ISA Project Documentation

- **Student Names and IDs:**

  Nadav Mishan - 209363662

  Shachar Weinberg- 313132284

  Github Link - https://github.com/NadavMishan/Computer-Organization/tree/main

The Assembler:

The assembler converts the assembly files into two output files:

1. **dmemin**: Contains the initial memory image.
2. **imemin**: Contains the instructions in hexadecimal format.

The assembler processes the input file line by line and performs the following steps:

- Creates a copy of the original file and performs all operations on the copied file.
- **Removes headers**:
  The assembler deletes the headers and stores them in a data array to enable the execution of BRANCH and JAL functions according to their required locations.
- **Removes comments**:
  Deletes comments marked with an asterisk (#).
- **Reads memory and instruction data**:
  Extracts information about memory and the instructions to be executed and writes them into the respective output files (dmemin and imemin).

  Deletes the copied file:
  After processing, the temporary copy of the file is deleted.

The Simulator:

The simulator executes the machine code written by the Assembler.

We initialise the variables and get into an infinite while loop that runs the fetch-decode-execute loop (simMain.c).

- **Fetch**:

  Reads the line from imemin.txt based on the PC

- **Decode**: (decodeInstruction() in InstructionActions.h)

  Breaks down the command into Opcode, rd, rs, rt, rm, imm1, imm2

- **Execute**: (executeInstruction____() in InstrucitonActions.h)

  Execute the instruction based on the instruction set given, we broke it down to 4 types

Basic: Basic arithmetic and branching, OPcodes 0->15

LwSw: Load word and store word, OPcodes 16,17 (LoadwordStoreword.h)

IO: Hardware instructions, OPcodes 18,19,20

halt: Exist program, OPcode 21

Inside the loop we also run the “parallel” hardware components and the intrerupt check

- **Interrupts**: (Interrupts() in hardware.h)

  Each loop checks if the timer/disk/irq2 interrupt needs to be raised. also checks if we need to jump to the ISR.

- **Hardware Cycle**: (HardwareCycle() in hardware.h)

  Runs the “parallel” Hardware cycle.

  Clock: Increments the clock by one each loop

  Timer: If the timer is enabled raises the timer by +1 and resets if needed.

  Monitor: Checks for write actions

  Disk: Checks for read/ write actions and runs the Disk clock that raises the interrupt when we are done with a read/write action. (diskActions.h)

We are also logging all the required output files- inside the loop in append mode or after a halt command in write mode. (loggers.h)

We also wrote 2 utility files to help us run the project. one to help us read/ write to files and one to help us with type conversions . (fileActions.h , utils.h)

Assembly Test Programms:

- **Mulmat** – Performs the multiplication of two 4x4 matrices using the following principles:
  - For each element in the output matrix, 4 multiplications are performed.
  - The accumulator register V0 is used to perform the multiplication and accumulate the sum without losing the result of the previous multiplication.
  - The multiplication is performed between a row in the first matrix and a column in the second matrix.
  - After completing 16 multiplications (one row of the first matrix), the program proceeds to the next row in the first matrix- enabled by $s2.

- ❖ The columns in the second matrix are accessed repeatedly throughout all multiplications, so it is necessary to enable returning to the required column each time- enabled by $t2 and the function “next”.

- ❖ A total of 64 multiplications are performed, and after storing the results, the program halts- enabled by $s1.

- **Binom**- the program calculates the Newton binom coefficient recursively according to the algorithm, based on the following principles:
  - ❖ In each call to the binom function, 4 slots are allocated on the stack to store n, k, the return address, and the value of the $s0 register, which holds the result returned by the function in the relevant call.
  - ❖ During each function call, binom(n-1, k) and binom(n-1, k-1) are calculated recursively, and their results are added together as required, based on the relevant values stored in the stack during that call.
  - ❖ After the calculation, the allocated stack space is freed, and execution returns to the required point in the program.
  - ❖ The function considers the relevant stopping condition of the algorithm.
- **Circle**- draws a full circle on the screen, that placed at the center of the screen, in the white color. based on the following principles:
  - ❖ A pixel will be lit if it satisfies the circle equation: x2+y2<$v0=radius^2 (where x represents the horizontal distance from the circle's center, and y represents the vertical distance.

  - ❖ The “search” function finds the first pixel (in the current row) that needs to be lit according to the equation. Once found, it passes control to the “light” function.

  - ❖ The “light” function is responsible for lighting all relevant pixels in the row that satisfy the circle equation. If a pixel does not satisfy the equation, the function stops processing the current row and proceeds to the next row, as the remaining pixels in that row are also irrelevant.

  - ❖ The transition to the next row is handled by the “fix” function.

- **Disktest**- Moves the data from sectors 0->7 to sectors 1->8
  - ❖ The disk works by copying sector 0->7 to the memory and after we finish reading it writes the saved memory to the disk at sectors 1->8
  - ❖ After each Read/Write operation we wait for diskstatus to return to 0.