Computer Organization & Architecture Formula Notes

Addressing Modes: The most common addressing techniques are

- Immediate Addressing Mode: The operand is actually present in the instruction: OPERAND = A
- Direct Addressing Mode: Address field contains the effective address of the operand: EA = A
- Indirect Addressing Mode: With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is know as indirect addressing: EA = (A)
- Register Addressing Mode: Register addressing is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address: EA = R
- Register Indirect Addressing Mode: Register indirect addressing is similar to indirect addressing, except that the address field refers to a register instead of a memory location. It requires only one memory reference and no special calculation. EA = (R)
- Displacement Addressing Mode: A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing, which is broadly categorized as displacement addressing: EA = A + (R)
- Stack Addressing Mode: A stack is a linear array or list of locations. It is sometimes referred to as a pushdown list or last-in-first-out queue.
- Other modes: Base addressing, indexed, Auto increment, and auto decrement.

Instruction Set Operations:

- Arithmetic/Logical : Integer ALU ops. ADD , AND , SUB , OR .
- Load/Stores : Data transfer between memory and registers. LOAD , STORE (Reg-reg), MOVE (Mem-mem)
- Control : Instructions to change the program execution sequence. BEQZ , BNEQ , JMP , CALL , RETURN , TRAP.

Performance (Relative speed of computer): Performance is inversely proportional to execution time.

- Computer X is n times faster than Y is computed as:

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X}$$

$$n = \frac{\text{Performance}_X}{\text{Performance}_Y} \quad \text{where} \quad \text{Performance} = \frac{1}{\text{Execution Time}}$$

Enhanced Machine Execution time :

$$\text{Exec time}_{new} = \underbrace{\text{Exec time}_{old} \times \left((1 - Fraction_{enhanced}) \right.}_{\text{Time spent using UNenhanced mode}} \left. + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right)$$

Time spent using enhancement

Amdahl's Law:

$$Speedup_{overall} = \frac{ExecTime_{old}}{ExecTime_{new}} = \frac{1}{\left((1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right)}$$

CPU performance : Elapsed user CPU time on an unloaded system.

- CPU Performance Equation:

$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

$$\text{CPU time} = \text{IC} \times \text{CPI} \times \text{Clock cycle time} = \frac{\text{IC} \times \text{CPI}}{\text{Clock rate}}$$

The **program status register** (PSR): It also called the **program status word** (PSW), is one of the special purpose registers found on most computers. The PSR contains a number of bits to reflect the state of the CPU as well as the result of the most recent computation. Some of the common bits are:

C the **carry-out** from the last arithmetic computation
V Set to 1 if the last arithmetic operation resulted in an **overflow**
N Set to 1 if the last arithmetic operation resulted in a **negative** number
Z Set to 1 if the last arithmetic operation resulted in a **zero**
I **Interrupts enabled** (Interrupts are discussed later)

PC: The program counter contains the address of the assembly language instruction to be executed next.

IR : The instruction register contains the binary word corresponding to the machine language version of the instruction currently being executed.

MAR: The memory address register contains the address of the word in main memory that is being accessed. The word being addressed contains either data or a machine language instruction to be executed.

MBR: The memory buffer register (also called MDR for memory data register) is the register used to communicate data to and from the memory.

Data Bus: The data lines provide a path for moving data among system modules. These lines, collectively, are called the data bus.

Address Bus: The address lines are used to designate the source or destination of the data on the data bus.

Control Bus: The control lines are used to control the access to and the use of the data and address lines.

- Control signals transmit both command and timing information among system modules.
- Timing signals indicate the validity of data and address information. Command signals specify operations to be performed.

Typical control lines:

- Memory write: Causes data on the bus to be written into the addressed location.
- Memory read: Causes data from the addressed location to be placed on the bus.
- I/O write: Causes data on the bus to be output to the addressed I/O port.
- I/O read: Causes data from the addressed I/O port to be placed on the bus.
- Transfer ACK: Indicates that data have been accepted from or placed on the bus.
- Bus request: Indicates that a module needs to gain control of the bus.
- Bus grant: Indicates that a requesting module has been granted control of the bus.
- Interrupt request: Indicates that an interrupt is pending.
- Interrupt ACK: Acknowledges that the pending interrupt has been recognized.
- Clock: Is used to synchronize operations.
- Reset: Initializes all modules

Reading Memory:

- First place an address in the MAR.
- Assert a READ control signal to command memory to be read.
- Wait for memory to produce the result.
- Copy the contents of the MBR to a register in the CPU.

Writing Memory:

- First place and address in the MAR
  Copy the contents of a register in the CPU to the MBR.
  Assert a WRITE control signal to command the memory.

