CS2600 - Computer Organization – A brief summary

Syllabus:

- Introduction: Function and structure of a computer, Functional components of a computer, Interconnection of components, Performance of a computer.
- Representation of Instructions: Machine instructions, Operands, Addressing modes, Instruction formats, Instruction sets, Instruction set architectures - CISC and RISC architectures.
- Processing Unit: Organization of a processor - Registers, ALU and Control unit, data path in a CPU, Instruction cycle, Organization of a control unit - Operations of a control unit, Hardwired control unit, Micro-programmed control unit.
- Memory Subsystem: Semiconductor memories, Memory cells - SRAM and DRAM cells, Internal Organization of a memory chip, Organization of a memory unit, Interleaved memories, Cache memory unit - Concept of cache memory, Mapping methods, Organization of a cache memory unit, Fetch and write mechanisms, Memory management unit - Concept of virtual memory, Address translation, Hardware support for memory management.
- Input/Output Subsystem: Access of I/O devices, I/O ports, I/O control mechanisms - program controlled I/O, Interrupt controlled I/O, and DMA controlled I/O, USB bus, Secondary storage devices.

Introduction:

Computer Organization vs Computer Architecture:

Computer architecture is the conceptual design and fundamental operational structure of a computer system. It is a functional description of requirements and design implementations for the various parts of a computer.

Computer organization (CO) is how operational attributes are linked together and contribute to realize the architectural specifications. CO encompasses all physical aspects of computer systems

Main Functions of a Computer:

- Data processing
- Data storage
- Data movement
- Control

Main Structural Blocks/Parts:

Central Processing Unit (CPU): Controls the operation of the computer and performs its data processing functions. Often simply referred to as processor. Major structural components of a CPU are:

- Control Unit (CU): Controls the operation of the CPU and hence the computer.
- Arithmetic and Logic Unit (ALU): Performs computer's data processing functions. Register: Provides storage internal to the CPU.
- CPU Interconnection: Communication among the control unit, ALU, and register.

Main Memory: Stores data. I/O: Moves data between the computer and its external environment.

System Interconnection: e.g. BUS for communication among CPU, main memory, and I/O.

Performance Analysis of a Computer:
Basic performance equation:

$$T = N*S/R$$

T - Processor time required to execute a program (not total time used)
N - Actual number of machine instructions (including that due to loops)
S - Average No. of clock cycles/instruction
R - Cycles/sec

SPEC rating (ratio) = TR / TC;
TR - Running time of the Reference Computer
TC - Running time of the Computer under test

$SPEC = [(SPEC_1)*(SPEC_2)* \ldots *(SPEC_n)]^{1/n}$
n – No. of programs in the SPEC Suite.
Higher the SPEC score, better the performance.

Instructions:

An instruction set is a list of all the instructions that a processor can execute.

Typical Categories of Instructions:
- Arithmetic - add, subtract
- Logic - and, or and not
- Data - move, input, output, load and store
- Control flow – goto, if, call and return.

CPU must:
—Fetch instructions
—Interpret instructions
—Fetch data
—Process data
—Write data

Various Addressing Modes:

Addressing Mode	Example of Instruction	Meaning	When used
Register	Add R4, R3	Regs[R4]←Regs[R4]+Regs[R3]	When a value is in a register.
Immediate	Add R4,#3	Regs[R4]←Regs[R4]+3	For constants.
Displacement	Add R4,100(R1)	Regs[R4]←Regs[R4]+Mem[100+Regs[R1]]	Accessing local variables(+ simulates register indirect, direct addressing modes)
Register indirect	Add R4,(R1)	Regs[R4]←Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or acomputed address.
Indexed	Add R3,(R1+R2)	Regs[R3]←Regs[R3]+Mem[Regs[R1]+ Regs[R2]]	Sometimes useful in array addressing:R1 = base of array R2 = index amount.
Direct or absolute	Add R1,(1001)	Regs[R1]←Regs[R1]+Mem[1001]	Sometimes useful for accessing static data; address constant may need to be large
Memory Indirect	Add R1, @(R3)	Regs[R1]←Regs[R1]+Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer *p*, then mode yields **p*
Autoincrement	Add R1, (R2)+	Regs[R1]←Regs[R1] + Mem[Regs[R2]] Regs[R2]←Regs[R2]+*d*	Useful for stepping through arrays within a loop. R2 point to start of array; each reference increments R2 by size of an element, *d*.

Autodecrement	Add R1, -(R2)	Regs[R2]←Regs[R2]–*d* Regs[R1]←Regs[R1]+ Mem[Regs[R2]]	Same use as autoincrement. Autodecrement/increment can also act as push/pop to implement a stack.
Scaled	Add R1, 100(R2)[R3]	Regs[R1]← Regs[R1]+ Mem[100+Regs[R2]+ Regs[R3]*d]	Used to index arrays. May be applied to any indexed addressing mode in some computers.

MIPS Architecture:

MIPS (Microprocessor without Interlocked Pipeline Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA).

