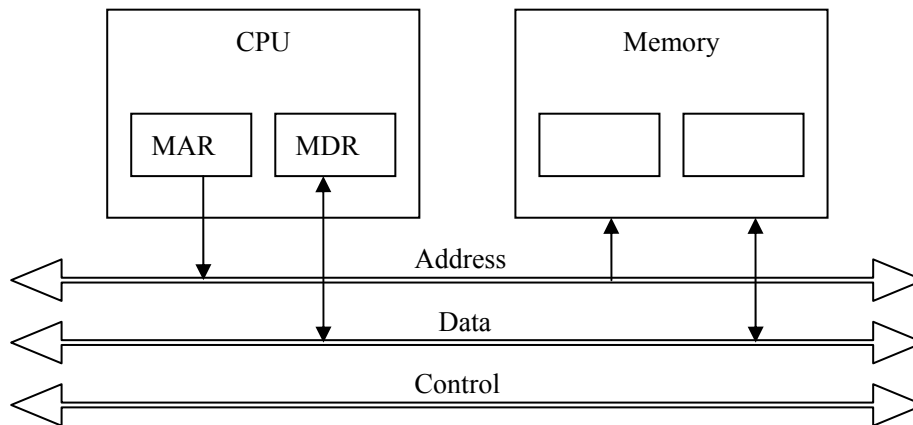


# Lecture 7

## Registers, Memory, Cache



Memory bus of modern machines

- *MAR* may or may not be there in *CPU*.
- Width of Address bus, Data bus and Control bus.
- Address and Data bus is sometimes shared.
- Write operation will be affected.
- Why did the designers of the *JVN* machine choose 40 bits for word size?
- Instruction size was chosen to be 20 bits, (8 for *opcode* and 12 for address)
- Now 20 bits are not sufficient to hold (reasonably) large numbers.
- We need 30-35 bits to do any meaningful calculations.
- With a 35-bit word, we waste a lot of bits in the program area since one word would hold only one instruction.
- With 40 bits, two instructions can be stored in one word, saving time and space.

## Problems with Accumulator Machine

- Most instructions involve use of *ACC*.
- $X = A + B$  is our example.
- This is translated as:

LOAD A	}	All three instructions require a memory access.
ADD B		
STORE X		

- While executing our program, memory is accessed in almost every instruction.
- Typically, memory speed is much less than that of *CPU* (*ALU* + control).
- Even if our *CPU* is fast, performance depends on speed of memory.
- Memory & *CPU* chips are made using different technology.
  
- In early machines magnetic main memory was used (very slow).
- These days we use *DRAM* (Dynamic Random Access Memory) chips.
- *DRAM* chips are slower than *CPU* chips. So we still have the problem.

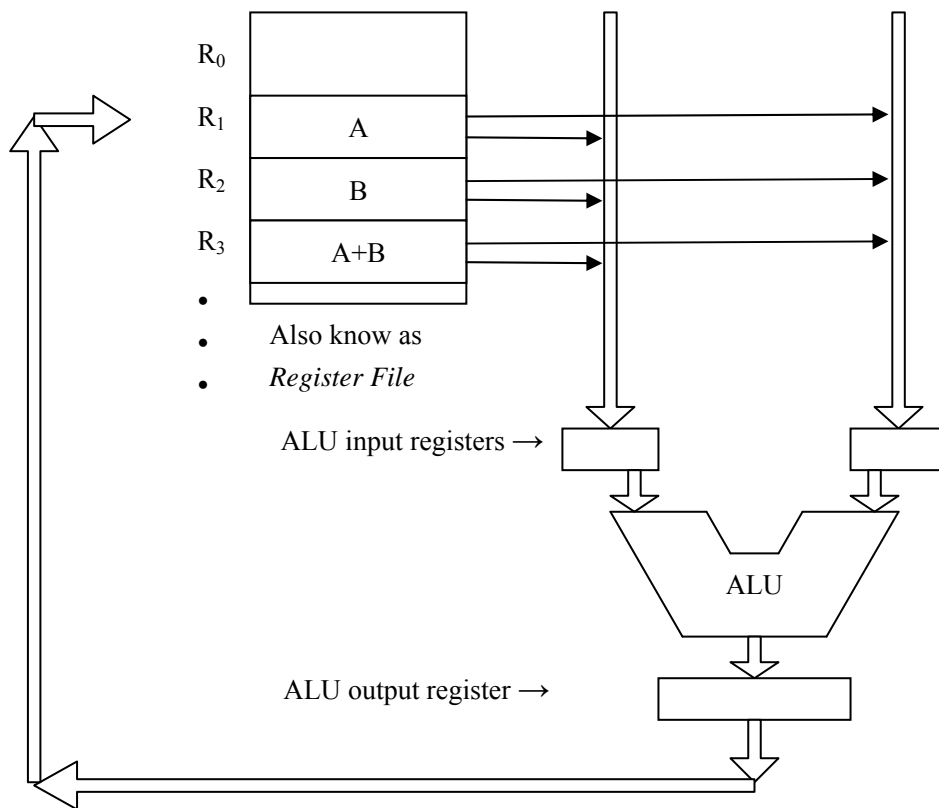
Issue: *CPU* and memory have speed mismatch.