Fetch-execute cycle:

- Copy the contents of the PC into the MAR.
- Assert a READ control signal to the memory.
- While waiting on the memory, increment the PC to point to the next instruction
- Copy the MBR into the IR.
- Decode the bits found in the IR to determine what the instruction says to do.

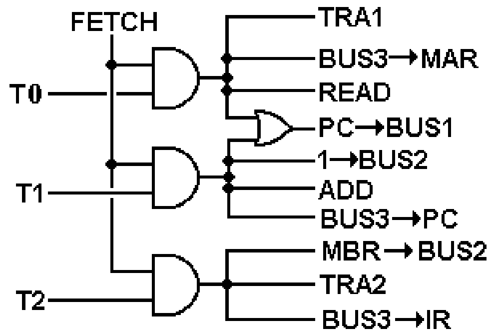
Instruction fetch Control Signals:

T0: PC to Bus1, Transfer Bus1 to Bus3, Bus3 to MAR, READ.
T1: PC to Bus1, +1 to Bus2, Add, Bus3 to PC.
T2: MBR to Bus2, Transfer Bus2 to Bus3, Bus3 to IR.

(OR)

T0: PC $\rightarrow$ Bus1, TRA1, Bus3 $\rightarrow$ MAR, READ.
T1: PC $\rightarrow$ Bus1, +1 $\rightarrow$ Bus2, ADD, Bus3 $\rightarrow$ PC.
T2: MBR $\rightarrow$ Bus2, TRA2, Bus3 $\rightarrow$ IR.

Hardwired Control Unit for Instruction Fetch:



The Fetch Cycle:

t1: MAR ← (PC)
t2: MBR ← Memory
PC ← (PC) + I
t3: IR ← (MBR)

OR

t1: MAR ← (PC)
t2: MBR ← Memory
t3: PC ← (PC) + I
IR ← (MBR)

The Indirect Cycle:

t1: MAR ← (IR(Address))

t2: MBR ← Memory

t3: IR(Address) ← (MBR(Address))

The Interrupt Cycle:

t1: MBR ← (PC)
t2: MAR ← Save_Address
PC ← Routine_Address
t3: Memory ← (MBR)

The Execute Cycle:

- ADD R1, X (adds the contents of the location X to register R1)

$$t1: MAR \leftarrow (IR(address))$$

$$t2: MBR \leftarrow Memory$$

$$t3: R1 \leftarrow (R1) + (MBR)$$

- ISZ X (increment and skip if zero)

$$\begin{aligned} &t1: MAR \leftarrow (IR(address)) \\ &t2: MBR \leftarrow Memory \\ &t3: MBR \leftarrow (MBR) + 1 \\ &t4: Memory \leftarrow (MBR) \\ &\quad \text{If } ((MBR) = 0) \text{ then } (PC \leftarrow (PC) + I) \end{aligned}$$

- BSA X (The address of the instruction that follows the BSA instruction is saved in location X, and execution continues at location X + I. The saved address will later be used for return.)

$$\begin{aligned} &t1: MAR \leftarrow (IR(address)) \\ &\quad\quad MBR \leftarrow (PC) \\ &t2: PC \leftarrow (IR(address)) \\ &\quad\quad Memory \leftarrow (MBR) \\ &t3: PC \leftarrow (PC) + I \end{aligned}$$

Memory Systems:

- In BIG-ENDIAN systems, the most significant byte of a multi-byte data item always has the lowest address, while the least significant byte has the highest address.
- In LITTLE-ENDIAN systems, the least significant byte of a multi-byte data item always has the lowest address, while the most significant byte has the highest address.

Pipelining : Important Stages in Pipelining are:

- Fetch instruction (FI): Read the next expected instruction into a buffer.
- Decode instruction (DI): Determine the opcode and the operand specifiers.
- Calculate operands (CO): Calculate the effective address of each source operand. This may involve displacement, register indirect, indirect, or other forms of address calculation.
- Fetch operands (FO): Fetch each operand from memory.
- Execute instruction (EI): Perform the indicated operation and store the result, if any, in the specified destination operand location.
- Write operand (WO): Store the result in memory.

Note: Pipelining decreases execution time but can increase cycle time.

Ideal pipeline speedup:

$$\frac{\text{Time per instruction on unpipelined machine}}{\text{Number of pipe stages}}$$

$$\text{Speedup from pipelining} = \frac{\text{Average instruction time unpipelined}}{\text{Average instuction time pipelined}}$$

$$\text{Speedup} = \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

Pipeline hazards: A hazard is a condition that prevents an instruction in the pipe from executing its next scheduled pipe stage.

- Structural hazards These are conflicts over hardware resources.
- Data hazards These occurs when an instruction needs data that is not yet available because a previous instruction has not computed (FP pipes) or stored it.
- Control hazards These occur for branch instructions since the branch condition (for compare and branch) and the branch PC are not available in time to fetch an instruction on the next clock.

$$\begin{aligned}\text{CPI pipelined} &= \text{Ideal CPI} + \text{Pipeline stall clk cycles per instuction} \\ &= 1 + \text{Pipeline stall clk cycles per instruction}\end{aligned}$$

$$\text{Speedup} = \frac{\text{CPI unpipelined}}{1 + \text{Pipeline stall cycles per instruction}}$$

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall cycles per instruction}}$$

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline Stall cycles per instruction}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

Structural hazards (Resources hazards): Caused by Functional units not fully pipelined, shared resources among pipelines.

Data hazards: Occur when instruction i generates a result read by instruction i+1. Can occur on the register file, on the program counter (actually control hazards) or on a memory location. RAW, WAR and WAW hazards possible.

- Read after write (RAW), or true dependency: A hazard occurs if the read takes place before the write operation is complete.
- Write after read (RAW), or anti-dependency: A hazard occurs if the write operation completes before the read operation takes place.
- Write after write (RAW), or output dependency: Two instructions both write to the same location. A hazard occurs if the write operations take place in the reverse order of the intended sequence

Control hazards (Branch hazards): Occur when two instructions accesses to the PC are reordered and the first instruction modifies the PC (RAW). Some of the approach to deal with the control hazards are: Multiple streams, Prefetch branch target, Loop buffer, Branch prediction, and Delayed branch.

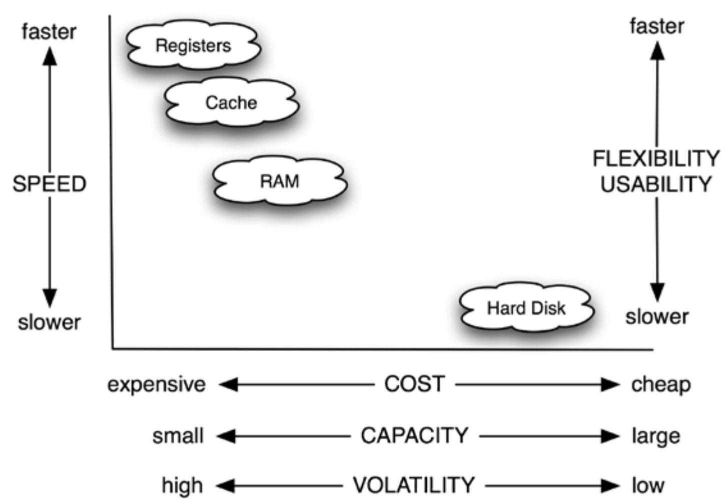
Performance of branches for Control Hazards:

$$\text{Pipeline speedup} = \frac{\text{pipeline depth}}{1 + \text{pipeline stalls from branches}}$$

$$\text{Pipeline stall cycles from branches} = \text{branch frequency} \times \text{branch penalty}$$

$$\text{Pipeline speedup} = \frac{\text{pipeline depth}}{1 + \text{branch frequency} \times \text{branch penalty}}$$

Cache Memory:



$$\text{Memory stall cycles} = \text{IC} \times \text{Mem refs per instruction} \times \text{Miss rate} \times \text{Miss penalty}$$

- IC * Mem refs per instruction: This is the frequency with which the CPU uses memory.
- Miss rate: This is the fraction of references that are not satisfied in the upper level. They require an access to the lower, slower level to be satisfied.
- Miss penalty: The penalty is the length of time it takes to access the lower level. A low miss rate is not much help if the miss penalty is very high.

Direct mapped Cache : Block can only go in one place in the cache (usually address MOD number of blocks in cache).
Fully associative Cache: Block can go anywhere in cache.

Set associative Cache: Block can go in one of a set of places in the cache. A set is a group of blocks in the cache. In a set-associative cache, a block is first mapped to a set by using block address MOD number of sets in the cache. A block may then be placed anywhere in that set. If sets have n blocks, the cache is said to be n-way set associative.

Write-through: In this scheme, the block is written both to the cache and main memory.

Write back (also copy back): In this scheme, only the block in cache is modified. Main memory is modified when the block must be replaced in the cache. This requires the use of a dirty bit to keep track of which blocks have been modified.

Cache Performance:

$$\text{Avg mem access time} = \text{hit time} + \text{miss rate} \times \text{miss penalty}$$

$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory access}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty}\right) \times \text{Clock Cycle Time}$$

Improving Cache Performance: The increasing speed gap between CPU and main memory has made the performance of the memory system increasingly important.

- Reducing the miss rate.
- Reducing the miss penalty.
- Reducing the time to hit in a cache.

Reducing Cache Misses:

- Compulsory : Cold start misses or first reference misses : The first access to a block can NOT be in the cache, so there must be a compulsory miss. These are suffered regardless of cache size.
- Capacity: If the cache is too small to hold all of the blocks needed during execution of a program, misses occur on blocks that were discarded earlier. In other words, this is the difference between the compulsory miss rate and the miss rate of a finite size fully associative cache.
- Conflict: If the cache has sufficient space for the data, but the block can NOT be kept because the set is full, a conflict miss will occur. This is the difference between the miss rate of a non-fully associative cache and a fully-associative cache. These misses are also called collision or interference misses.

Cache Optimization Summary

Technique	Miss Rate	Miss Pen.	Hit time
Larger Block Size	+	-	
Higher Associativity	+		-
Victim Caches	+		
Pseudo-associative	+		
Hardware Prefetching	+		
Compiler-controlled Pre	+		
Compiler Techniques	+		
Giving Read Misses Priority		+	
Subblock Placement		+	
Early Restart/Crit Wd First		+	
Nonblocking Caches		+	
Second-Level Caches		+	
Small and Simple Caches	-		+
Avoiding Address Trans.			+
Pipelining Writes			+

where + for increases, and - for decreases

Magnetic disks:

- Track: A surface is divided into tracks, which are concentric circles containing data.
- Cylinder: The set of tracks at corresponding locations on all of the surfaces is called a cylinder.
- Sector: A track is divided into sectors each of which holds a fixed amount of data, usually 512 - 4096 bytes
- Performance of Magnetic disk (for each access) depends on: Seek time, Rotational delay, and Transfer time.
  Average access time = Seek time + Rotational delay + Transfer time
- Seek time: The seek time is the time necessary to move the arm from its previous location to the correct track for the current I/O request.
- Rotational delay: The time necessary for the requested sector to rotate under the head. (Some disks can read data out of order into a buffer, reducing rotational delay for a large transfer. A full track can be transferred in one rotation regardless of where the I/O actually starts.)
- Transfer time: The transfer time is the time it takes to read or write a sector.
- Density: Disk capacity is measured in areal density (the number of bits per square inch). This is the product of tracks per inch on a surface and bits per inch on a track.
- Transfer rate: Since transfer rate is proportional to bits per track, improvements in density usually lead to higher bandwidth. This increase is usually the square root of overall density increase.

Programmed I/O : Programmed I/O (PIO) refers to using input and output (or move) instructions to transfer data between memory and the registers on a peripheral interface.

- The advantage of PIO is that it is simple to implement. In many cases the CPU will be fast enough to transfer data as fast as the peripherals (e.g. a hard disk) can supply or accept it.
- The disadvantage of PIO is that the CPU is tied up for the duration of the transfer while doing a relatively simple task.

DMA (Direct Memory Access): is a procedure for transferring data directly between controller and memory without the help of CPU.

DMA device: is like a separate CPU with functions related to data transfer, such as:

- shuts the CPU down for short periods
- seizes control of the system bus
- transfers data between memory and external device without the intermediation of CPU

Disk controller: accepts commands from CPU such as:

- select disk drive
- set up DMA registers
- return current sector address
- return current track address
- move head in/out one track
- move head to track 0
- read/write current sector

DMA controllers can operate in a cycle stealing mode in which they take over the bus for each byte of data to be transferred and then return control to the CPU. They can also operate in burst mode in which a block of data is transferred before returning bus control to the CPU. The choice depends on the speed at which data is arriving relative to the bus bandwidth and whether a particular application will allow the CPU to be locked off the bus for the duration of one transfer.

Modes of DMA:

- Single A single byte (or word) is transferred. The DMA must release and re-acquire the bus for each additional byte. This is commonly-used by devices that cannot transfer the entire block of data immediately. The peripheral will request the DMA each time it is ready for another transfer. The floppy disk controller only has a one-byte buffer, so it uses this mode.
- Block/Demand Once the DMA acquires the system bus, an entire block of data is transferred, up to a maximum of 64K. If the peripheral needs additional time, it can assert the READY signal. READY should not be used excessively, and for slow peripheral transfers, the Single Transfer Mode should be used instead. The difference between Block and Demand is the once a Block transfer is started, it runs until the transfer count reaches zero.
- Cascade This mechanism allows a DMA channel to request the bus, but then the attached peripheral device is responsible for placing addressing information on

the bus. This is also known as ``Bus Mastering". When a DMA channel in Cascade Mode receives control of the bus, the DMA does not place addresses and I/O control signals on the bus like it normally does. Instead, the DMA only asserts the -DACK signal for this channel.

- Autoinitialize This mode causes the DMA to perform Byte, Block or Demand transfers, but when the DMA transfer counter reaches zero, the counter and address is set back to where they were when the DMA channel was originally programmed. This means that as long as the device requests transfers, they will be granted.