Computer workstation systems using MIPS processors are:
SGI, MIPS Computer Systems, Inc., Whitechapel Workstations, Olivetti, Siemens-Nixdorf, Acer, Digital Equipment Corporation, NEC, and DeskStation.

Operating systems ported to the architecture include:
SGI's IRIX, Microsoft's Windows NT (until v4.0), Windows CE, Linux, BSD, UNIX System V, SINIX, QNX.

MIPS Addressing Mode:

Multiple forms of addressing are generically called addressing modes. The MIPS addressing modes are the following:

1. Register addressing, where the operand is a register
2. Base or displacement addressing, where the operand is at the memory location, whose address is the sum of a register and a constant in the instruction
3. Immediate addressing, where the operand is a constant within the instruction itself
4. PC-relative addressing, where the address is the sum of the PC and a constant in the instruction
5. Pseudodirect addressing, where the jump address is the 26 bits of the instruction concatenated with the upper bits of PC.

MIPS Instruction Set:

Category	Instruction	Example	Meaning	Comments
Arithmetic	Add	Add $s1,$s2,$s3	$s1=$s2+$s3	Three register operands
	Subtract	Sub $s1,$s2,$s3	$s1=$s2-$s3	Three register operands
	Add Intermediate	Addi $s1,$s2,100	$s1 = $s2 + 100	Used to add constants
Data transfer	Load word	Lw $s1,100($s2)	$s1=Memory[$s2+100]	Word from memory to register
	Store word	Sw $s1,100($s2)	Memory[$s2+100]=$s1	Word from register to memory
Logical	And	And $s1,$2,$s3	$s1=$s2&$s3	Three reg. operands; bit-by-bit AND
	Or	or $s1,$2,$s3	$s1=$s2|$s3	Three reg. operands; bit-by-bit OR
	Nor	nor $s1,$2,$s3	$s1=~($s2|$s3)	Three reg. operands; bit-by-bit NOR
	And intermediate	Andi $s1,$s2,100	$s1=$s2&100	Bit-by-bit AND reg with constant

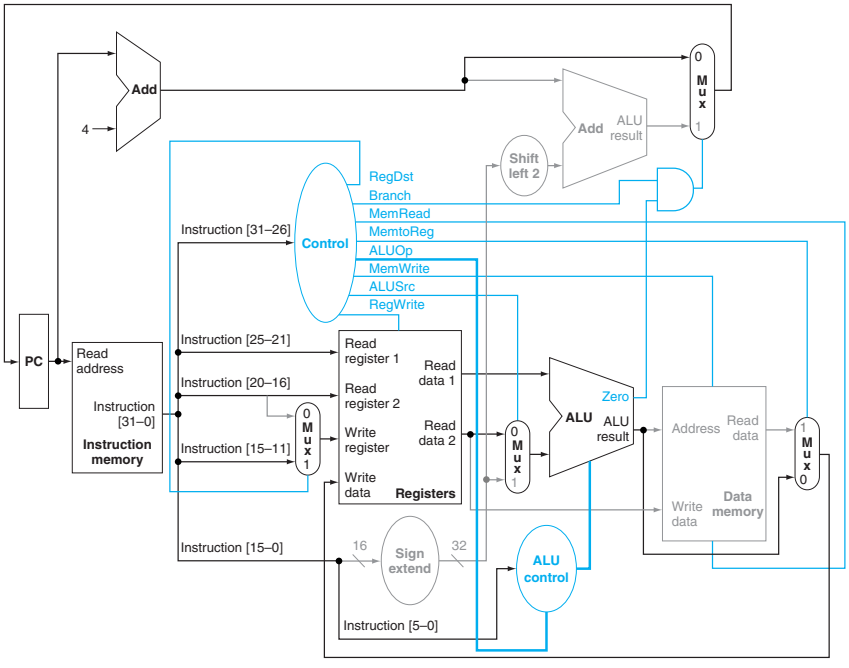
	Or intermediate	ori $s1,$2,100	$s1=$s2|100	Bit-by-bit OR reg with constant
	Shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	Shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	Branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	Branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Non equal test; PC-relative branch
	Set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	Set less than intermediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than constant
Unconditional jump	Jump	j 2500	go to 10000	Jump to target address
	Jump register	jr $ra	go to $ra	For switch, procedure return
	Jump and link	jal 2500	$ra=PC+4; go to 10000	For procedure call

RISC vs CISC Architectures:

