructions.

While reading the opcode, we find out what type of instruction it is.
Every opcode has the last two bits set high. That's the defining characteristic of an instruction.

The first 5 bits will determine what instruction it is. Let's call it X.

If X is 00000, then it is a load instruction, and it is an I-type instruction. Refer to funct3 in the instruction to determine how many bits to load.

If X is 00100, then it is an arithmetic function, and it is an I-type instruction. Refer to funct3 to determine what type of arithmetic function (addi, slli, slti, xori, srli, srai, ori, andi).

If X is 00110, then the instruction is a wide arithmetic function of I-type. Example: addiw, ssliw, srliw, sraiw. Refer to funct3 and funct7 to determine the type.

If X is 00101, then it is a U-type instruction. There's auipc and lui.

If X is 01000, then it is an S-type instruction (sb, sh, sw, sd). Refer to funct3 to determine how many bits you're storing.

If X is 01100, then it is an R-type instruction. Refer to funct3 and funct7 to determine the type of instruction you will be executing.

If X is 11000, then it is an SB-type instruction. Refer to funct3 to determine the branch-making process.

If X is 11001, then it is the jalr instruction.

If X is 11011, then it is the jal instruction.

R-type instructions:

Description: Is an Arithmetic instruction format

7 5 5 3 5 7 ←-- number of bits per field

[funct7][rs2][rs1][funct3][rd][opcode]

Examples:

- add
- sub
- xor

I-type instructions:

Description: Loads and immediate arithmetic

Examples:

- addi
- xori
- load

S-type instructions:

Description: Stores

Examples:

- sd

SB-Type instructions:

Description: Conditional branch format

Examples:

- beq
- bge
- bgeu

UJ-Type instructions:

Description: Unconditional Jump format

Examples:

- jal

U-Type instructions:

Description: Upper immediate format

Examples:

- lui
- auipc