Two Solutions:      1. **Registers**      and      2. **Cache Memory**

Use of registers: What if we provide some extra memory locations inside *CPU*, with same speed?

- Frequently required data items are stored/ kept in these locations (called *registers*)
- In the previous example, *A* & *B* would be in registers and *X* would also be stored in a register.
- When *A*, *B* and *X* are no longer required, they are written back to memory.
- Assumption: Other instructions may require *A*, *B* and/or *X*, and their execution will be very fast.
- Initially we have to read *A* & *B* from memory, and in the end we have to write *X* back to memory. We do spend some time doing this.
- Typically 8, 16, 32 or 64 registers are provided. These are known as *general purpose* registers or *special purpose* registers, depending on how they are used.

## Register Machine



Data Path of a register machine

## Register Machine

- Conceptually, it is still a *JVN* machine.
- It has *PC*, *IR* and other registers.
- *MAR* may not be present if registers have direct connection to address bus.
- Fetch-Decode-Execute cycle is similar to what we have seen.
- Most instructions now use registers

Example:      Add  $R_1, R_2$       means       $R_1 = R_1 + R_2$

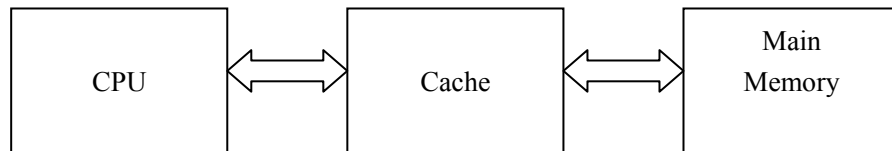
                 or, Add  $R_1, R_2, R_3$       may mean       $R_1 = R_2 + R_3$

- Two/three operand instructions.

## CPU & Memory Speed Mismatch

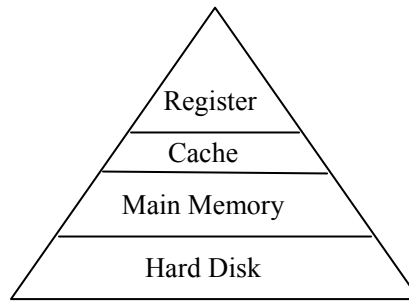
Use of cache memory

- Fast memory, much smaller in size than main memory, but a lot more than just a few registers.
- It is between *CPU* and memory