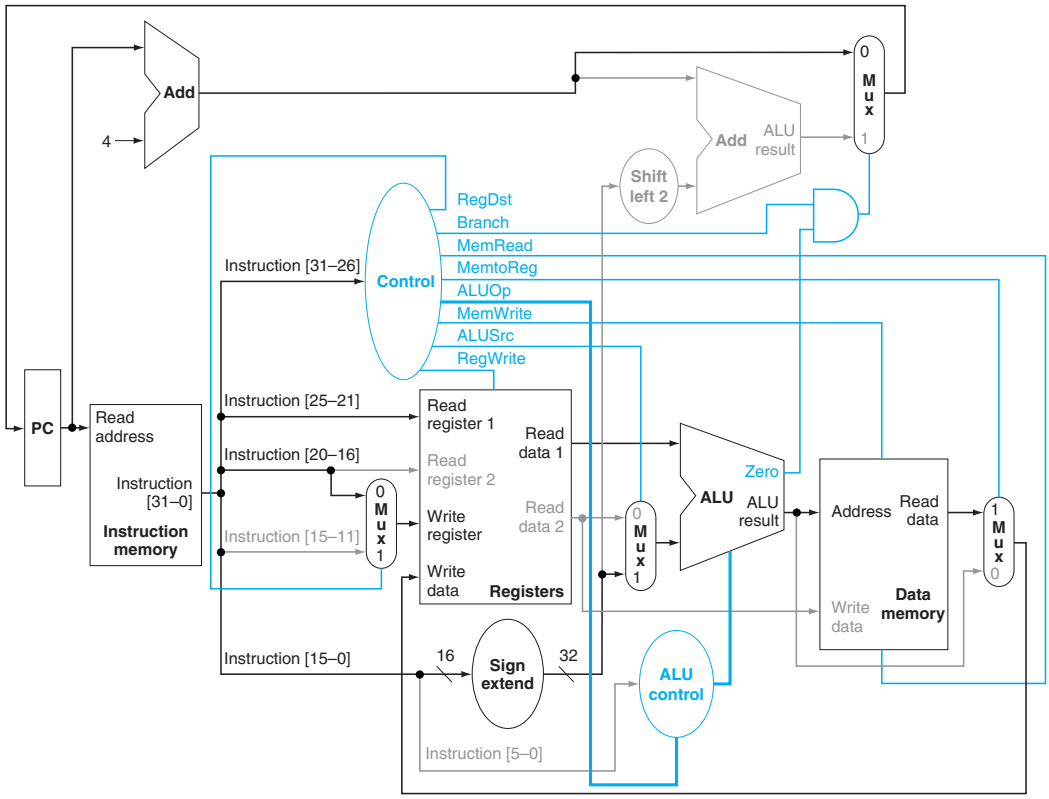
The term RISCs stands for Reduced Instruction Set Computers. It was originally introduced as a notion to mean architectures that can execute as fast as one instruction per clock cycle. The term CISCs stands for Complex Instruction Set Computers. This architecture resulted by increasing the number and complexity of instructions.

Processing Unit:

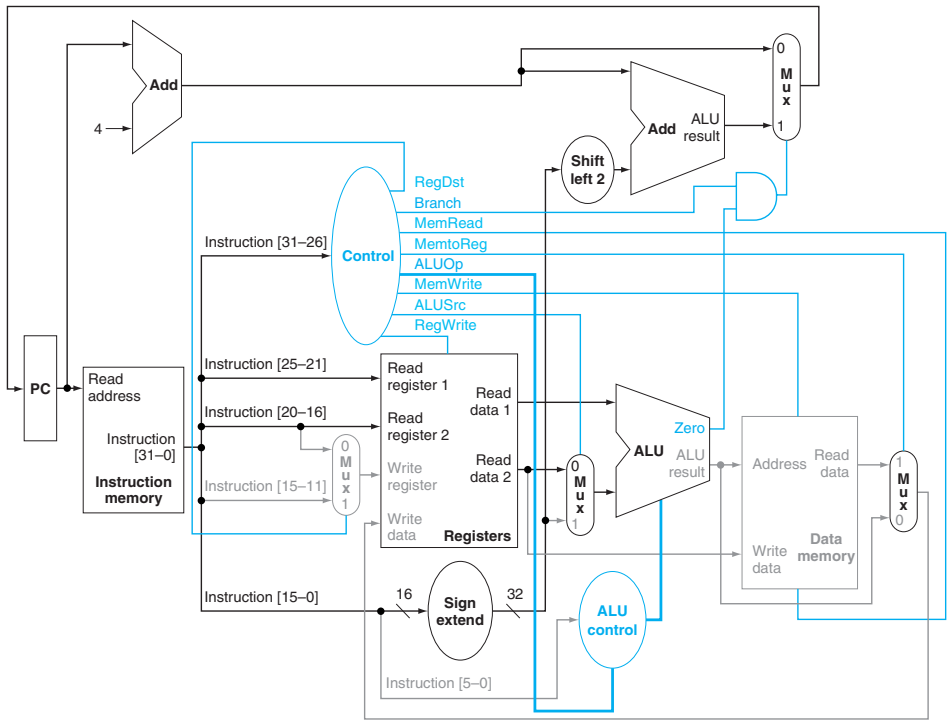
Simple datapath for an R-type instruction such as add:



