Name: Ai Vi Tran
Student ID: 104209393

1/
1.1/ ROM(read only memory): All reading at full speed (just get the address and go there)
 - Content built-in at time of manufacture.
1.2/ RAM (Random access memory)
 - Ram is volatile memory that temporarily stores the files you are working on
 - Rom is non-volatile memory that permanently stores instructions for the computer.
1.3/ Static RAM – retains information until power removed. Fast, larger area of silicon per byte, modest power requirement.
> Dynamic RAM - retains information as long as the contents are refreshed frequently enough. Smaller area of silicon per byte, low power requirement.
– Does not use flip-flops.
– Uses tiny capacitors to store electric charge.
– Because the charge leaks away have to rewrite ("refresh") every few milliseconds.
1.4/ Flash Memory (EEPROM)
 - Charge stored between insulators. Write bit by injecting electrons through a barrier layer (physically damaging it). Used in USB drives. Good for about 30,000 writes.

2/ (2^10)^3 = 2^30 => 30 bits are needed to address all bytes.

3/ > Von Neumann:
– Data and Instructions stored in same location
– Stack central to handling multiple tasks/interrupts
> Harvard:
– Separates data an instructions
– Increased efficiency and security
– Reduced generality and versatility

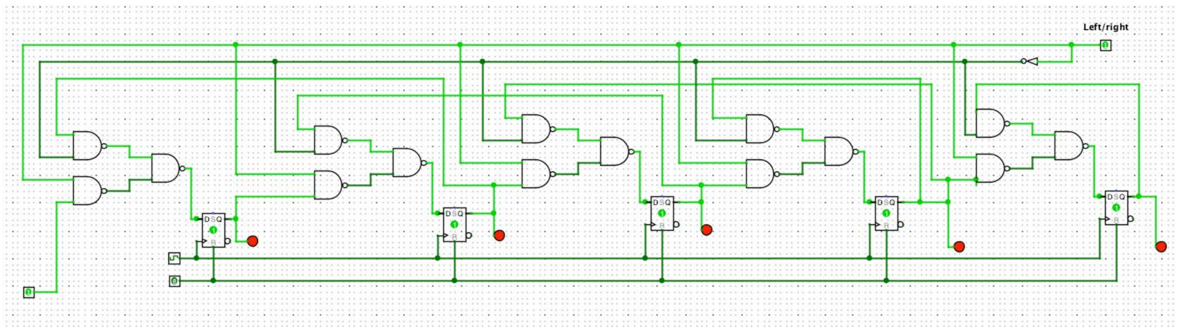
4/ CPU CACHES
> Store frequently accessed instructions/data in high-speed memory.
> What's in cache depends on caching algorithms
> Can have separate Instruction and Data caches:
– Instructions smaller, more predictable format so have dedicated hardware (read access) to data caches (RW access).
– Increases processing speed – Instructions and Data can be loaded at the same time.
– May save power (can power-down cache if not in use)
> Can have separate layers (L1, L2, L3) depending on speed/frequency requirement.

5/ I NTERRUPTS
Stacks allow interrupt-based hardware access.
> A device (e.g. I/O) issues an electrical signal, which feeds into a priority encoder which then issues an INT signal to the CPU (depending on relative priority).
> The current work of the CPU (including the value of the IP) is pushed onto the stack.
> The CPU loads the Interrupt's handler routine (code which specifies what to do when interrupted by the specific hardware), executes the code.
> The INT Handler ends with a RETurn instruction, which pops the stored IP off the stack into the IP, and processing resumes

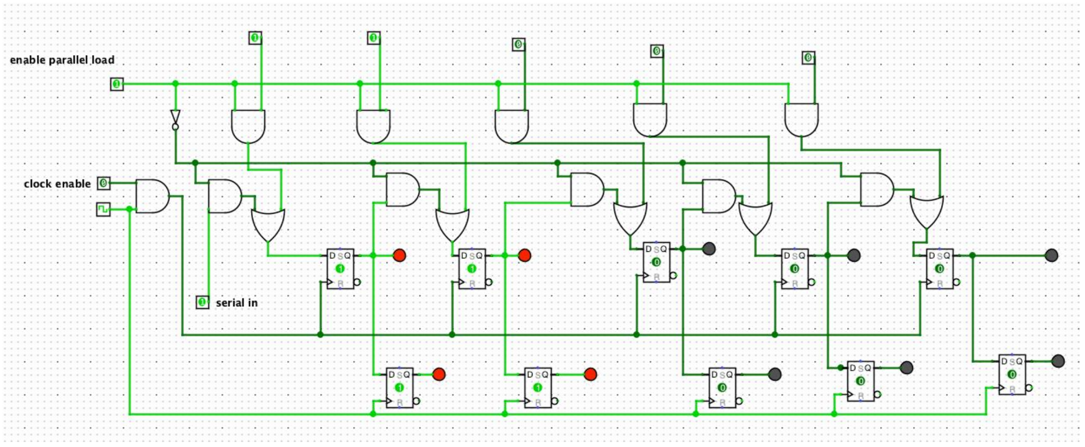
6/ STACKS
> Random access memory requires knowing the address of every byte/word you want to access
> Stacks offer a way of organising and accessing memory without random (indexed) access:
– There are hardware stacks and software stacks.
> Hardware stacks created out of dedicated shift registers
> Software stacks typically defined in RAM using conventions (we'll come back to this):

I N SIMPLE TERMS...
> A stack allows us to mothball/backup/hibernate a process/ task at will on the receipt of an interrupt or code invocation.
> To do this, we
1. push instructions/data that we will need later onto the stack;
2. do the task;
3. and then pop the stored data back off the stack and
4. continue as before.

11/



12/


enable parallel load
clock enable
serial in