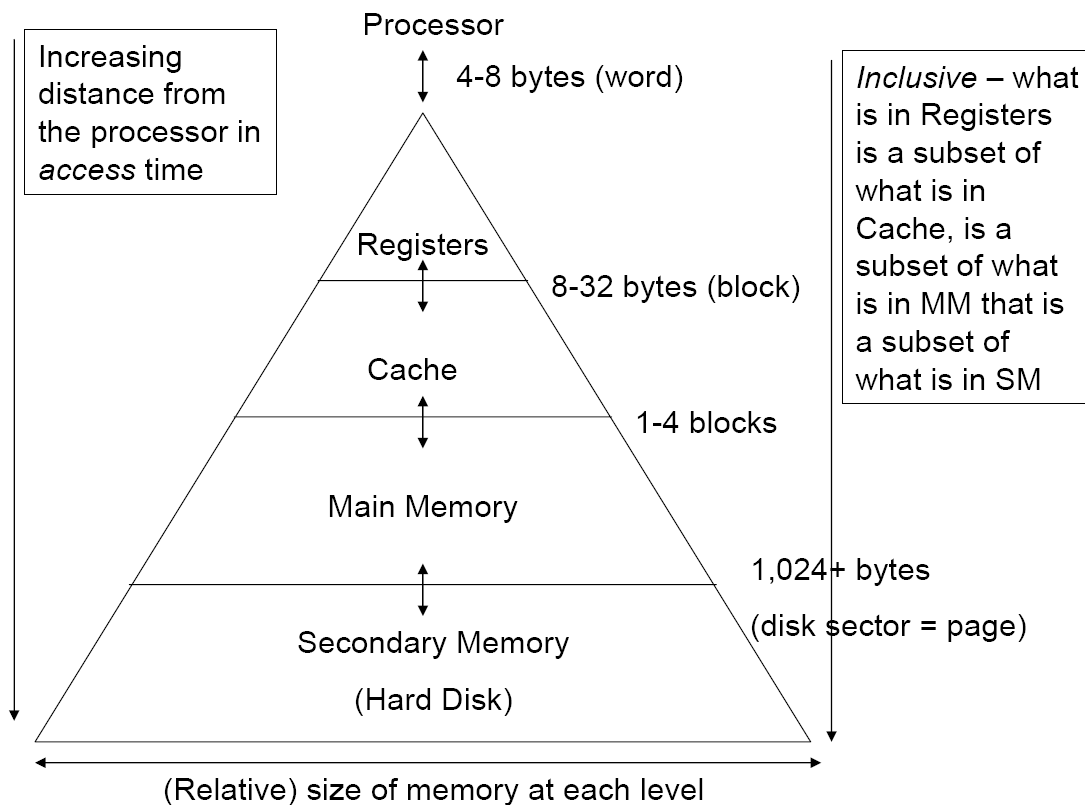


- Based on principle of locality.
- *CPU* always looks for an item by first checking if it is already in the cache.
- If it is found then proceed ('*cache hit*').
- If an item is not found ('*cache miss*') then request is sent to main memory.
- On cache miss, a small block of memory locations (16-64 words) is brought in and stored in cache
- Many of these words will be required eventually, saving us memory accesses.
- This creates an illusion that we are working with a large, fast memory.
- To achieve this effect we need a cache hit rate of 90% or more.
- *SRAM* (Static Random Access Memory) chips are used for the cache.

## Memory Hierarchy

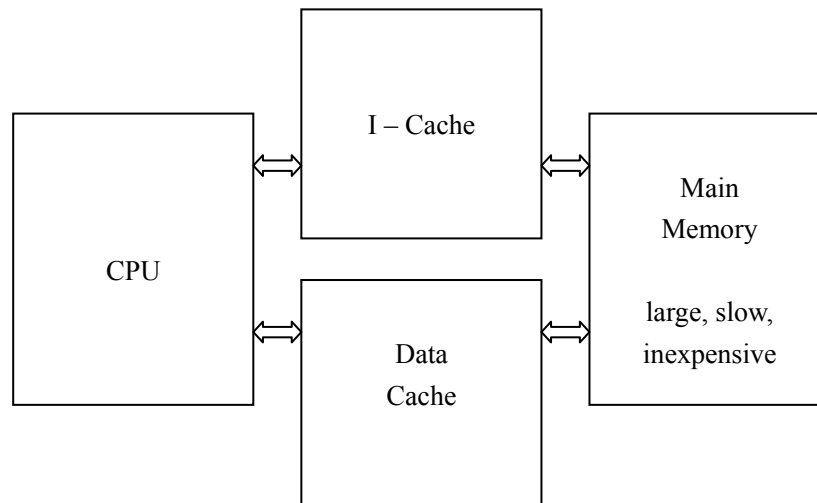


- Fastest device is on the top
- As we move down the pyramid speed drops, but cost per byte also drops.
- When we cannot store data in main memory we have to use disk. All large databases are stored in disks.
- Disks are slower than memory, but offer large capacity at low cost.



## Split Cache

- Success of cache memory depends on locality of locations (addresses) accessed.
- Since program and data are in two different parts of memory, locations accessed during instruction fetch and instruction execution are far apart.
- There are actually two different ‘localities’ intermixed.
- What if we handle these two separately?
- This leads to the idea of a split cache, one for instructions and the other for data.



- Now programs are stored in I-cache.
- Performance of split cache is better than that of unified cache memory (in which program and data are in the same cache).
- With split cache, hit rate can exceed 95%.