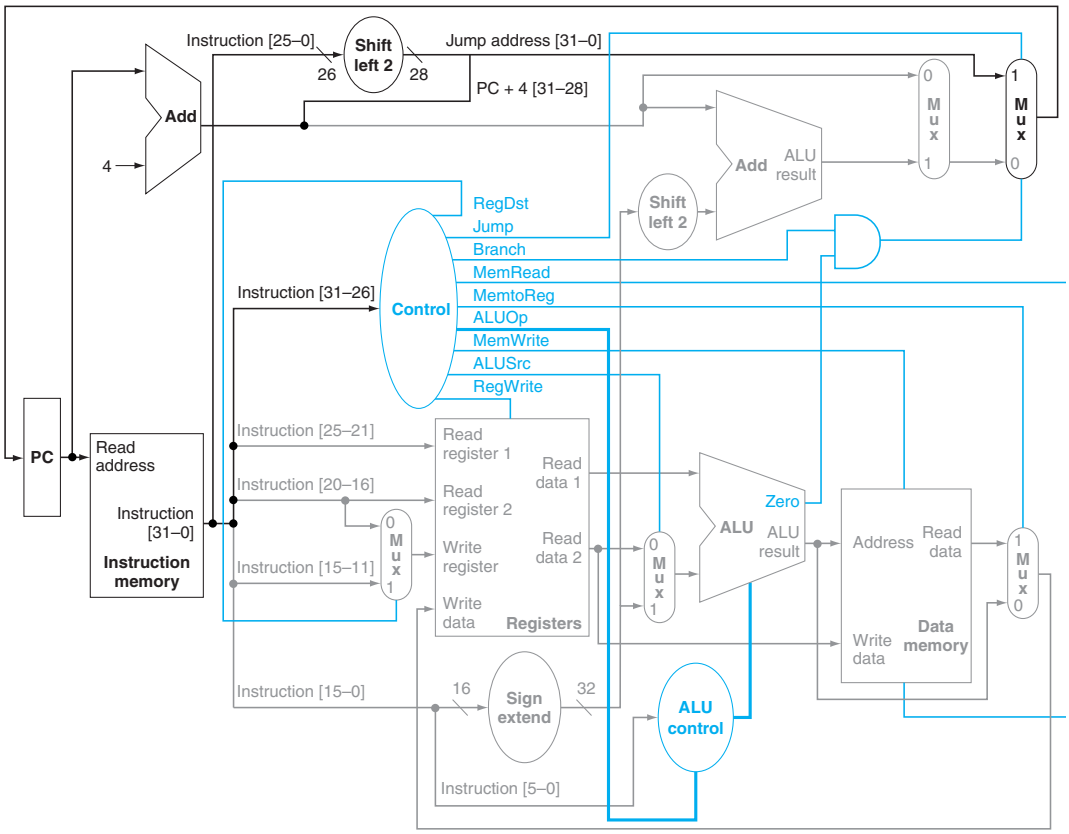
Simple datapath for a load instruction:



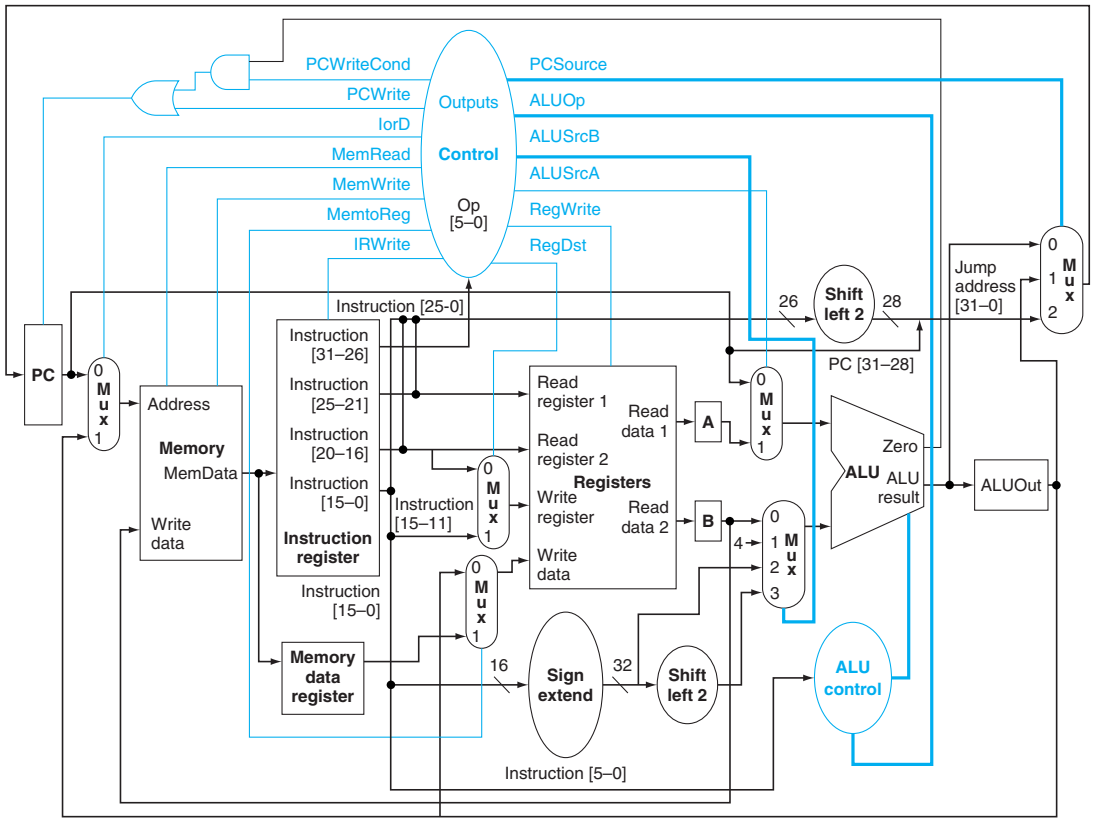
Simple datapath for a branch on equal instruction:



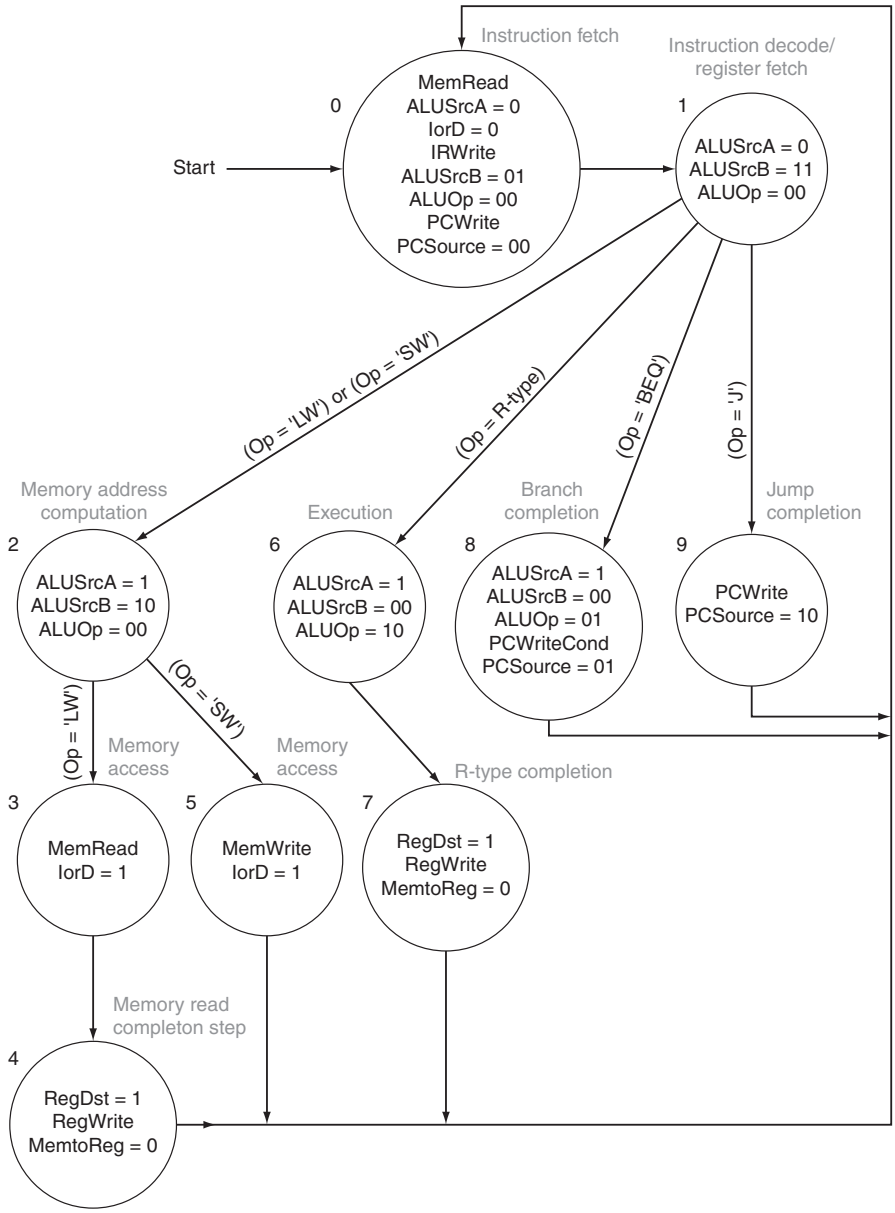
Simple datapath along with control lines for jump instruction:



Complete datapath for multicycle implementation along with the necessary control lines:



Complete finite state machine control for the multicycle datapath implementation:



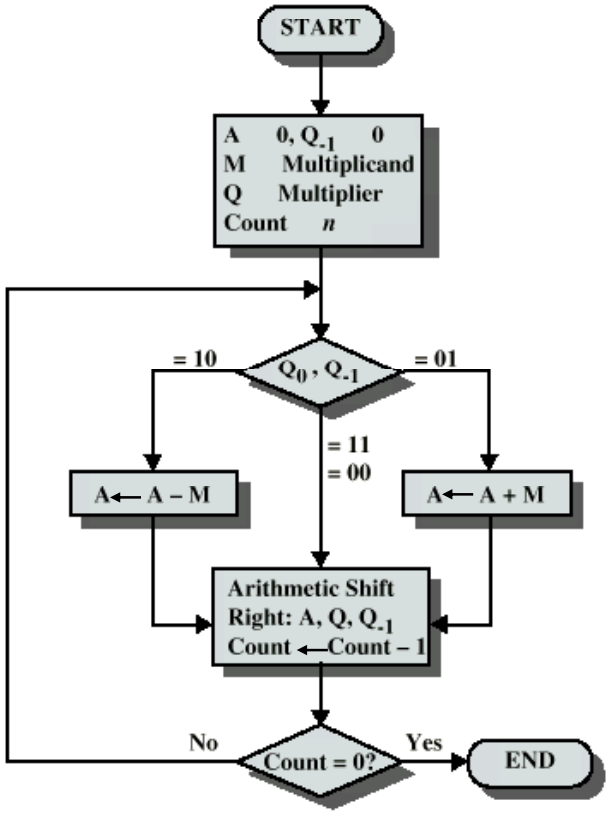
Types of Adders:

- Ripple Carry Adder
- Carry Lookahead – Full Adder
- High Level Generate & Propagate Carry Adder

Types of Multiplier Units:

- Sequential multiplier - generates partial products sequentially and adds each newly generated product to previously accumulated partial product
- Parallel multiplier - generates partial products in parallel, accumulates using a fast multi-operand adder
- Array multiplier - array of identical cells generating new partial products; accumulating them simultaneously

Flowchart for Booth's Algorithm:

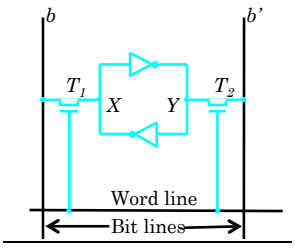


Fast Multiplication done by:
- Bit-pair recoding - Carry-save Addition of Summands - Pipelined and Booth Array Tree - etc.

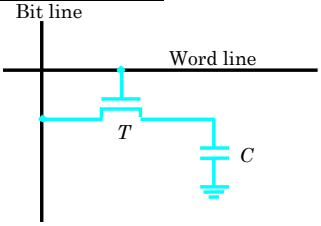
Memory Subsystem:

Various Memory Cells:

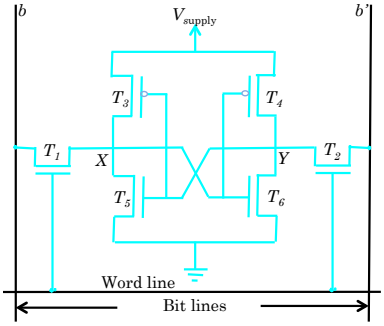
- Static RAM cell:



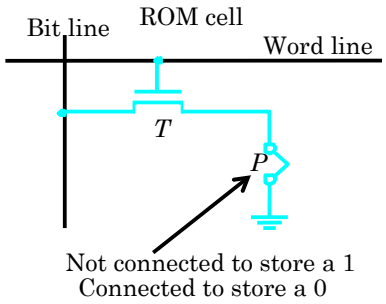
- Dynamic RAM cell:



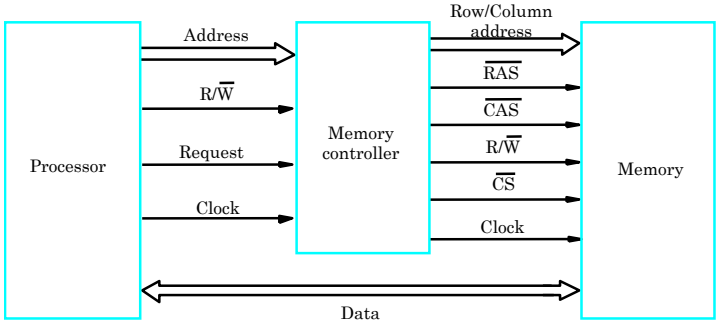
- CMOS RAM cell:



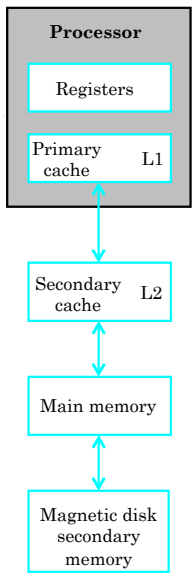
- ROM cell:



Memory Controller:



Memory Hierarchy:



Cache Memory:

Locality of reference:
Temporal locality: Accesses to the same memory location that occur close together in time.
Spatial locality: Accesses to nearby memory locations that occur close together in time.

On a read *miss*:
- Read-through (or *early start*)-Reading a word from main memory to CPU
- Read-no-through- Reading a block from main memory to cache and then from cache to CPU.

On a write *miss*:
- Write-allocate-The block is loaded on a cache miss, followed by a write *hit* action.
- Write-no-allocate-The block is modified in the main memory and not loaded into the cache.

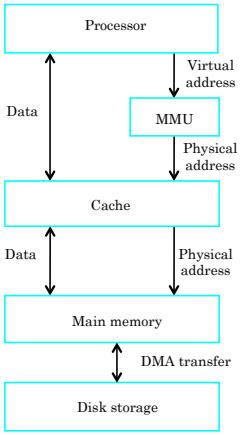
On a write *hit*:
- Write-through-Propagates each write through the cache and onto the main memory. Easy to implement. Main memory always has the most up to date copy of the data. Write is slower. Requires more main memory bandwidth.
- Write-back-Data is written only to the cache block. Delays updating the copies of the block. Updated block is marked with a *dirty* bit. Requires less write bandwidth.

Replacement Algorithms:
- First-In-First-Out (FIFO) - May not match temporal locality
- Recency-based - Least recently used (LRU)
- Frequency-based - Least frequently used (LFU)

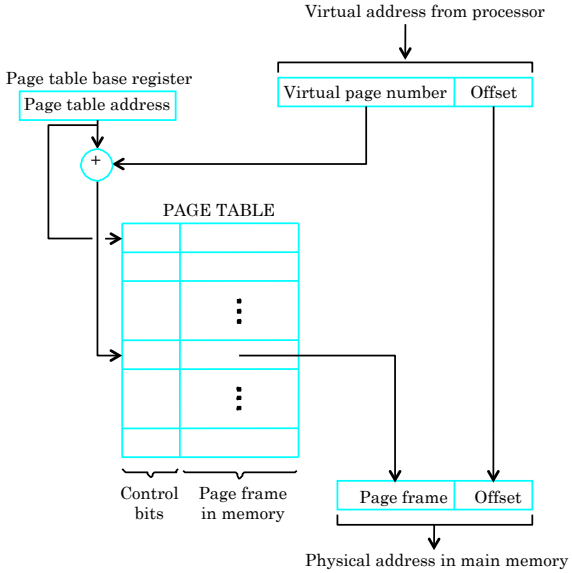
Performance Considerations:
- Interleaving
- On-chip caches
  - Hit rate (h) and miss penalty (M)
  - avg. time = h*C + (1 – h)*M, where C is the cache access time and M is the main memory access time
- Two-level caches for high performance
  - avg. time = $h_1*C_1 + (1-h_1)*h_2*C_2 + (1-h_1)*(1-h_2)*M$, where C_1 is the cache1 access time and C_2 is the cache2 access time

Virtual Memory:
- Main memory acts as a "cache" for the secondary storage.
- Motivations for virtual memory:
  - To allow efficient and safe sharing of memory among multiple programs.
  - To allow a single user program to exceed the size of primary memory.

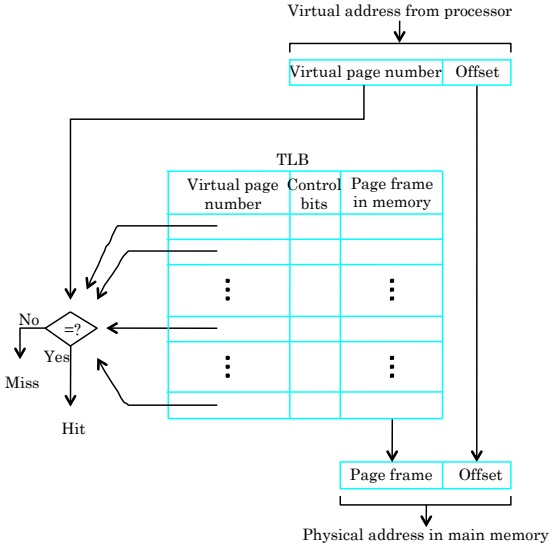
Virtual Memory Organization:



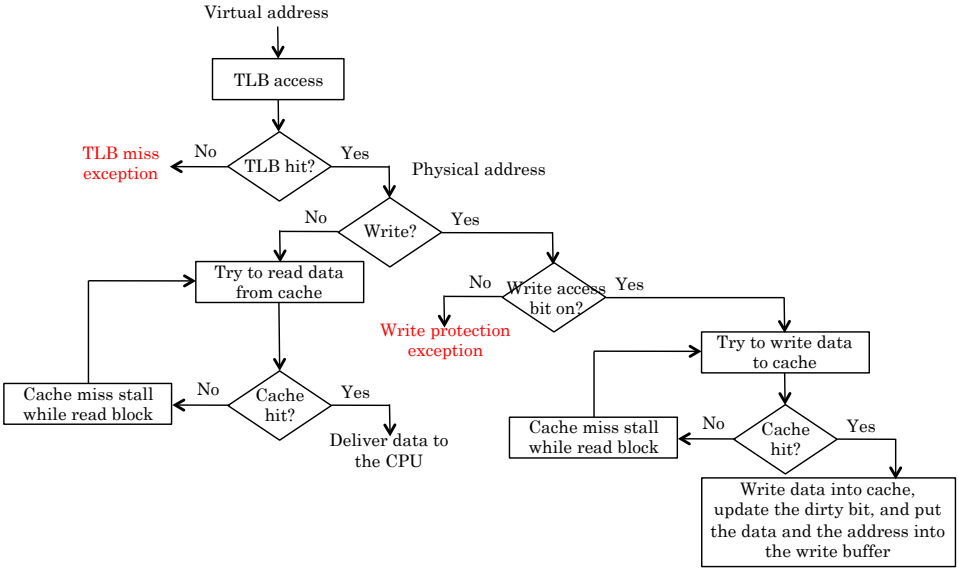
Virtual Memory Address Translation:



Translation Lookaside Buffer (TLB):



Flowchart - Processing a Read or a Write Request:



I/O Subsystem:

I/O Mechanisms:

- Memory-mapped I/O:

  When I/O devices and the memory share the same address space, the arrangement is called *Memory-mapped I/O.* In Memory-mapped I/O portions of address space are assigned to I/O devices and reads and writes to those addresses are interpreted as commands to the I/O device.

- Programmed I/O:
  - CPU has direct control over I/O, sensing status, read/write commands, transferring data.
  - CPU waits for I/O module to complete operation.
  - Wastes CPU time.

- Interrupts:

  Overhead in a polling interface lead to the invention of interrupts to notify the processor when an I/O device requires attention from the processor.
  - When I/O Device is ready, it sends the INTERRUPT signal to processor via a dedicated controller line.
  - Using interrupt we are ideally eliminating WAIT period.
  - In response to the interrupt, the processor executes the Interrupt Service Routine (ISR).
  - All the registers, flags, program counter values are saved by the processor before running ISR.
  - The time required to save status & restore contribute to execution overhead is called "Interrupt Latency".
  - Three methods of controlling interrupts:
    - Ignoring interrupts
    - Disabling interrupts
    - Special Interrupt request line
  - Techniques to handle multiple devices:
    - Polling
    - Vectored Interrupts
    - Interrupt Nesting
    - Daisy chaining

- Direct Memory Access:

  DMA is the transfer of large blocks of data at high speed between the main memory and I/O device without continuous intervention by the processor.
  - DMA controller acts as a processor, but it is controlled by CPU.
  - To initiate transfer of a block of words, the processor sends the following data to controller.
  - The starting address of the memory block.
  - The word count Control to specify the mode of transfer such as read or write a control to start the DMA transfer.
  - DMA controller performs the requested I/O operation and sends a interrupt to the processor upon completion.

Buses:

A bus is a shared communication link, which uses one set of wires to connect multiple subsystems. The two major advantages of the bus organization are versatility and low cost.
A bus generally contains a set of control lines and a set of data lines.

- The control lines are us
- ed to signal requests and acknowledgments, and to indicate what type of information is on the data lines. The control lines are used to indicate what the bus contains and to implement the bus protocol.
- The data lines of the bus carry information between the source and the destination. This information may consist of data, complex commands, or addresses.

The major disadvantage of a bus is that it creates a communication bottleneck, possibly limiting the maximum I/O throughput.
The two basic schemes for communication on the bus are synchronous and asynchronous.

- If a bus is synchronous (e.g. Processor-memory), it includes a clock in the control lines and a fixed protocol for communicating that is relative to the clock. This type of protocol can be implemented easily in a small finite state machine.
- An asynchronous bus is not clocked. It can accommodate a wide variety of devices, and the bus can be lengthened without worrying about clock skew or synchronization problems. To coordinate the transmission of data between sender and receiver, an asynchronous bus uses a handshaking protocol.

Bus Arbitration:
Bus master - device that initiates data transfers on the bus. The next device can take control of the bus after the current master relinquishes control.

Bus arbitration is needed to resolve conflicts when two or more devices want to become the bus master at the same time. In short, arbitration is the process of selecting the next bus master from among multiple candidates. Conflicts can be resolved based on fairness or priority in a centralized or distributed mechanisms.

- Centralized Arbitration - In centralized arbitration schemes, a single arbiter is used to select the next master. A simple form of centralized arbitration uses a bus request line, a bus grant line, and a bus busy line. Each of these lines is shared by potential masters, which are daisy-chained in a cascade.
- Distributed Arbitration - All devices waiting to use the bus has to carry out the arbitration process- no central arbiter. For example, a conflict can always be resolved in favor of the device with the highest arbitration number.

Universal Bus (USB):

The USB has been designed to meet several key objectives:

- Provide a simple, low-cost, and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer.
- Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections.
- Enhance user convenience through a ''plug-and-play'' mode of operation.

The USB supports two speeds of operation, called low- speed (1.5 megabits/s) and full-speed (12 megabits/s). The most recent revision of the bus specification (USB 2.0) introduced a third speed of operation, called high-speed (480 megabits/s).

USB Architecture: