B.E-CSE **2018-2019**

15BECS402 **COMPUTER ARCHITECTURE** **3H-5C**

Instruction Hours/week: L: 3 T:0 P:4 **Marks:** Internal**:40** External**:60** Total**:100**

End Semester Exam:3 Hours

COURSE OBJECTIVES:

To expose the students to the following :

- How Computer Systems work & the basic principles
- Instruction Level Architecture and Instruction Execution
- The current state of art in memory system design
- How I/O devices are accessed and its principles.
- To provide the knowledge on Instruction Level Parallelism
- To impart the knowledge on micro programming
- Concepts of advanced pipelining techniques.

COURSE OUTCOMES:

- Draw the functional block diagram of a single bus architecture of a computer and describe the function of the instruction execution cycle, RTL interpretation of instructions, addressing modes, instruction set.
- Write assembly language program for specified microprocessor for computing
  16 bit multiplication, division and I/O device interface (ADC,
  Control circuit, serial port communication).
- Write a flowchart for Concurrent access to memory and cache coherency in Parallel
  Processors and describe the process.
- Given a CPU organization and instruction, design a memory module and analyze its
  operation by interfacing with the CPU.
- Given a CPU organization, assess its performance, and apply design techniques to
  enhance performance using pipelining, parallelism and RISC methodology

UNIT 1:

Functional blocks of a computer: CPU, memory, input-output subsystems, control unit.

Instruction set architecture of a CPU – registers, instruction execution cycle, RTL interpretation of instructions, addressing modes, instruction set. Case study – instruction sets of some common CPUs.

Data representation: signed number representation, fixed and

floating point representations, character representation. Computer arithmetic – integer addition and subtraction, ripple carry adder, carry look-ahead adder, etc. multiplication – shift-andadd, Booth multiplier, carry save multiplier, etc. Division restoring and non-restoring techniques, floating point arithmetic.

UNIT 2:

Introduction to x86 architecture. **CPU control unit design**: hardwired and micro-programmed design approaches, Case study – design of a simple hypothetical CPU. **Memory system design**: semiconductor memory technologies, memory organization.

UNIT 3:

Peripheral devices and their characteristics: Input-output subsystems, I/O device interface, I/O transfers – program controlled, interrupt driven and DMA, privileged and non-privileged instructions, software interrupts and exceptions. Programs and processes – role of interrupts in process state transitions, I/O device interfaces – SCII, USB

UNIT 4:

Pipelining: Basic concepts of pipelining, throughput and speedup, pipeline hazards. **Parallel Processors**: Introduction to parallel processors, Concurrent access to memory and cache coherency.

UNIT 5:

Memory organization: Memory interleaving, concept of hierarchical memory organization, cache memory, cache size vs. block size, mapping functions,
replacement algorithms, write policies.

TEXT BOOKS:

1. "Computer Organization and Design: The Hardware/Software Interface",
   5th Edition by David A. Patterson and John L. Hennessy,
   Elsevier.
2. "Computer Organization and Embedded Systems", 6th Edition by CarlHamacher, McGraw Hill Higher Education.

REFERENCES:

1. "Computer Architecture and Organization", 3rd Edition by John P. Hayes, WCB/McGraw-Hill
2. "Computer Organization and Architecture: Designing for Performance", 10th Edition by William Stallings, Pearson Education.
3. "Computer System Design and Architecture", 2nd Edition by Vincent P. Heuring and Harry F. Jordan, Pearson Education.

KARPAGAM ACADEMY OF HIGHER EDUCATION

Faculty of Engineering

Department of Computer Science and Engineering

Lecture Plan

Subject Name: COMPUTER ARCHITECTURE **Subject Code: 15BECS402**

S.No	Topic Name	No.of Periods	Supporting Materials	Teaching Aids
	UNIT- I Functional blocks of a computer			
1	**Functional blocks of a computer**	1	Web	BB
2	CPU, memory, input-output subsystems, control unit	1	T[1]3	BB
3	Instruction set architecture of a CPU	1	T[1]5	PPT
4	registers, instruction execution cycle	1	T[1]5	PPT
5	RTL interpretation of instructions, addressing modes, instruction set. Case study – instruction sets of some common CPUs.	1	T[1]3	PPT
6	**Data representation**: signed number representation, fixed and floating point representations, character representation	1	T[1]56	PPT
7	Computer arithmetic – integer addition and subtraction, ripple carry adder, carry look-ahead adder	1	T[1]95	PPT
8	Multiplication – shift-andadd, Booth multiplier, carry save multiplier	1	T[1]68	BB
9	Division restoring and non-restoring techniques	1	Web	PPT
10	Floating point arithmetic	1	T[1]12	BB
11	Tutorial : Data Representation	1	Web	BB
	Total	**11**		
	UNIT- II INTRODUCTION TO X86 ARCHITECTURE			
12	**Introduction** to x86 architecture	1	T[1]200	PPT
13	**CPU control unit design**	1	web	PPT
14	Hardwired and micro-programmed design approaches	1	R[1] 201	BB
15	micro-programmed design approaches	1	web	PPT
16	Case study – design of a simple hypothetical CPU	1	R[1]214	PPT
17	**Memory system design**	1	R[1]218	PPT
18	semiconductor memory technologies	1	R[1]218	PPT
19	Memory organization.	1	T[1]218	PPT
20	Memory organization.-Example	1	T[1]221	BB

21	Tutorial : X86 Architecture	1	R[1]221	PPT
Total		**10**		
	UNIT- III PERIPHERAL DEVICES AND THEIR CHARACTERISTICS			
22	Input-output subsystems	1	web	PPT
23	I/O device interface, I/O transfers	1	web	PPT
24	Program controlled, interrupt driven	1	web	PPT
25	DMA	1	T[1]-488	BB
26	privileged and non-privileged instructions	1	T[1]-193	PPT
27	software interrupts and exceptions	1	T[1]-266	BB
28	Programs and processes	1	T[1]-305	PPT
29	role of interrupts in process state transitions	1	T[1]-343	BB
30	I/O device interfaces, SCII, USB	1	web	PPT
31	Tutorial: Implementation	1	web	PPT
Total		**10**		
	UNIT- IV PIPELINING			
32	Basic concepts of pipelining	1	T[1]139	PPT
33	Throughput in pipelining	1	T[1]139	PPT
34	Speedup in pipelining	1	T[1]-140	PPT
35	Pipeline hazards	1	T[1]152	BB
36	**Parallel Processors**	1	T[1]159	PPT
37	Introduction to parallel processors	1	T[1]162	BB
38	Concurrent access to memory	1	T[1]163	PPT
39	Cache coherency	1	T[1]133	PPT
40	Revision	1	web	PPT
41	Tutorial : Implementation of pipelining	1	T[1]133	BB
Total		**10**		
	UNIT- V MEMORY ORGANIZATION			
42	Memory organization	1	T[1]248	PPT
43	Memory interleaving	1	T[1]465	BB
44	concept of hierarchical memory organization	1	T[1]465	BB
45	cache memory	1	T[1]255	PPT
46	cache size vs. block size	1	T[1]248	PPT
47	mapping functions	1	T[1]-1087	PPT
48	write policies	1	T[1]-1087	PPT
49	Tutorial-Illustration	1	T[1]-690	BB

50	Tutorial: Cache Memory	1	T[1]-690	PPT
51	Revisions	1	T[1]-752	BB
52	**Discussion on Previous University Question Papers**			
Total		**10**		
	Total Hours	**52**		

TEXT BOOKS

S.NO	Title of the book			Year of publication
1	Computer Organization and Design: The Hardware/Software Interface", 5th Edition by David A. Patterson and John L. Hennessy, Elsevier.			2012
2	Computer Organization and Embedded Systems", 6th Edition by CarlHamacher, McGraw Hill Higher Education			2013

REFERNCE BOOKS

S.NO	Title of the book			Year of publication
1	"Computer Architecture and Organization", 3rd Edition by John P. Hayes, WCB/McGraw-Hill			2011
2	"Computer Organization and Architecture: Designing for Performance" , 10th Edition by William Stallings, Pearson Education.			2012
3	"Computer System Design and Architecture", 2nd Edition by Vincent P. Heuring and Harry F. Jordan, Pearson Education.			2011

WEBSITES

1. https://www.javatpoint.com/COA-tutorial
2. https://nptel.ac.in/content/syllabus_pdf/106106132.pdf

Chapter – 1
Introduction

1.1 Computer Organization and Architecture

- Computer Architecture refers to those attributes of a system that have a direct impact on the logical execution of a program. Examples:
  - the instruction set
  - the number of bits used to represent various data types
  - I/O mechanisms
  - memory addressing techniques
- Computer Organization refers to the operational units and their interconnections that realize the architectural specifications. Examples are things that are transparent to the programmer:
  - control signals
  - interfaces between computer and peripherals
  - the memory technology being used
- So, for example, the fact that a multiply instruction is available is a computer architecture issue. How that multiply is implemented is a computer organization issue.

- Architecture is those attributes visible to the programmer
  - Instruction set, number of bits used for data representation, I/O mechanisms, addressing techniques.
  - e.g. Is there a multiply instruction?
- Organization is how features are implemented
  - Control signals, interfaces, memory technology.
  - e.g. Is there a hardware multiply unit or is it done by repeated addition?
- All Intel x86 family share the same basic architecture
- The IBM System/370 family share the same basic architecture
- This gives code compatibility
  - At least backwards
- Organization differs between different versions

1.2 Structure and Function

- Structure is the way in which components relate to each other
- Function is the operation of individual components as part of the structure
- All computer functions are:
  - **Data processing**: Computer must be able to process data which may take a wide variety of forms and the range of processing.
  - **Data storage**: Computer stores data either temporarily or permanently.
  - **Data movement**: Computer must be able to move data between itself and the outside world.
  - **Control**: There must be a control of the above three functions.

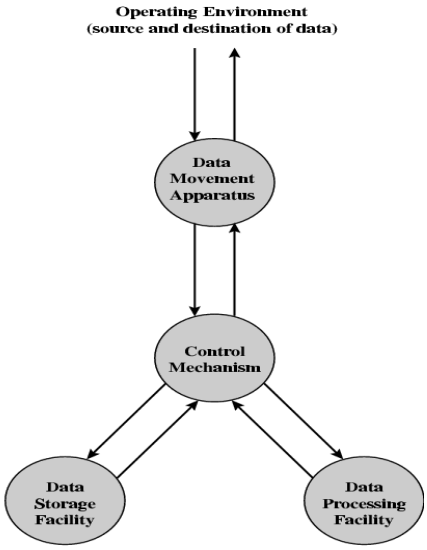


Fig: Functional view of a computer

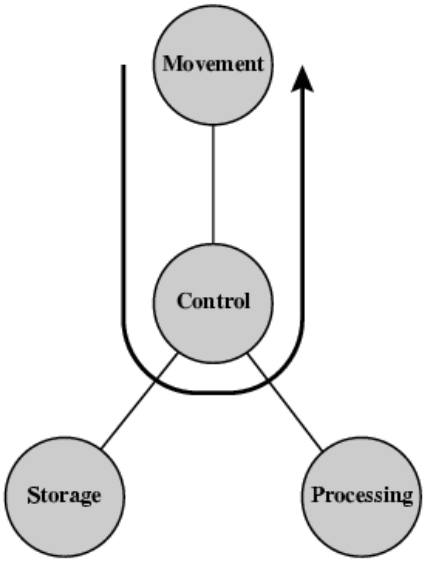


Fig: Data movement operation

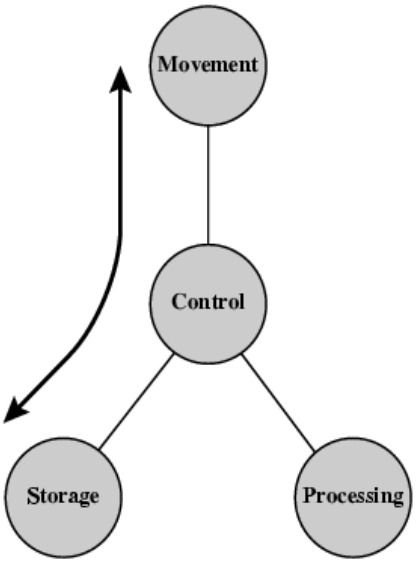


Fig: Storage Operation

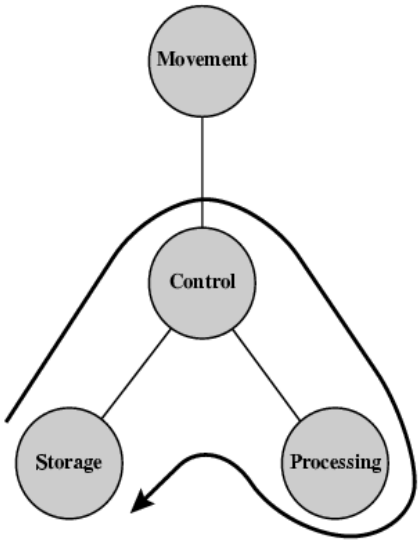


Fig: Processing from / to storage

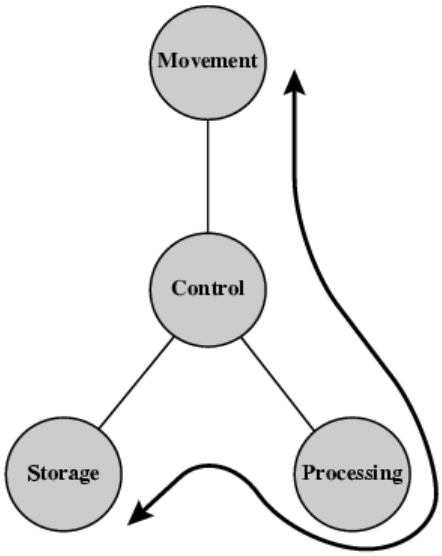


Fig: Processing from storage to i/o

- Four main structural components:
  - Central processing unit (CPU)
  - Main memory
  - I / O
  - System interconnections
- CPU structural components:
  - Control unit
  - Arithmetic and logic unit (ALU)
  - Registers
  - CPU interconnections

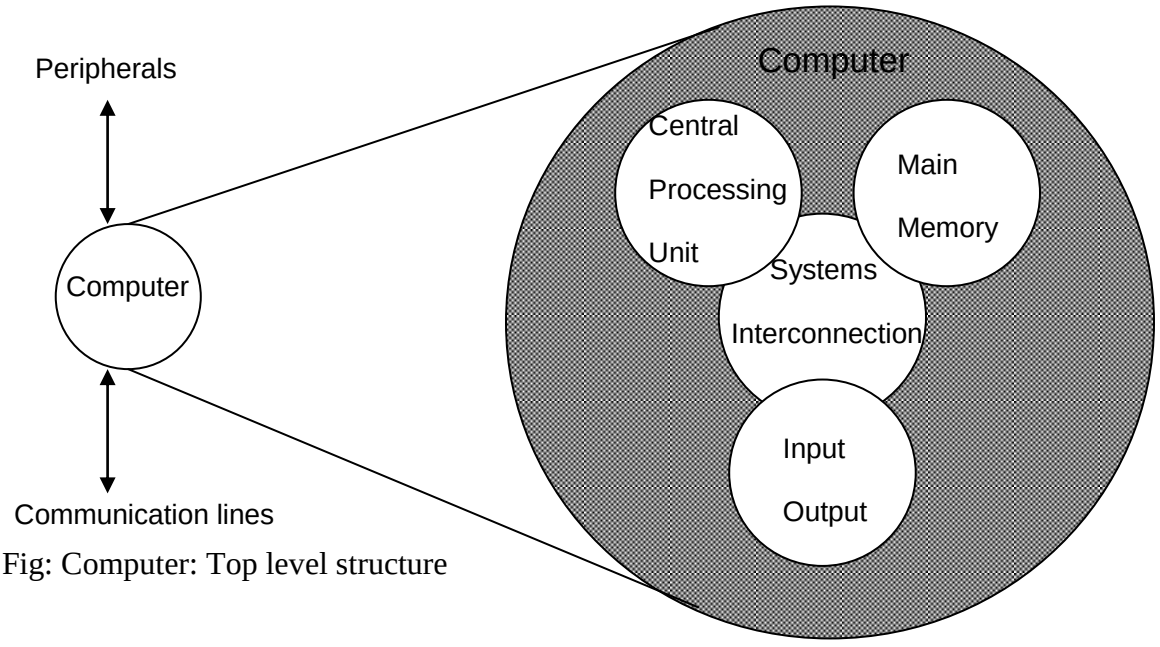


Fig: Computer: Top level structure

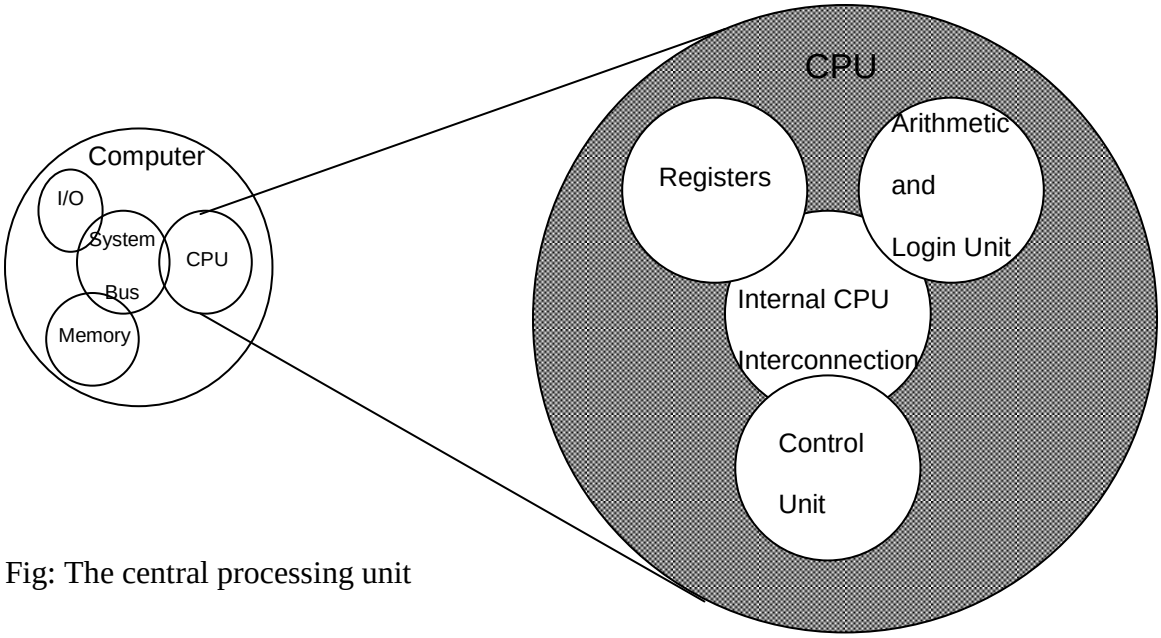


Fig: The central processing unit

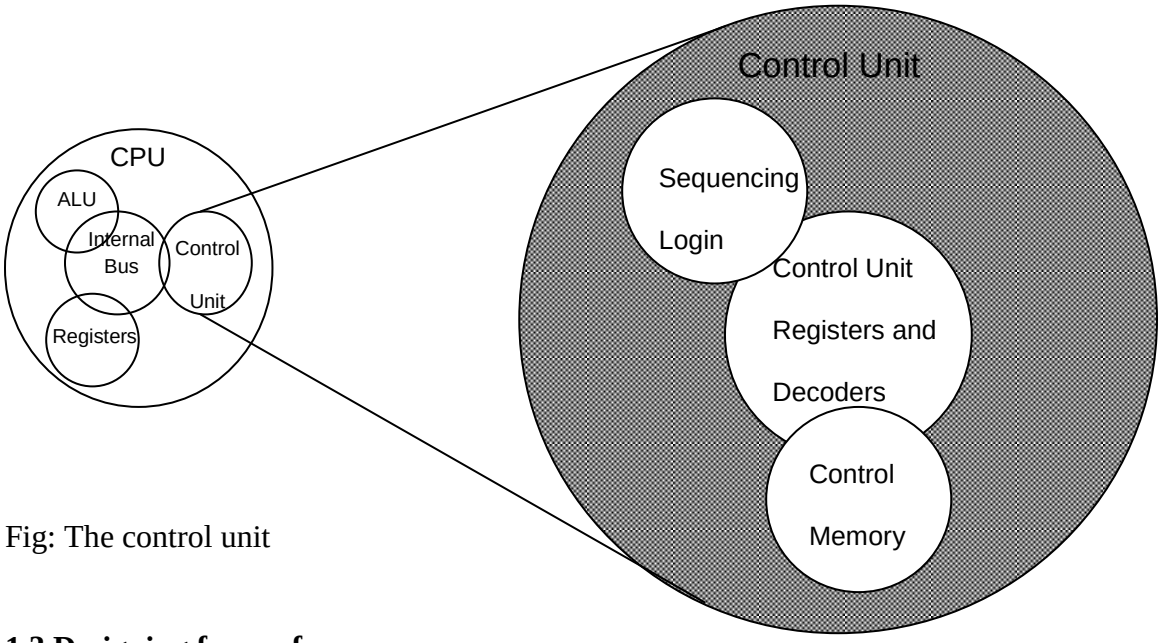


Fig: The control unit

1.3 Designing for performance

Some of the driving factors behind the need to design for performance:

- Microprocessor Speed
  - Pipelining
  - On board cache, on board L1 & L2 cache
  - Branch prediction: The processor looks ahead in the instruction code fetched from memory and predicts which branches, or group of instructions are likely to be processed next.
  - Data flow analysis: The processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions to prevent delay.

- Speculative execution: Using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, holding the results in temporary locations.

- Performance Mismatch
  - Processor speed increased
  - Memory capacity increased
  - Memory speed lags behind processor speed

Below figure depicts the history; while processor speed and memory capacity have grown rapidly, the speed with which data can be transferred between main memory and the processor has lagged badly.

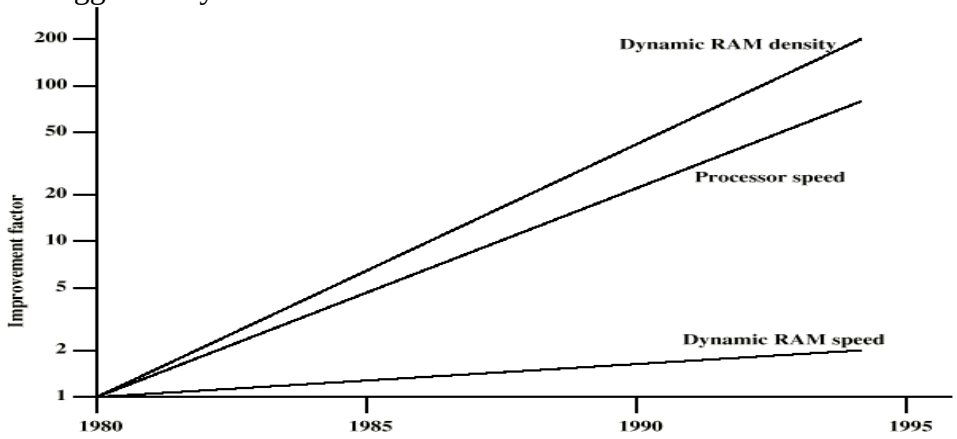


Fig: Evolution of DRAM and processor Characteristics

The effects of these trends are shown vividly in figure below. The amount of main memory needed is going up, but DRAM density is going up faster (number of DRAM per system is going down).

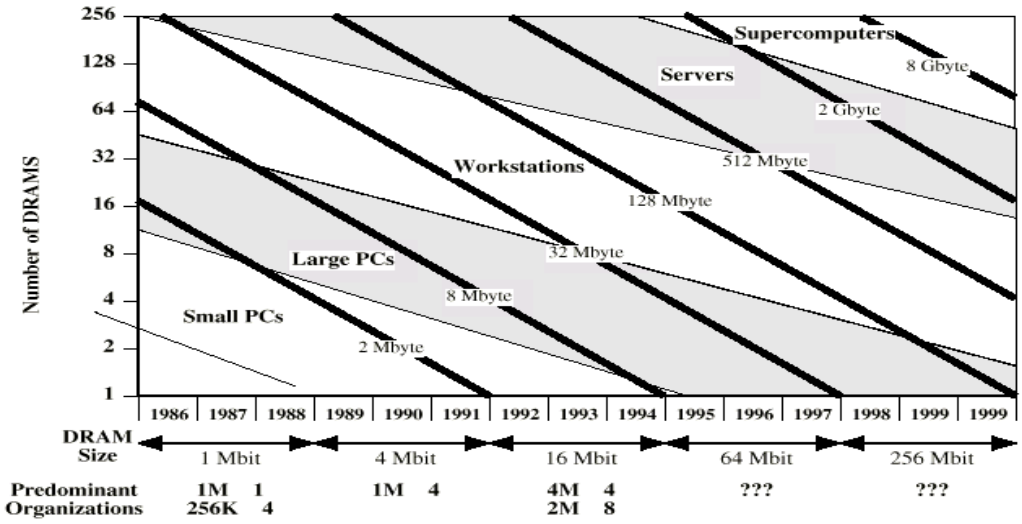


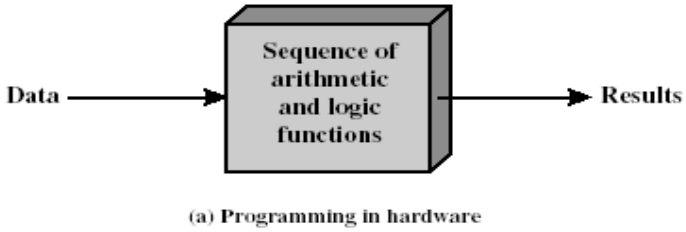
Fig: Trends in DRAM use

Solutions

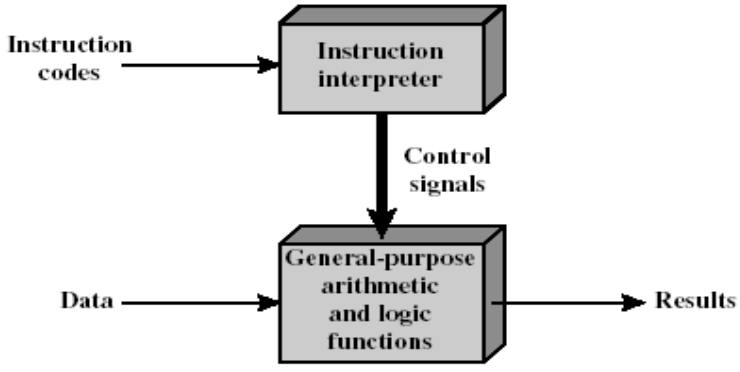
- Increase number of bits retrieved at one time
  - Make DRAM "wider" rather than "deeper" to use wide bus data paths.
- Change DRAM interface
  - Cache
- Reduce frequency of memory access
  - More complex cache and cache on chip
- Increase interconnection bandwidth
  - High speed buses
  - Hierarchy of buses

1.4 Computer Components

- The Control Unit (CU) and the Arithmetic and Logic Unit (ALU) constitute the Central Processing Unit (CPU)
- Data and instructions need to get into the system and results need to get out
  - Input/output (I/O module)
- Temporary storage of code and results is needed
  - Main memory (RAM)
- Program Concept
  - Hardwired systems are inflexible
  - General purpose hardware can do different tasks, given correct control signals
  - Instead of re-wiring, supply a new set of control signals

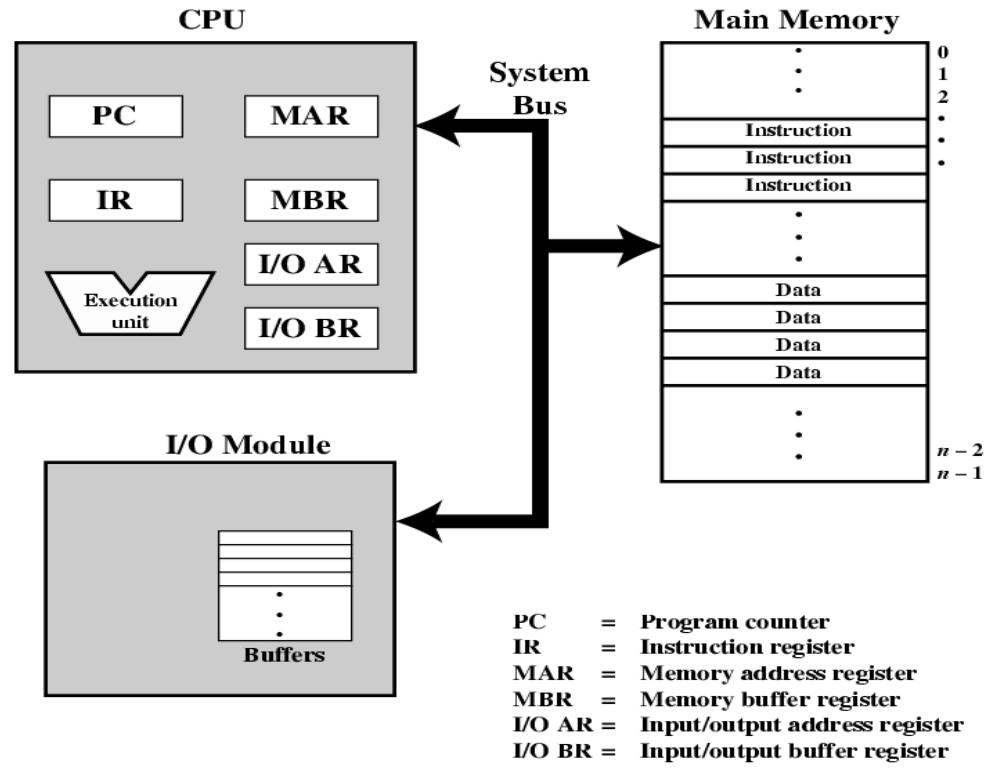


(a) Programming in hardware



(b) Programming in software

Fig: Hardware and Software Approaches



1.5 Computer Function

The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory.

- Two steps of Instructions Cycle:
  - Fetch
  - Execute

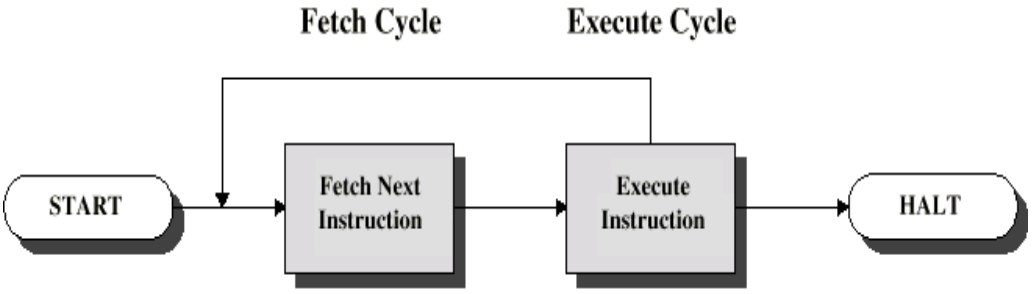


Fig: Basic Instruction Cycle

- Fetch Cycle
  - Program Counter (PC) holds address of next instruction to fetch
  - Processor fetches instruction from memory location pointed to by PC
  - Increment PC
    - Unless told otherwise
  - Instruction loaded into Instruction Register (IR)

- Execute Cycle
  - Processor interprets instruction and performs required actions, such as:
    - Processor - memory
      - data transfer between CPU and main memory
    - Processor - I/O
      - Data transfer between CPU and I/O module
    - Data processing
      - Some arithmetic or logical operation on data
    - Control
      - Alteration of sequence of operations
      - e.g. jump
    - Combination of above

Example of program execution.

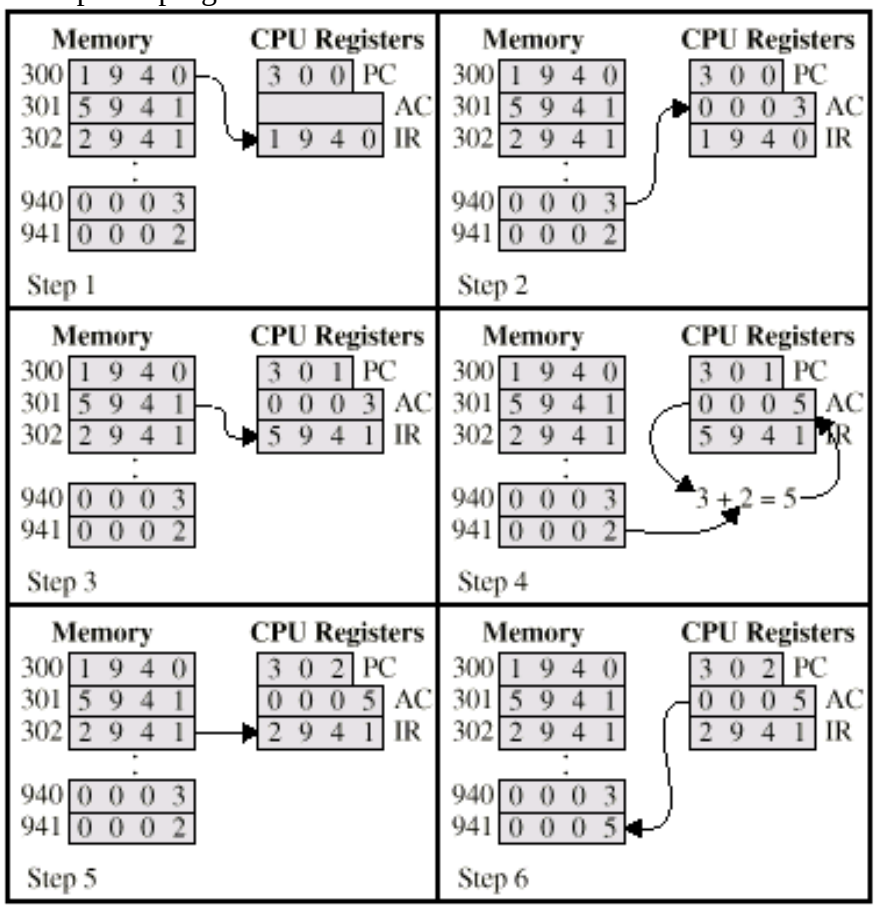


Fig: Example of program execution (consists of memory and registers in hexadecimal)

- The PC contains 300, the address of the first instruction. The instruction (the value 1940 in hex) is loaded into IR and PC is incremented. This process involves the use of MAR and MBR.
- The first hexadecimal digit in IR indicates that the AC is to be loaded. The remaining three hexadecimal digits specify the address (940) from which data are to be loaded.
- The next instruction (5941) is fetched from location 301 and PC is incremented.

- The old contents of AC and the contents of location 941 are added and the result is stored in the AC.
- The next instruction (2941) is fetched from location 302 and the PC is incremented.
- The contents of the AC are stored in location 941.

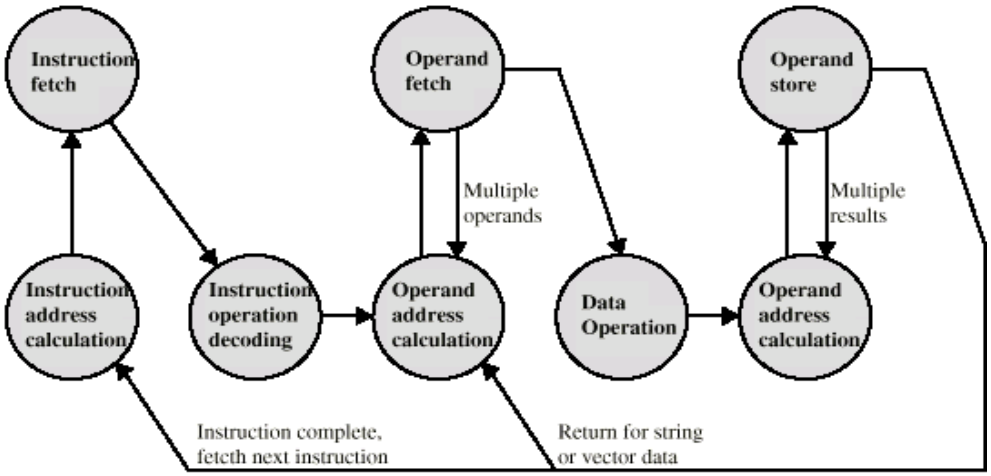
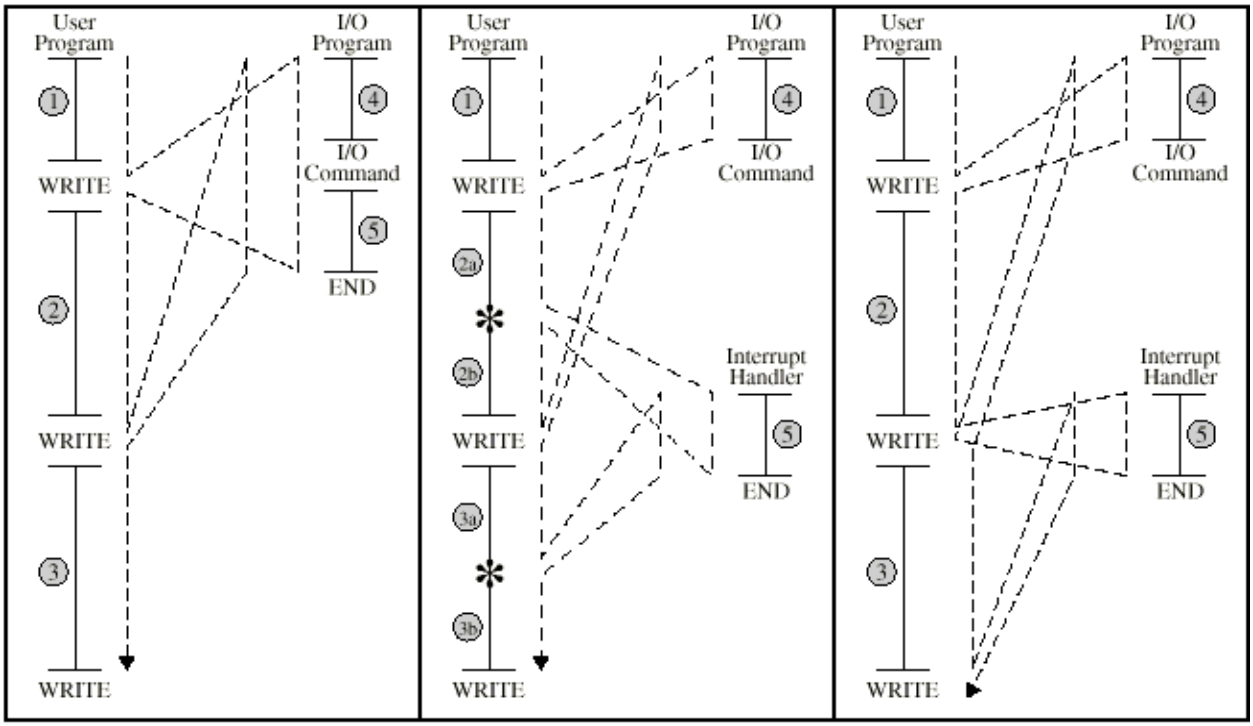


Fig: Instruction cycle state diagram

Interrupts:

- Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Program
  - e.g. overflow, division by zero
- Timer
  - Generated by internal processor timer
  - Used in pre-emptive multi-tasking
- I/O
  - from I/O controller
- Hardware failure
  - e.g. memory parity error



- Indicated by an interrupt signal
- If no interrupt, fetch next instruction
- If interrupt pending:
  - Suspend execution of current program
  - Save context
  - Set PC to start address of interrupt handler routine
  - Process interrupt
  - Restore context and continue interrupted program

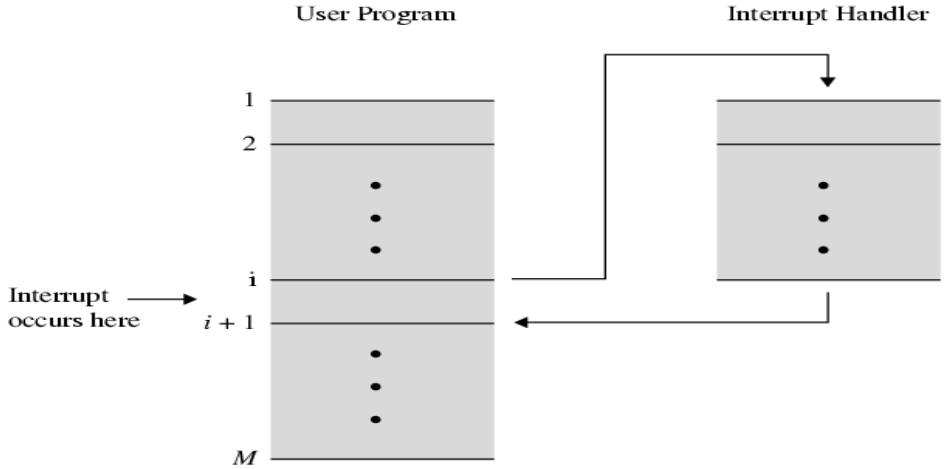


Fig: Transfer of control via interrupts

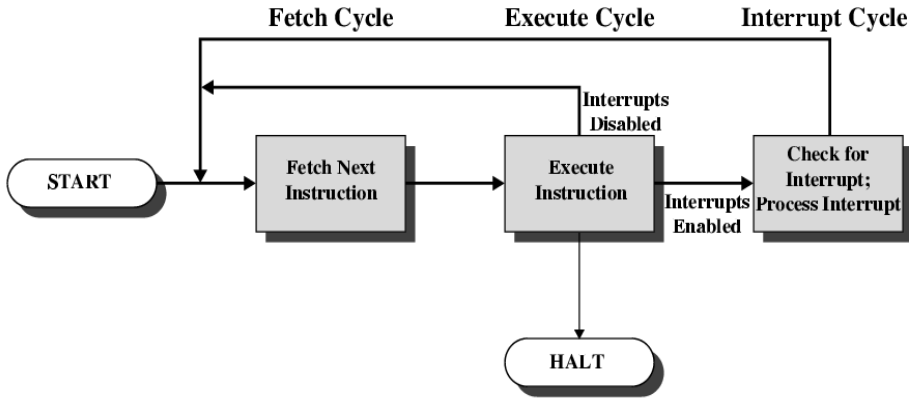


Fig: Instruction Cycle with Interrupts

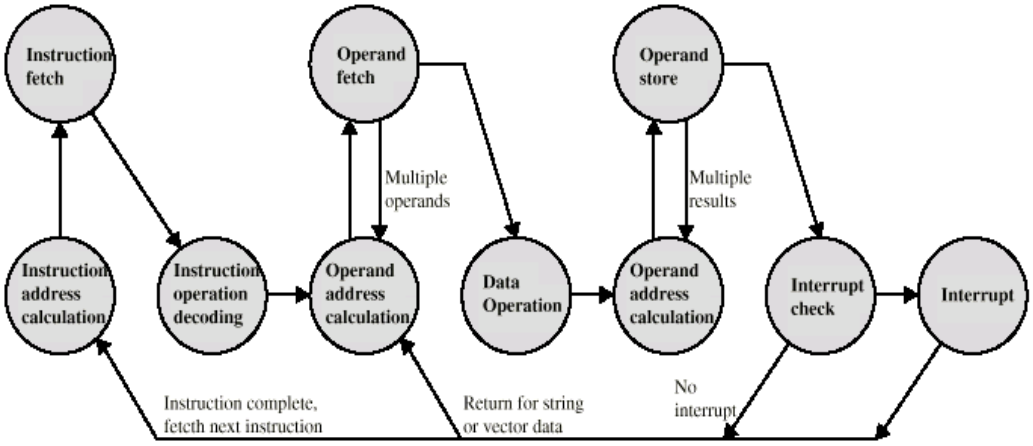


Fig: Instruction cycle state diagram, with interrupts

- Multiple Interrupts
  - Disable interrupts (approach #1)
    - Processor will ignore further interrupts whilst processing one interrupt
    - Interrupts remain pending and are checked after first interrupt has been processed
    - Interrupts handled in sequence as they occur
  - Define priorities (approach #2)
    - Low priority interrupts can be interrupted by higher priority interrupts
    - When higher priority interrupt has been processed, processor returns to previous interrupt

1.6 Interconnection structures

The collection of paths connecting the various modules is called the interconnecting structure.

- All the units must be connected
- Different type of connection for different type of unit
  - Memory
  - Input/Output
  - CPU

- Memory Connection
  - Receives and sends data
  - Receives addresses (of locations)
  - Receives control signals
    - Read
    - Write
    - Timing

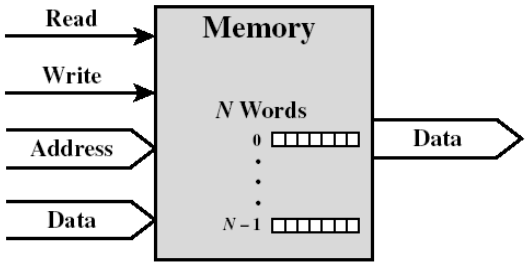


Fig: Memory Module

- I/O Connection
  - Similar to memory from computer's viewpoint
  - Output
    - Receive data from computer
    - Send data to peripheral
  - Input
    - Receive data from peripheral
    - Send data to computer
  - Receive control signals from computer
  - Send control signals to peripherals
    - e.g. spin disk
  - Receive addresses from computer
    - e.g. port number to identify peripheral
  - Send interrupt signals (control)

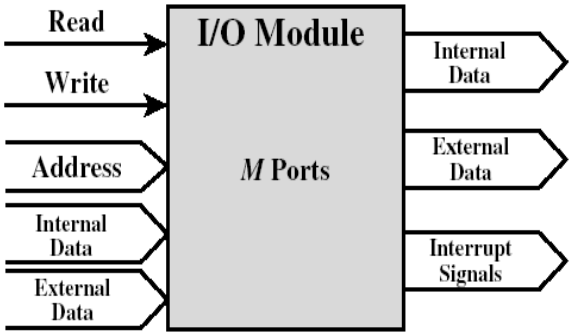


Fig: I/O Module

- CPU Connection
  - Reads instruction and data
  - Writes out data (after processing)
  - Sends control signals to other units
  - Receives (& acts on) interrupts

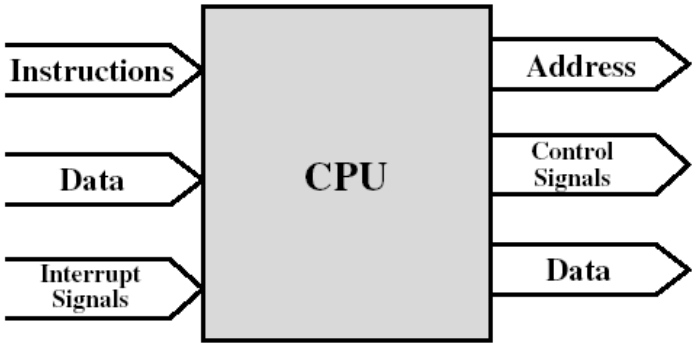


Fig: CPU Module

1.7 Bus interconnection

- A bus is a communication pathway connecting two or more devices
- Usually broadcast (all components see signal)
- Often grouped
  - A number of channels in one bus
  - e.g. 32 bit data bus is 32 separate single bit channels
- Power lines may not be shown
- There are a number of possible interconnection systems
- Single and multiple BUS structures are most common
- e.g. Control/Address/Data bus (PC)
- e.g. Unibus (DEC-PDP)
- Lots of devices on one bus leads to:
  - Propagation delays
  - Long data paths mean that co-ordination of bus use can adversely affect performance
  - If aggregate data transfer approaches bus capacity
- Most systems use multiple buses to overcome these problems

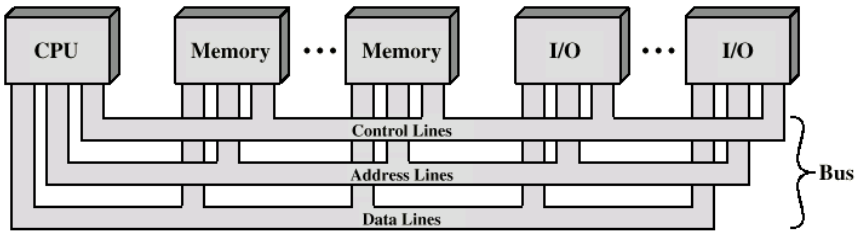


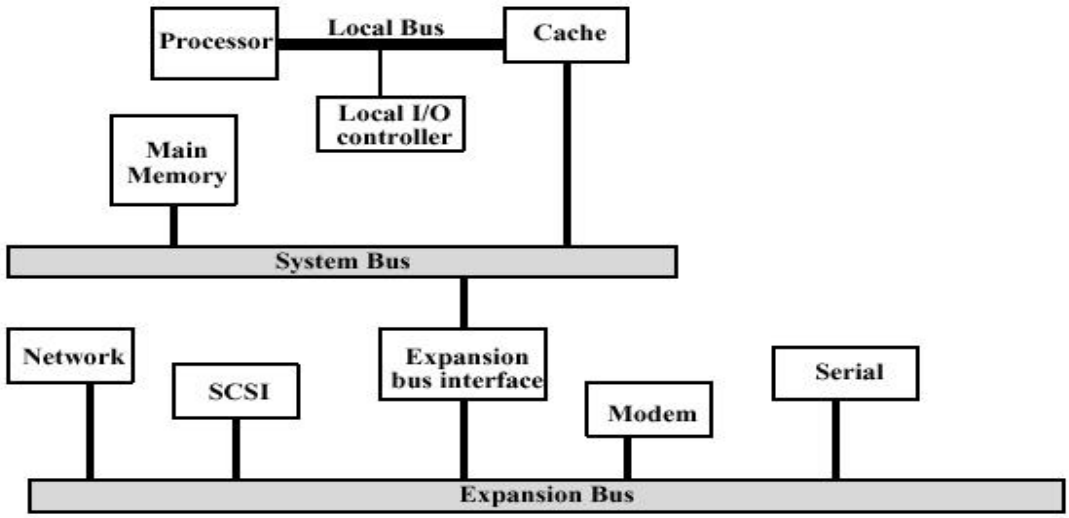
Fig: Bus Interconnection Scheme

- Data Bus
  - Carries data
    - Remember that there is no difference between "data" and "instruction" at this level
  - Width is a key determinant of performance
    - 8, 16, 32, 64 bit
- Address Bus
  - Identify the source or destination of data
  - e.g. CPU needs to read an instruction (data) from a given location in memory
  - Bus width determines maximum memory capacity of system
    - e.g. 8080 has 16 bit address bus giving 64k address space
- Control Bus
  - Control and timing information
    - Memory read
    - Memory write
    - I/O read
    - I/O write
    - Transfer ACK
    - Bus request
    - Bus grant
    - Interrupt request
    - Interrupt ACK
    - Clock
    - Reset

Multiple Bus Hierarchies

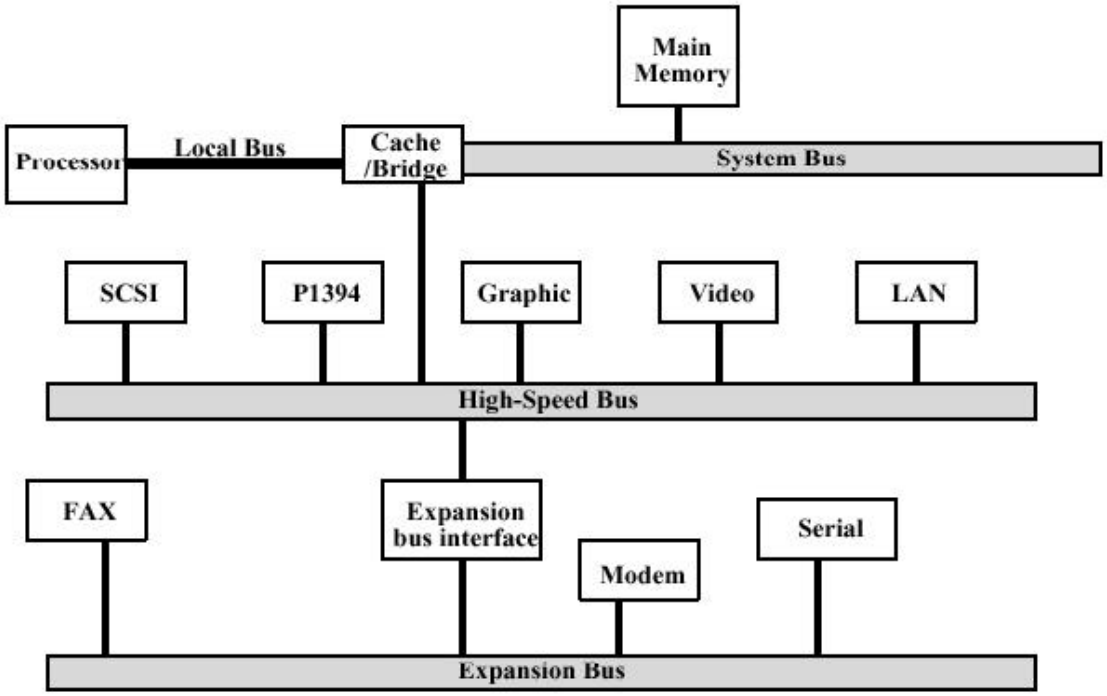
- A great number of devices on a bus will cause performance to suffer
  - Propagation delay - the time it takes for devices to coordinate the use of the bus
  - The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus (in available transfer cycles/second)
- Traditional Hierarchical Bus Architecture
  - Use of a cache structure insulates CPU from frequent accesses to main memory
  - Main memory can be moved off local bus to a system bus
  - Expansion bus interface
    - buffers data transfers between system bus and I/O controllers on expansion bus
    - insulates memory-to-processor traffic from I/O traffic

Traditional Hierarchical Bus Architecture Example



- High-performance Hierarchical Bus Architecture
  - Traditional hierarchical bus breaks down as higher and higher performance is seen in the I/O devices
  - Incorporates a high-speed bus
    - specifically designed to support high-capacity I/O devices
    - brings high-demand devices into closer integration with the processor and at the same time is independent of the processor
    - Changes in processor architecture do not affect the high-speed bus, and vice versa
  - Sometimes known as a mezzanine architecture

High-performance Hierarchical Bus Architecture Example



Elements of Bus Design

- Bus Types
  - Dedicated
    - Separate data & address lines
  - Multiplexed
    - Shared lines
    - Address valid or data valid control line
    - Advantage - fewer lines
    - Disadvantages
      - More complex control
      - Ultimate performance
- Bus Arbitration
  - More than one module controlling the bus
    - e.g. CPU and DMA controller
  - Only one module may control bus at one time
  - Arbitration may be centralised or distributed
- Centralised Arbitration
  - Single hardware device controlling bus access
    - Bus Controller
    - Arbiter
  - May be part of CPU or separate
- Distributed Arbitration

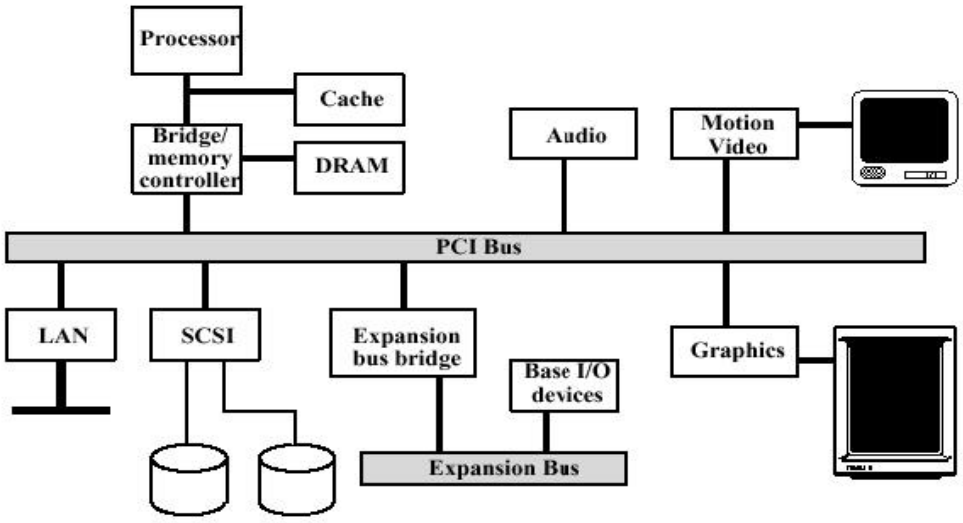
- Each module may claim the bus
- Control logic on all modules

- Timing
  - Co-ordination of events on bus
  - Synchronous
    - Events determined by clock signals
    - Control Bus includes clock line
    - A single 1-0 is a bus cycle
    - All devices can read clock line
    - Usually sync on leading edge
    - Usually a single cycle for an event
- Bus Width
  - Address: Width of address bus has an impact on system capacity i.e. wider bus means greater the range of locations that can be transferred.
  - Data: width of data bus has an impact on system performance i.e. wider bus means number of bits transferred at one time.
- Data Transfer Type
  - Read
  - Write
  - Read-modify-write
  - Read-after-write
  - Block

1.8 PCI

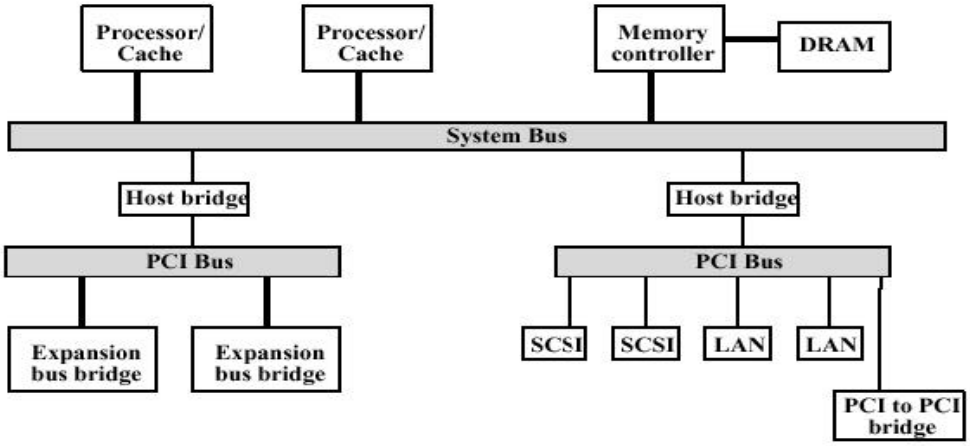
- PCI is a popular high bandwidth, processor independent bus that can function as mezzanine or peripheral bus.
- PCI delivers better system performance for high speed I/O subsystems (graphic display adapters, network interface controllers, disk controllers etc.)
- PCI is designed to support a variety of microprocessor based configurations including both single and multiple processor system.
- It makes use of synchronous timing and centralised arbitration scheme.
- PCI may be configured as a 32 or 64-bit bus.
- Current Standard
  - up to 64 data lines at 33Mhz
  - requires few chips to implement
  - supports other buses attached to PCI bus
  - public domain, initially developed by Intel to support Pentium-based systems
  - supports a variety of microprocessor-based configurations, including multiple processors
  - uses synchronous timing and centralized arbitration

Typical Desktop System



Note: Bridge acts as a data buffer so that the speed of the PCI bus may differ from that of the processor's I/O capability.

Typical Server System



Note: In a multiprocessor system, one or more PCI configurations may be connected by bridges to the processor's system bus.

PCI Bus Lines

- Systems lines
  - Including clock and reset
- Address & Data
  - 32 time mux lines for address/data
  - Interrupt & validate lines
- Interface Control
- Arbitration
  - Not shared
  - Direct connection to PCI bus arbiter

- Error lines
- Interrupt lines
    - Not shared
- Cache support
- 64-bit Bus Extension
    - Additional 32 lines
    - Time multiplexed
    - 2 lines to enable devices to agree to use 64-bit transfer
- JTAG/Boundary Scan
    - For testing procedures

PCI Commands

- Transaction between initiator (master) and target
- Master claims bus
- Determine type of transaction
    - e.g. I/O read/write
- Address phase
- One or more data phases

PCI Enhancements: AGP

- AGP – Advanced Graphics Port
    - Called a port, not a bus because it only connects 2 devices

Chapter – 2
Central Processing Unit

The part of the computer that performs the bulk of data processing operations is called the Central Processing Unit (CPU) and is the central component of a digital computer. Its purpose is to interpret instruction cycles received from memory and perform arithmetic, logic and control operations with data stored in internal register, memory words and I/O interface units. A CPU is usually divided into two parts namely processor unit (Register Unit and Arithmetic Logic Unit) and control unit.

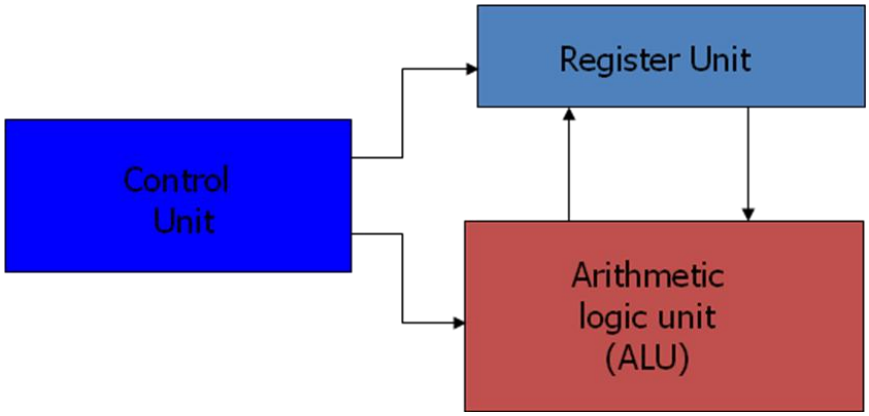


Fig: Components of CPU

Processor Unit:
The processor unit consists of arithmetic unit, logic unit, a number of registers and internal buses that provides data path for transfer of information between register and arithmetic logic unit. The block diagram of processor unit is shown in figure below where all registers are connected through common buses. The registers communicate each other not only for direct data transfer but also while performing various micro-operations.

Here two sets of multiplexers select register which perform input data for ALU. A decoder selects destination register by enabling its load input. The function select in ALU determines the particular operation that to be performed.

For an example to perform the operation: $R_3 \leftarrow R_1 + R_2$

1. MUX A selector (SELA): to place the content of R_1 into bus A.
2. MUX B selector (SELB): to place the content of R_2 into bus B.
3. ALU operation selector (OPR): to provide arithmetic addition A + B.
4. Decoder destination selector (SELD): to transfer the content of the output bus into R_3.

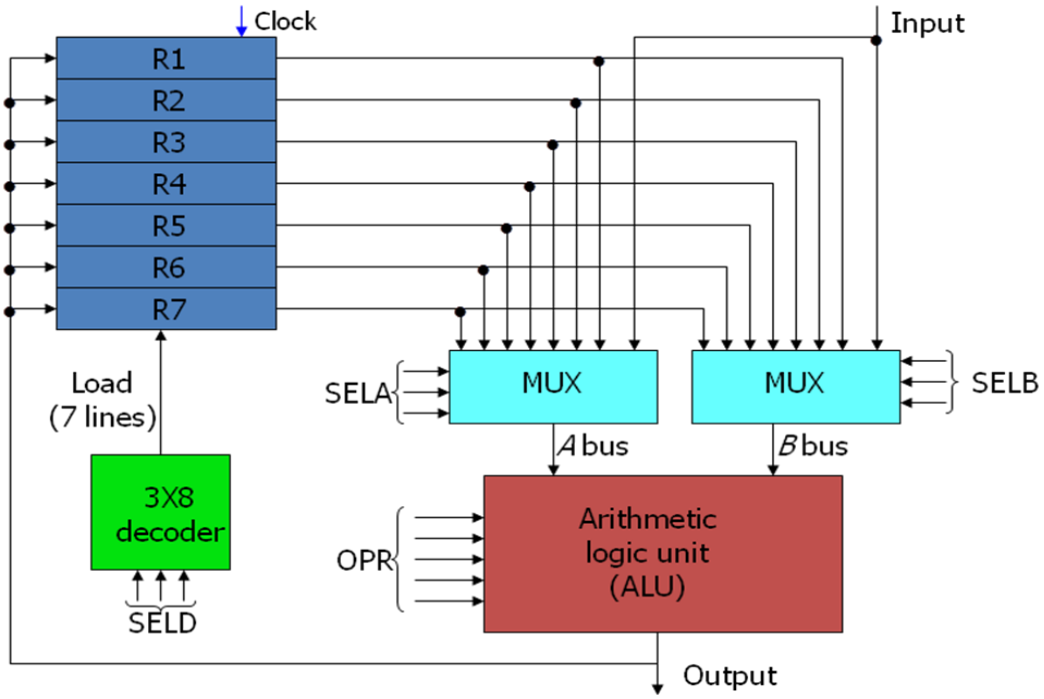


Fig: Processor Unit

Control unit:

The control unit is the heart of CPU. It consists of a program counter, instruction register, timing and control logic. The control logic may be either hardwired or micro-programmed. If it is a hardwired, register decodes and a set of gates are connected to provide the logic that determines the action required to execute various instructions. A micro-programmed control unit uses a control memory to store micro instructions and a sequence to determine the order by which the instructions are read from control memory.

The control unit decides what the instructions mean and directs the necessary data to be moved from memory to ALU. Control unit must communicate with both ALU and main memory and coordinates all activities of processor unit, peripheral devices and storage devices. It can be characterized on the basis of design and implementation by:

- Defining basic elements of the processor
- Describing the micro-operation that processor performs
- Determining the function that the control unit must perform to cause the micro-operations to be performed.

Control unit must have inputs that allow determining the state of system and outputs that allow controlling the behavior of system.

The input to control unit are:

- Flag: flags are headed to determine the status of processor and outcome of previous ALU operation.

- Clock: All micro-operations are performed within each clock pulse. This clock pulse is also called as processor cycle time or clock cycle time.
- Instruction Register: The op-code of instruction determines which micro-operation to perform during execution cycle.
- Control signal from control bus: The control bus portion of system bus provides interrupt, acknowledgement signals to control unit.

The outputs from control unit are:

- Control signal within processor: These signals causes data transfer between registers, activate ALU functions.
- Control signal to control bus: These are signals to memory and I/O module. All these control signals are applied directly as binary inputs to individual logic gate.

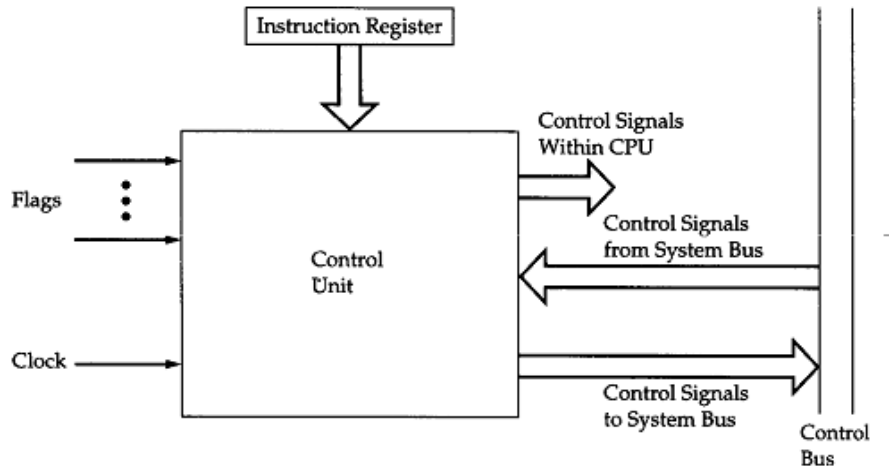


Fig: Control Unit

2.1 CPU Structure and Function

Processor Organization

- Things a CPU must do:
  - Fetch Instructions
  - Interpret Instructions
  - Fetch Data
  - Process Data
  - Write Data

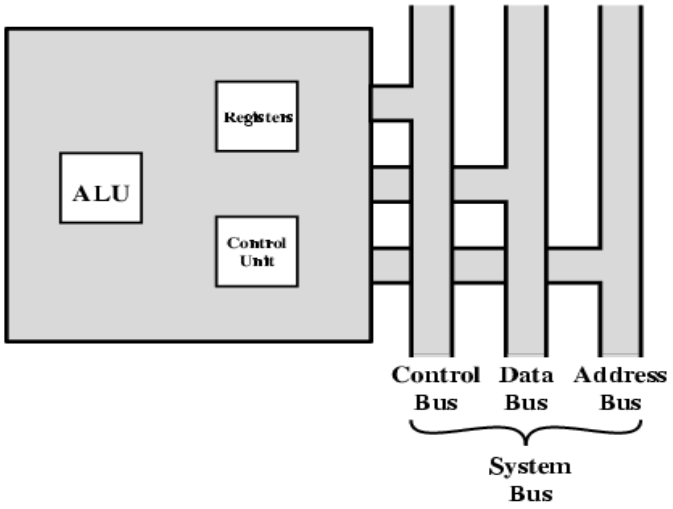


Fig: The CPU with the System Bus

- A small amount of internal memory, called the registers, is needed by the CPU to fulfill these requirements

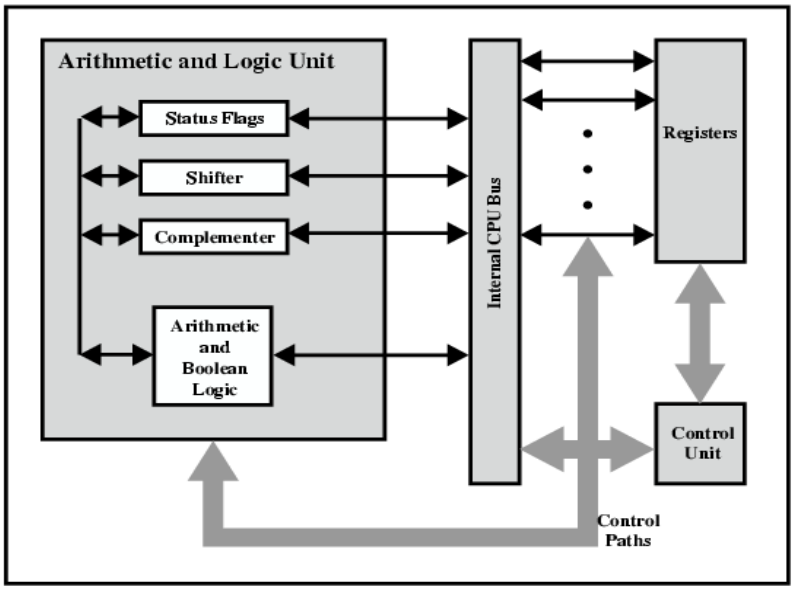


Fig: Internal Structure of the CPU

- Components of the CPU
  - Arithmetic and Logic Unit (ALU): does the actual computation or processing of data
  - Control Unit (CU): controls the movement of data and instructions into and out of the CPU and controls the operation of the ALU.

Register Organization

- Registers are at top of the memory hierarchy. They serve two functions:
  1. User-Visible Registers - enable the machine- or assembly-language programmer to minimize main-memory references by optimizing use of registers
  2. Control and Status Registers - used by the control unit to control the operation

of the CPU and by privileged, OS programs to control the execution of programs

User-Visible Registers

Categories of Use

- General Purpose registers - for variety of functions
- Data registers - hold data
- Address registers - hold address information
- Segment pointers - hold base address of the segment in use
- Index registers - used for indexed addressing and may be auto indexed
- Stack Pointer - a dedicated register that points to top of a stack. Push, pop, and other stack instructions need not contain an explicit stack operand.
- Condition Codes (flags)

Design Issues

- Completely general-purpose registers or specialized use?
  - Specialized registers save bits in instructions because their use can be implicit
  - General-purpose registers are more flexible
  - Trend is toward use of specialized registers
- Number of registers provided?
  - More registers require more operand specifier bits in instructions
  - 8 to 32 registers appears optimum (RISC systems use hundreds, but are a completely different approach)
- Register Length?
  - Address registers must be long enough to hold the largest address
  - Data registers should be able to hold values of most data types
  - Some machines allow two contiguous registers for double-length values
- Automatic or manual save of condition codes?
  - Condition restore is usually automatic upon call return
  - Saving condition code registers may be automatic upon call instruction, or may be manual

Control and Status Registers

- Essential to instruction execution
  - Program Counter (PC)
  - Instruction Register (IR)
  - Memory Address Register (MAR) - usually connected directly to address lines of bus
  - Memory Buffer Register (MBR) - usually connected directly to data lines of bus
- Program Status Word (PSW) - also essential, common fields or flags contained include:
  - Sign - sign bit of last arithmetic operation
  - Zero - set when result of last arithmetic operation is 0
  - Carry - set if last op resulted in a carry into or borrow out of a high-order bit
  - Equal - set if a logical compare result is equality
  - Overflow - set when last arithmetic operation caused overflow
  - Interrupt Enable/Disable - used to enable or disable interrupts
  - Supervisor - indicates if privileged ops can be used

- Other optional registers
  - Pointer to a block of memory containing additional status info (like process control blocks)
  - An interrupt vector
  - A system stack pointer
  - A page table pointer
  - I/O registers
- Design issues
  - Operating system support in CPU
  - How to divide allocation of control information between CPU registers and first part of main memory (usual tradeoffs apply)

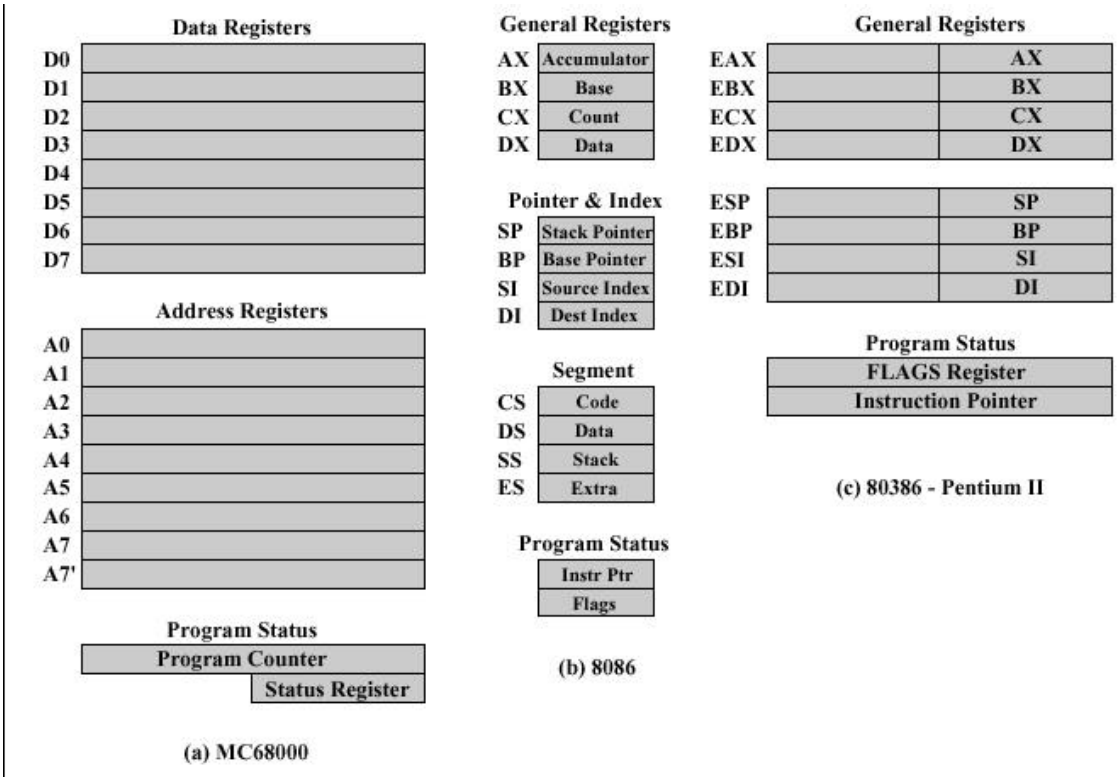


Fig: Example Microprocessor Register Organization

The Instruction Cycle

Basic instruction cycle contains the following sub-cycles.

- Fetch - read next instruction from memory into CPU
- Execute - Interpret the opcode and perform the indicated operation
- Interrupt - if interrupts are enabled and one has occurred, save the current process state and service the interrupt

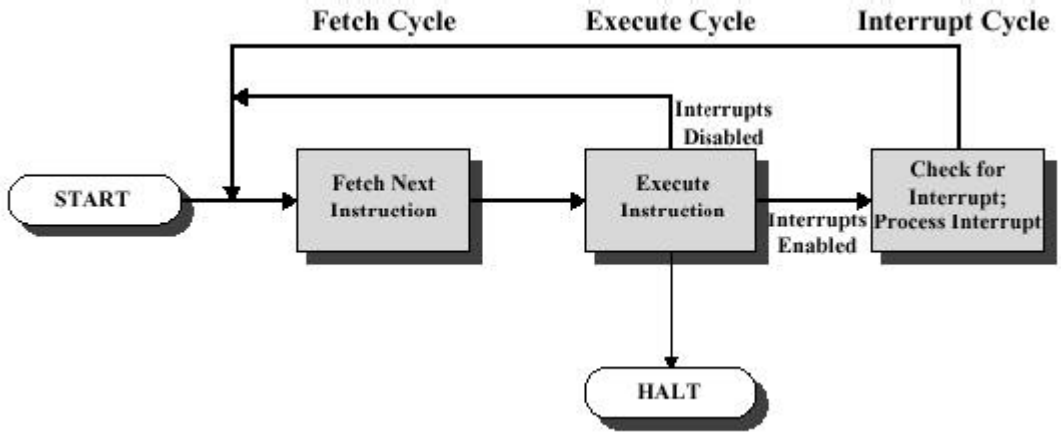


Fig: Instruction Cycles

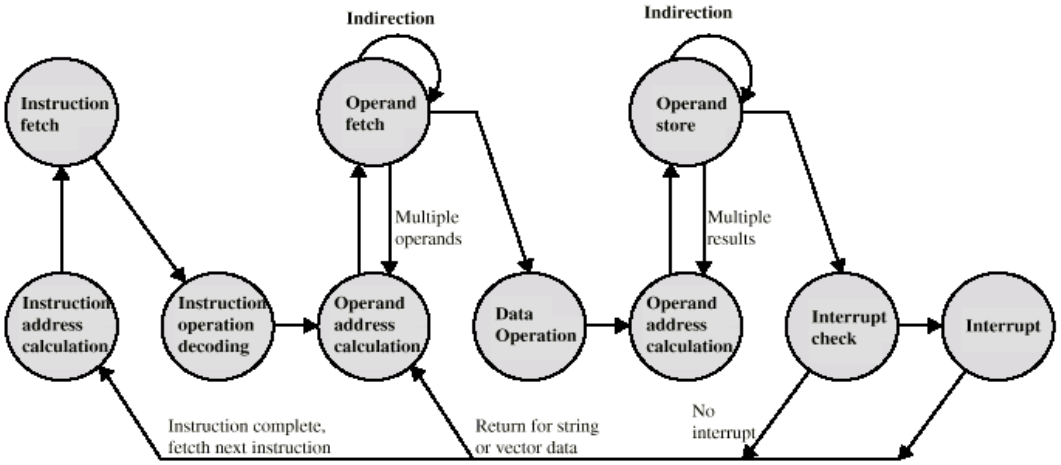


Fig: Instruction Cycle State Diagram

The Indirect Cycle

- Think of as another instruction sub-cycle
- May require just another fetch (based upon last fetch)
- Might also require arithmetic, like indexing

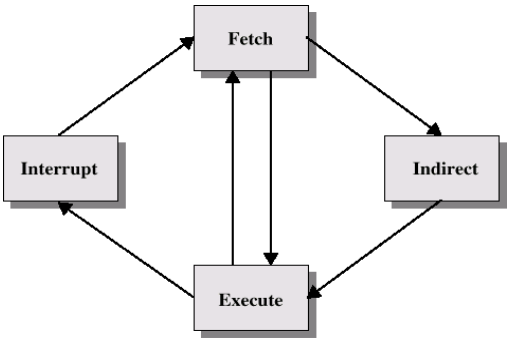


Fig: Instruction Cycle with Indirect

Data Flow

- Exact sequence depends on CPU design
- We can indicate sequence in general terms, assuming CPU employs:
  - a memory address register (MAR)
  - a memory buffer register (MBR)
  - a program counter (PC)
  - an instruction register (IR)

Fetch cycle data flow

- PC contains address of next instruction to be fetched
- This address is moved to MAR and placed on address bus
- Control unit requests a memory read
- Result is
  - placed on data bus
  - result copied to MBR
  - then moved to IR
- Meanwhile, PC is incremented

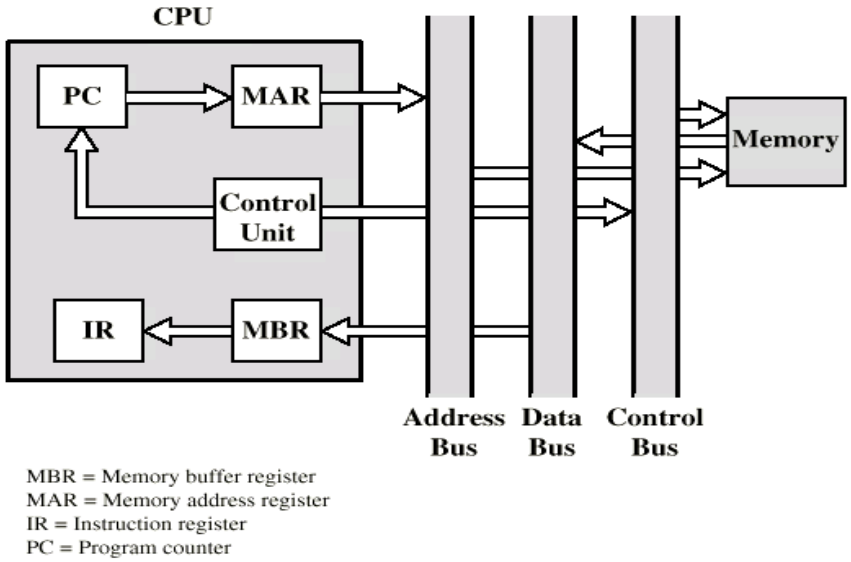


Fig: Data flow, Fetch Cycle

t1: MAR ← (PC)
t2: MBR ← Memory
PC ← PC + 1
t3: IR(Address) ← (MBR(Address))

Indirect cycle data flow

- Decodes the instruction
- After fetch, control unit examines IR to see if indirect addressing is being used. If so:
- Rightmost n bits of MBR (the memory reference) are transferred to MAR
- Control unit requests a memory read, to get the desired operand address into the MBR

t1: MAR ← (IR(Address))
t2: MBR ← Memory
t3: IR(Address) ← (MBR(Address))

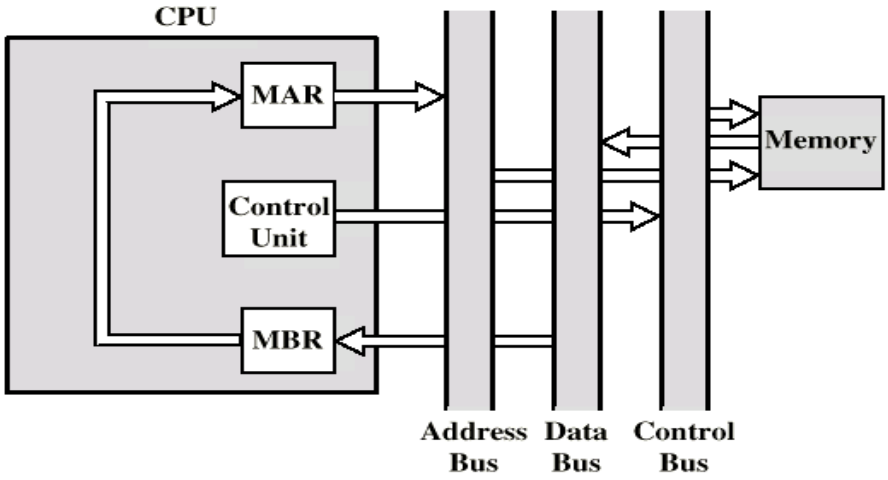


Fig: Data Flow, Indirect Cycle

Execute cycle data flow

- Not simple and predictable, like other cycles
- Takes many forms, since form depends on which of the various machine instructions is in the IR
- May involve
  - transferring data among registers
  - read or write from memory or I/O
  - invocation of the ALU

For example: ADD R_1, X
t1: MAR ← (IR(Address))
t2: MBR ← Memory
t3: R_1 ← (R_1) + (MBR)

Interrupt cycle data flow

- Current contents of PC must be saved (for resume after interrupt), so PC is transferred to MBR to be written to memory
- Save location's address (such as a stack ptr) is loaded into MAR from the control unit
- PC is loaded with address of interrupt routine (so next instruction cycle will begin by fetching appropriate instruction)

t1: MBR ← (PC)
t2: MAR ← save_address
PC ← Routine_address
t3: Memory ← (MBR)

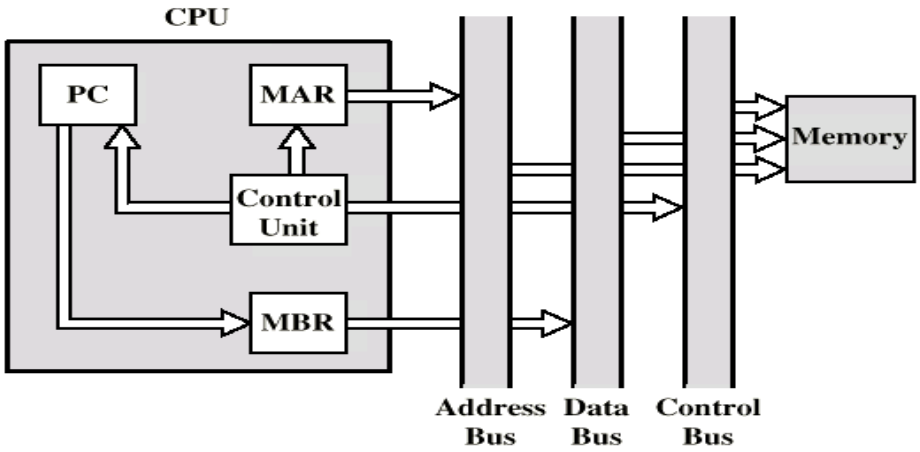
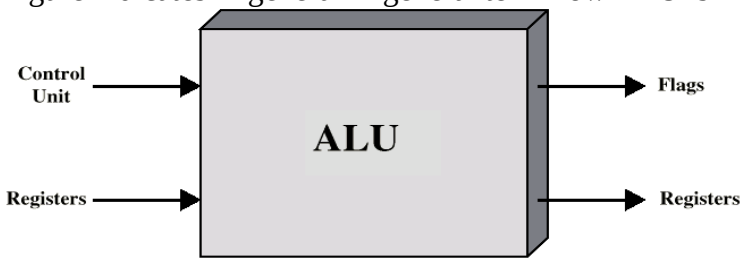


Fig: Data Flow, Interrupt Cycle

2.2 Arithmetic and Logic Unit

ALU is the combinational circuit of that part of computer that actually performs arithmetic and logical operations on data. All of the other elements of computer system- control unit, registers, memory, I/O are their mainly to bring data into the ALU for it to process and then to take the result back out. An ALU & indeed all electronic components in computer are based on the use of simple digital logic device that can store binary digit and perform simple Boolean logic function. Figure indicates in general in general term how ALU is interconnected with rest of the processor.



Data are presented to ALU in register and the result of operation is stored in register. These registers are temporarily storage location within the processor that are connected by signal path to the ALU. The ALU may also set flags as the result of an operation. The flags values are also stored in registers within the processor. The control unit provides signals that control the operation of ALU and the movement of data into an out of ALU.

The design of ALU has three stages.

1. **Design the arithmetic section**
   The basic component of arithmetic circuit is a parallel adder which is constructed with a number of full adder circuits connected in cascade. By controlling the data inputs to the parallel adder, it is possible to obtain different types of arithmetic operations. Below figure shows the arithmetic circuit and its functional table.

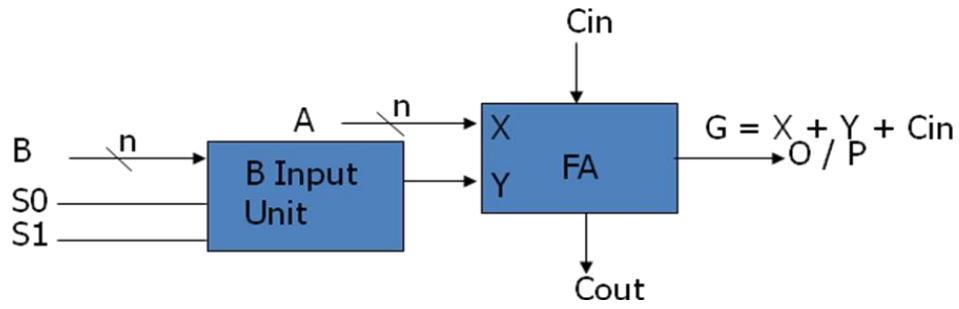


Fig: Block diagram of Arithmetic Unit

Functional table for arithmetic unit:

Select		Input	Output		Microoperation	
S_1	S_0	Y	Cin = 0	Cin = 1	Cin = 0	Cin = 1
0	0	0	A	A+1	Transfer A	Increment A
0	1	B	A+B	A+B+1	Addition	Addition with carry
1	0	B'	A+B'	A+B'+1	Subtraction with borrow	Subtraction
1	1	-1	A-1	A	Decrement A	Transfer A

2. Design the logical section

The basic components of logical circuit are AND, OR, XOR and NOT gate circuits connected accordingly. Below figure shows a circuit that generates four basic logic micro-operations. It consists of four gates and a multiplexer. Each of four logic operations is generated through a gate that performs the required logic. The two selection input S1 and S0 choose one of the data inputs of the multiplexer and directs its value to the output. Functional table lists the logic operations.

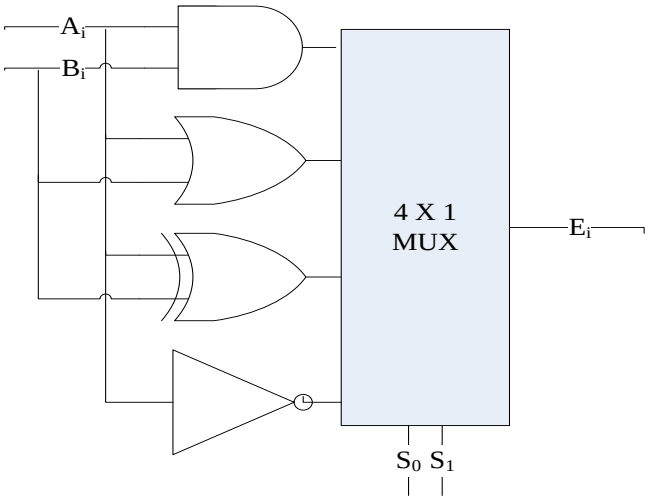


Fig: Block diagram of Logic Unit

Functional table for logic unit:

S1	S0	output	Microoperation
0	0	Ai && Bi	AND
0	1	Ai || Bi	OR
1	0	Ai XOR Bi	XOR
1	1	Ai'	NOT

3. Combine these 2 sections to form the ALU

Below figure shows a combined circuit of ALU where n data input from A are combined with n data input from B to generate the result of an operation at the G output line. ALU has a number of selection lines used to determine the operation to be performed. The selection lines are decoded with the ALU so that selection lines can specify distinct operations. The mode select S_2 differentiate between arithmetic and logical operations. The two functions select S_1 and S_0 specify the particular arithmetic and logic operations to be performed. With three selection lines, it is possible to specify arithmetic operation with S_2 at 0 and logical operation with S_2 at 1.

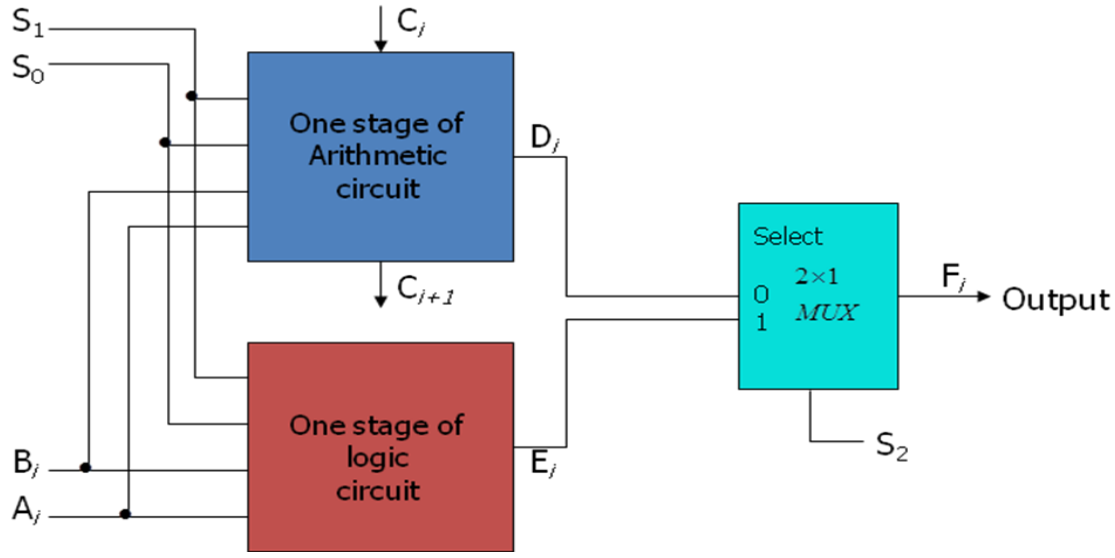
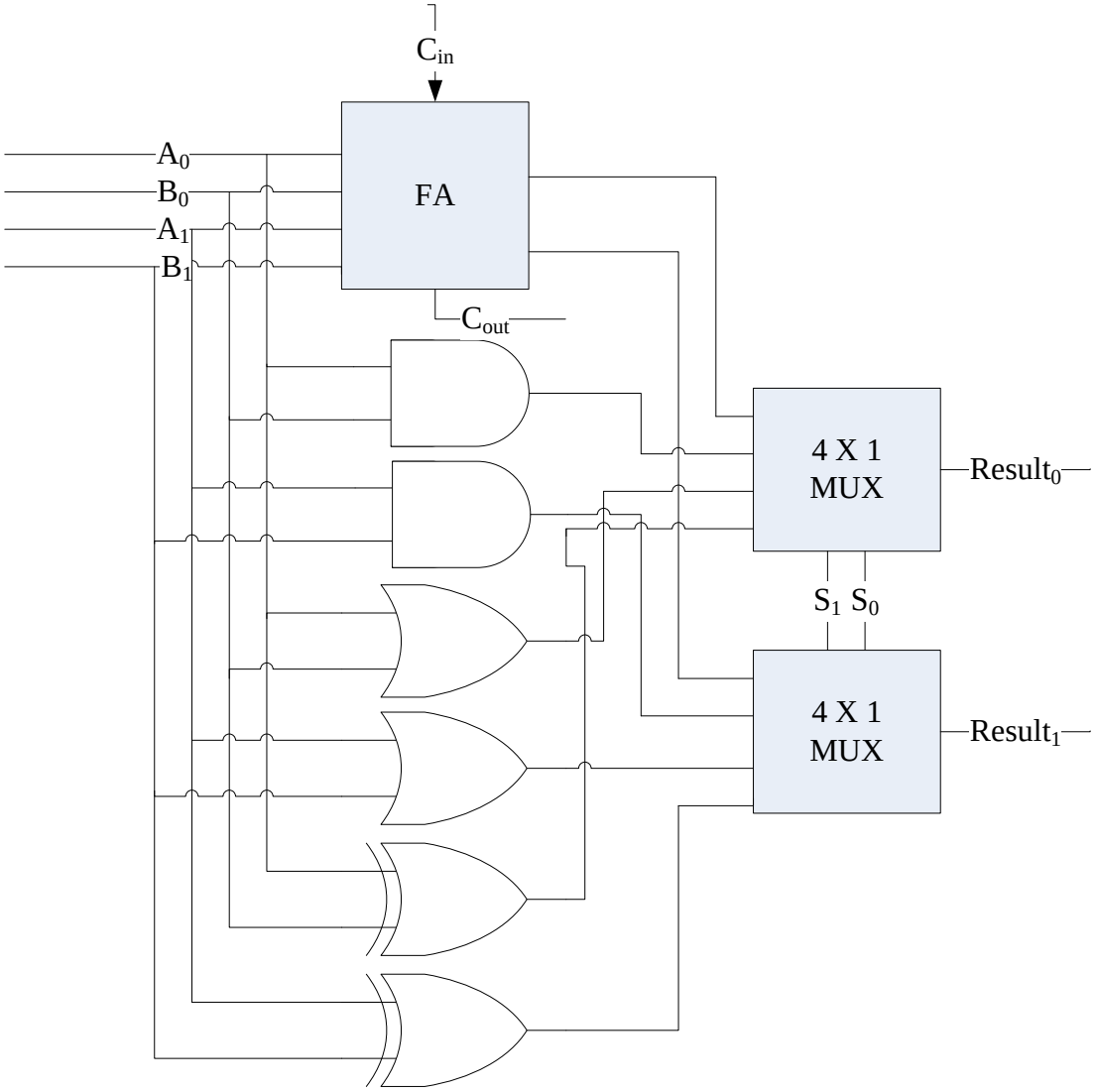


Fig: Block diagram of ALU

Example: Design a 2-bit ALU that can perform addition, AND, OR, & XOR.



2.3 Instruction Formats

The computer can be used to perform a specific task, only by specifying the necessary steps to complete the task. The collection of such ordered steps forms a 'program' of a computer. These ordered steps are the instructions. Computer instructions are stored in central memory locations and are executed sequentially one at a time. The control reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it until the completion of the program.

A computer usually has a variety of Instruction Code Formats. It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction. An n bit instruction that k bits in the address field and m bits in the operation code field come addressed 2^k location directly and specify 2^m different operation.

- The bits of the instruction are divided into groups called fields.
- The most common fields in instruction formats are:
  - An **Operation code** field that specifies the operation to be performed.
  - An **Address field** that designates a memory address or a processor register.
  - A **Mode field** that specifies the way the operand or the effective address is determined.

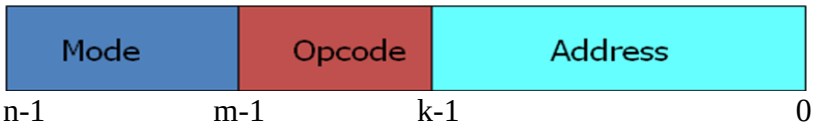


Fig: Instruction format with mode field

The operation code field (Opcode) of an instruction is a group of bits that define various processor operations such as add, subtract, complement, shift etcetera. The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address. Operation specified by an instruction is executed on some data stored in the processor register or in the memory location. Operands residing in memory are specified by their memory address. Operands residing in processor register are specified with a register address.

Types of Instruction

- Computers may have instructions of several different lengths containing varying number of addresses.
- The number of address fields in the instruction format of a computer depends on the internal organization of its registers.
- Most computers fall into one of 3 types of CPU organizations:

Single accumulator organization:- All the operations are performed with an accumulator register. The instruction format in this type of computer uses one address field. For example: ADD X, where X is the address of the operands .

General register organization:- The instruction format in this type of computer needs three register address fields. For example: ADD R1,R2,R3

Stack organization:- The instruction in a stack computer consists of an operation code with no address field. This operation has the effect of popping the 2 top numbers from the stack, operating the numbers and pushing the sum into the stack. For example: ADD

Computers may have instructions of several different lengths containing varying number of addresses. Following are the types of instructions.

1. **Three address Instruction**
   With this type of instruction, each instruction specifies two operand location and a result location. A temporary location T is used to store some intermediate result so as not to alter any of the operand location. The three address instruction format requires a very complex design to hold the three address references.
   Format: Op X, Y, Z; X ← Y Op Z

Example: ADD X, Y, Z; X ← Y + Z

- ADVANTAGE: It results in short programs when evaluating arithmetic expressions.
- DISADVANTAGE: The instructions requires too many bits to specify 3 addresses.

2. **Two address instruction**
   Two-address instructions are the most common in commercial computers. Here again each address field can specify either a processor register, or a memory word. One address must do double duty as both operand and result. The two address instruction format reduces the space requirement. To avoid altering the value of an operand, a MOV instruction is used to move one of the values to a result or temporary location T, before performing the operation.
   Format: Op X, Y; X ← X Op Y
   Example: SUB X, Y; X ← X - Y

3. **One address Instruction**
   It was generally used in earlier machine with the implied address been a CPU register known as accumulator. The accumulator contains one of the operand and is used to store the result. One-address instruction uses an implied accumulator (Ac) register for all data manipulation. All operations are done between the AC register and a memory operand. We use LOAD and STORE instruction for transfer to and from memory and Ac register.
   Format: Op X; Ac ← Ac Op X
   Example: MUL X; Ac ← Ac * X

4. **Zero address Instruction**
   It does not use address field for the instruction like ADD, SUB, MUL, DIV etc. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The name "Zero" address is given because of the absence of an address field in the computational instruction.
   Format: Op; TOS ← TOS Op (TOS – 1)
   Example: DIV; TOS ← TOS DIV (TOS – 1)

Example: To illustrate the influence of the number of address on computer programs, we will evaluate the arithmetic statement X=(A+B)*(C+D) using Zero, one, two, or three address instructions.

1. Three-Address Instructions:
   ADD R1, A, B; R1 ← M[A] + M[B]
   ADD R2, C, D; R2 ← M[C] + M[D]
   MUL X, R1,R2; M[X] ← R1 * R2

It is assumed that the computer has two processor registers R1 and R2. The symbol M[A] denotes the operand at memory address symbolized by A.

2. Two-Address Instructions:
   MOV R1, A; R1 ← M[A]
   ADD R1, B; R1 ← R1 + M[B]

```
MOV R2, C;   R2 ← M[C]
ADD R2, D;   R2 ← R2 + M[D]
MUL R1, R2;  R1 ← R1 * R2
MOV X, R1;   M[X] ← R1
```

3. One-Address Instruction:

```
LOAD A;      Ac ← M[A]
ADD B;       Ac ← Ac + M[B]
STORE T;     M[T] ← Ac
LOAD C;      Ac ← M[C]
ADD D;       Ac ← Ac + M[D]
MUL T;       Ac ← Ac * M[T]
STORE X;     M[X] ← Ac
```

Here, T is the temporary memory location required for storing the intermediate result.

4. Zero-Address Instructions:

```
PUSH A;      TOS ← A
PUSH B;      TOS ← B
ADD;         TOS ← (A + B)
PUSH C;      TOS ← C
PUSH D;      TOS ← D
ADD;         TOS ← (C + D)
MUL;         TOS ← (C + D) * (A + B)
POP X ;      M[X] ← TOS
```

2.4 Addressing Modes

- Specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.
- Computers use addressing mode techniques for the purpose of accommodating the following purposes:-
    - To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data and various other purposes.
    - To reduce the number of bits in the addressing field of the instructions.
        - Other computers use a single binary for operation & Address mode.
        - The mode field is used to locate the operand.
        - Address field may designate a memory address or a processor register.
        - There are 2 modes that need no address field at all (Implied & immediate modes).

Effective address (EA):

- The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode.
- The effective address is the address of the operand in a computational-type instruction.

The most well known addressing mode are:

- Implied Addressing Mode.
- Immediate Addressing Mode
- Register Addressing Mode
- Register Indirect Addressing Mode
- Auto-increment or Auto-decrement Addressing Mode
- Direct Addressing Mode
- Indirect Addressing Mode
- Displacement Address Addressing Mode
- Relative Addressing Mode
- Index Addressing Mode
- Stack Addressing Mode

Implied Addressing Mode:

- In this mode the operands are specified implicitly in the definition of the instruction.
  For example:- CMA - "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied-mode instructions.

Instruction

Opcode

Advantage: no memory reference. Disadvantage: limited operand

Immediate Addressing mode:

- In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field.
- This instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
- These instructions are useful for initializing register to a constant value;
  For example MVI B, 50H

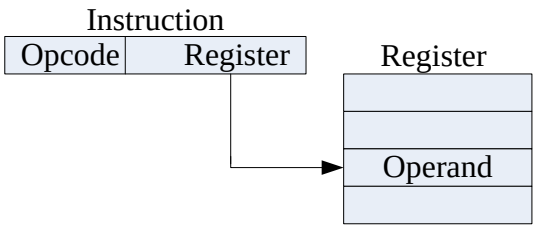
Instruction

Opcode	Operand

It was mentioned previously that the address field of an instruction may specify either a memory word or a processor register. When the address field specifies a processor register, the instruction is said to be in register-mode.
Advantage: no memory reference. Disadvantage: limited operand

Register direct addressing mode:

- In this mode, the operands are in registers that reside within the CPU.
- The particular register is selected from the register field in the instruction.
  For example MOV A, B



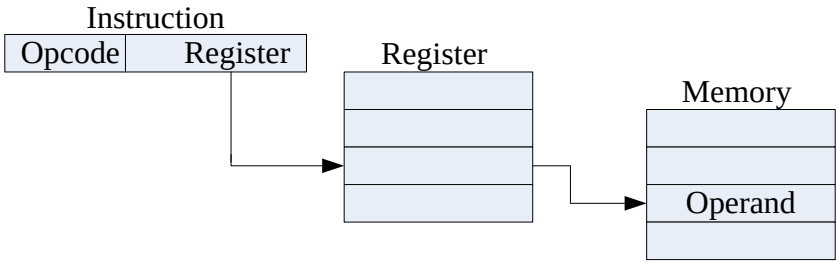
Effective Address (EA) = R

Advantage: no memory reference. Disadvantage: limited address space

Register indirect addressing mode:

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in the memory.
- In other words, the selected register contains the address of the operand rather than the operand itself.
- Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.

  For example LDAX B



Effective Address (EA) = (R)

Advantage: Large address space.

The address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

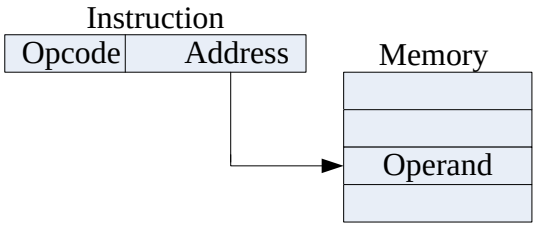
Disadvantage: Extra memory reference

Auto increment or Auto decrement Addressing Mode:

- This is similar to register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the registers refers to a table of data in memory, it is necessary to increment or decrement the registers after every access to the table.
- This can be achieved by using the increment or decrement instruction. In some computers it is automatically accessed.
- The address field of an instruction is used by the control unit in the CPU to obtain the operands from memory.
- Sometimes the value given in the address field is the address of the operand, but sometimes it is the address from which the address has to be calculated.

Direct Addressing Mode

- In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction.
  For example LDA 4000H

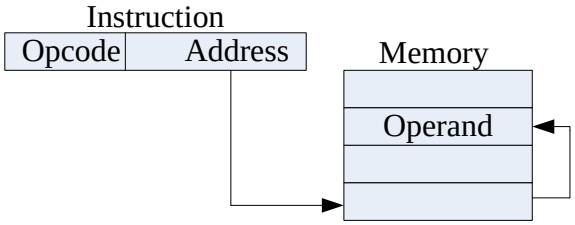


Effective Address (EA) = A

Advantage: Simple. Disadvantage: limited address field

Indirect Addressing Mode

- In this mode the address field of the instruction gives the address where the effective address is stored in memory.
- Control unit fetches the instruction from the memory and uses its address part to access memory again to read the effective address.

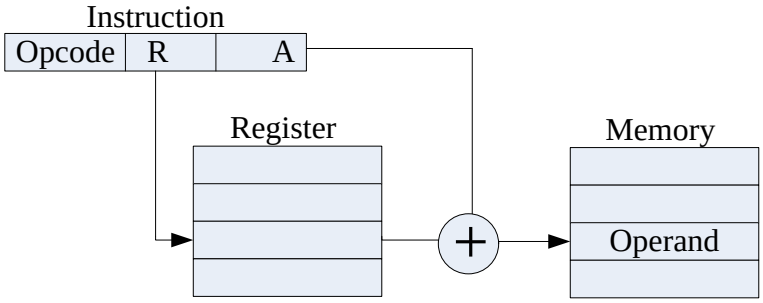


Effective Address (EA) = (A)

Advantage: Flexibility. Disadvantage: Complexity

Displacement Addressing Mode

- A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing.
- The address field of instruction is added to the content of specific register in the CPU.



Effective Address (EA) = A + (R)

Advantage: Flexibility. Disadvantage: Complexity

Relative Addressing Mode

- In this mode the content of the program counter (PC) is added to the address part of the instruction in order to obtain the effective address.
- The address part of the instruction is usually a signed number (either a +ve or a –ve number).
- When the number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.
  Effective Address (EA) = PC + A

Indexed Addressing Mode

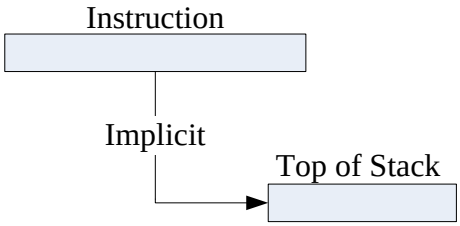
- In this mode the content of an index register (XR) is added to the address part of the instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value.
- Note: If an index-type instruction does not include an address field in its format, the instruction is automatically converted to the register indirect mode of operation.
  Effective Address (EA) = XR + A

Base Register Addressing Mode

- In this mode the content of a base register (BR) is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of the index register.
- The base register addressing mode is used in computers to facilitate the relocation of programs in memory i.e. when programs and data are moved from one segment of memory to another.
  Effective Address (EA) = BR + A

Stack Addressing Mode

- The stack is the linear array of locations. It is some times referred to as push down list or last in First out (LIFO) queue. The stack pointer is maintained in register.



Effective Address (EA) = TOS

Let us try to evaluate the addressing modes with as example.

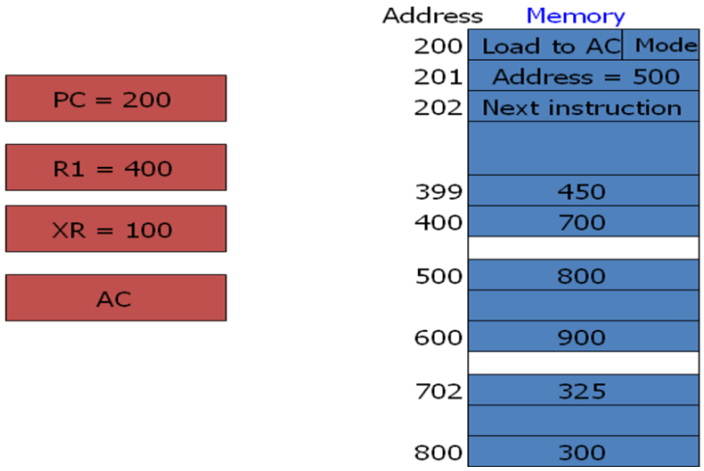


Fig: Numerical Example for Addressing Modes

Addressing Mode	Effective Address	Content of AC
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

Fig: Tabular list of Numerical Example

2.5 Data Transfer and Manipulation

Data transfer instructions cause transfer of data from one location to another without changing the binary information. The most common transfer are between the

- Memory and Processor registers
- Processor registers and input output devices
- Processor registers themselves

Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data manipulation Instructions

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. These instructions perform arithmetic, logic and shift operations.

Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Program Control Instructions

The program control instructions provide decision making capabilities and change the path taken by the program when executed in computer. These instructions specify conditions for altering the content of the program counter. The change in value of program counter as a result of execution of program control instruction causes a break in sequence of instruction execution. Some typical program control instructions are:

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

Subroutine call and Return

A subroutine call instruction consists of an operation code together with an address that specifies the beginning of the subroutine. The instruction is executed by performing two tasks:

- The address of the next instruction available in the program counter (the return address) is stored in a temporary location (stack) so the subroutine knows where to return.
- Control is transferred to the beginning of the subroutine.

The last instruction of every subroutine, commonly called return from subroutine; transfer the return address from the temporary location into the program counter. This results in a transfer of program control to the instruction where address was originally stored in the temporary location.

Interrupt

The interrupt procedure is, in principle, quite similar to a subroutine call except for three variations:

- The interrupt is usually initiated by an external or internal signal rather than from execution of an instruction.
- The address of the interrupt service program is determined by the hardware rather than from the address field of an instruction.
- An interrupt procedure usually stores all the information necessary to define the state of the CPU rather than storing only the program counter.

2.6 RISC and CISC

- Important aspect of computer – design of the instruction set for processor.
- Instruction set – determines the way that machine language programs are constructed.
- Early computers – simple and small instruction set, need to minimize the hardware used.
- Advent of IC – cheaper digital software, instructions intended to increase both in number of complexity.
- Many computers – more than 100 or 200 instructions, variety of data types and large number of addressing modes.

Complex Instruction Set Computers (CISC)

- The trend into computer hardware complexity was influenced by various factors:
  - Upgrading existing models to provide more customer applications
  - Adding instructions that facilitate the translation from high-level language into machine language programs
  - Striving to develop machines that move functions from software implementation into hardware implementation
- A computer with a large number of instructions is classified as a *complex instruction set computer* (CISC).
- One reason for the trend to provide a complex instruction set is the desire to simplify the compilation and improve the overall computer performance.

- The essential goal of CISC architecture is to attempt to provide a single machine instruction for each statement that is written in a high-level language.
- Examples of CISC architecture are the DEC VAX computer and the IBM 370 computer. Other are 8085, 8086, 80x86 etc.

The major characteristics of CISC architecture

- A large number of instructions– typically from 100 to 250 instructions
- Some instructions that perform specialized tasks and are used infrequently
- A large variety of addressing modes—typically from 5 to 20 different modes
- Variable-length instruction formats
- Instructions that manipulate operands in memory
- Reduced speed due to memory read/write operations
- Use of microprogram – special program in control memory of a computer to perform the timing and sequencing of the microoperations – fetch, decode, execute etc.
- Major complexity in the design of microprogram
- No large number of registers – single register set of general purpose and low cost

Reduced Instruction Set Computers (RISC)

A computer uses fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often. It is classified as a *reduced instruction set computer* (RISC).

- RISC concept – an attempt to reduce the execution cycle by simplifying the instruction set
- Small set of instructions – mostly register to register operations and simple load/store operations for memory access
- Each operand – brought into register using load instruction, computations are done among data in registers and results transferred to memory using store instruction
- Simplify instruction set and encourages the optimization of register manipulation
- May include immediate operands, relative mode etc.

The major characteristics of RISC architecture

- Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format
- Single-cycle instruction execution
- Hardwired rather than microprogrammed control

Other characteristics attributed to RISC architecture

- A relatively large number of registers in the processor unit
- Use of overlapped register windows to speed-up procedure call and return
- Efficient instruction pipeline – fetch, decode and execute overlap
- Compiler support for efficient translation of high-level language programs into machine language programs
- Studies that show improved performance for RISC architecture do not differentiate between the effects of the reduced instruction set and the effects of a large register file.
- A large number of registers in the processing unit are sometimes associated with RISC processors.
- RISC processors often achieve 2 to 4 times the performance of CISC processors.
- RISC uses much less chip space; extra functions like memory management unit or floating point arithmetic unit can also be placed on same chip. Smaller chips allow a semiconductor mfg. to place more parts on a single silicon wafer, which can lower the per chip cost dramatically.
- RISC processors are simpler than corresponding CISC processors, they can be designed more quickly.

Comparison between RISC and CISC Architectures

S.N.	RISC	CISC
1	Simple instructions taking one cycle	Complex instructions taking multiple cycles
2	Only load and store memory references	Any instructions may reference memory
3	Heavily pipelined	Not/less pipelined
4	Multiple register sets	Single register set
5	Complexity is in compiler	Complexity is in micro-programming
6	Instructions executed by hardware	Instructions interpreted by micro-programming
7	Fixed format instructions	Variable format instructions
8	Few instructions and modes	Large instructions and modes

Overlapped register windows

- Some computers provide multiple-register banks, and each procedure is allocated its own bank of registers. This eliminates the need for saving and restoring register values.
- Some computers use the memory stack to store the parameters that are needed by the procedure, but this required a memory access every time the stack is accessed.
- A characteristic of some RISC processors is their use of overlapped register windows to provide the passing of parameters and avoid the need for saving and restoring register values.
- The concept of overlapped register windows is illustrated in below figure.
- In general, the organization of register windows will have the following relationships:
  Number of global registers = G
  Number of local registers in each window = L
  Number of registers common to two windows = C
  Number of windows = W
- The number of registers available for each window is calculated as followed:
  Window size = L + 2C + G
- The total number of registers needed in the processor is
  Register file = (L + C)W + G

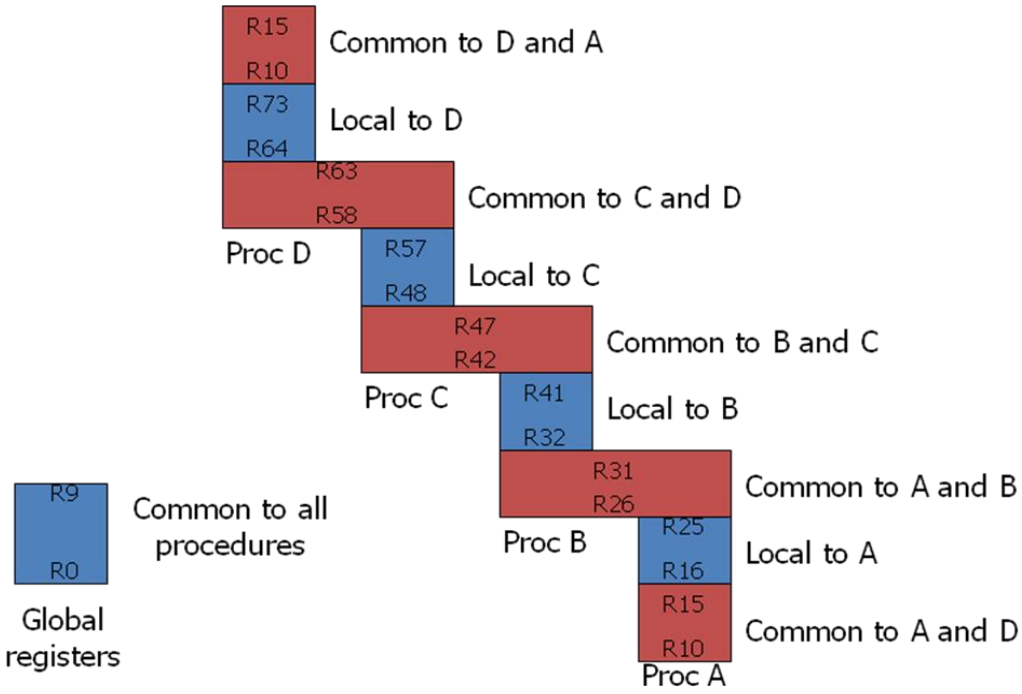


Fig: Overlapped Register Window

- A total of 74 registers
- Global Registers = 10 → common to all procedures
- 64 registers → divided into 4 windows A, B, C & D
- Each register window = 10 registers → local
- Two sets of 16 registers → common to adjacent procedures

Berkeley RISC I

- The Berkeley RISC I is a 32-bit integrated circuit CPU.
  - It supports 32-bit address and either 8-, 16-, or 32-bit data.
  - It has a 32-bit instruction format and a total of 31 instructions.
  - There are three basic addressing modes: Register addressing, immediate operand, and relative to PC addressing for branch instructions.
  - It has a register file of 138 registers; 10 global register and 8 windows of 32 registers in each
  - The 32 registers in each window have an organization similar to overlapped register window.

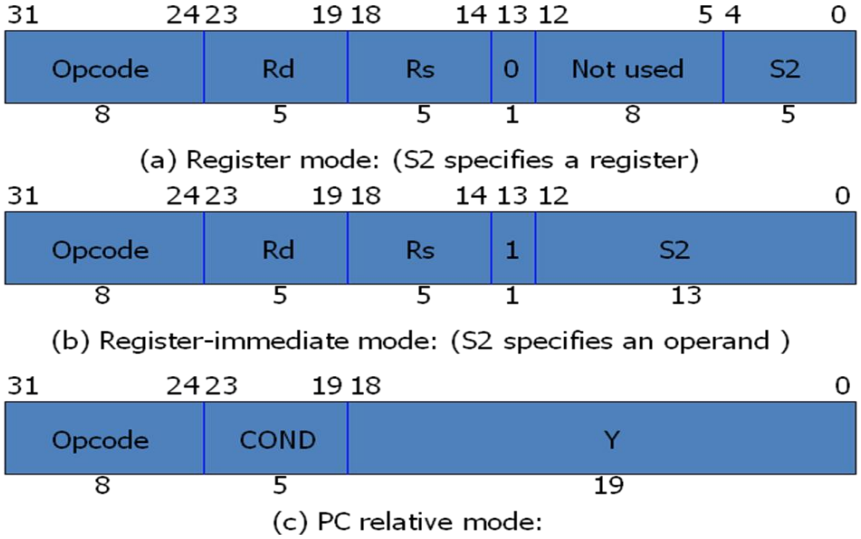
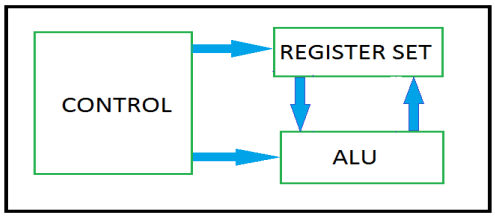


Fig: Instruction Format of Berkeley RISC I

- Above figure shows the 32-bit instruction formats used for *register-to-register instructions* and memory access instructions.
- Seven of the bits in the operation code specify an operation, and the eighth bit indicates whether to update the status bits after an ALU operation.
- For *register-to-register instructions* :
  - The 5-bit R_d field select one of the 32 registers as a destination for the result of the operation
  - The operation is performed with the data specified in fields Rs and S_2.
  - Thus the instruction has a three-address format, but the second source may be either a register or an immediate operand.
- For memory access instructions:
  - Rs to specify a 32-bit address in a register
  - S_2 to specify an offset
  - Register R_0 contains all 0's, so it can be used in any field to specify a zero quantity
- The third instruction format combines the last three fields to form a 19-bit relative address Y and is used primarily with the jump and call instructions.
  - The *COND field* replaces the Rd field for jump instructions and is used to specify one of *16 possible branch conditions*.

2.7 64 – bit Processor

- The brain of the PC is processor or CPU.
- It performs the system's calculating and processing operations.
- The term N-bits means that its ALU, internal registers and most of its instructions are designed to work with N-bit binary words.
- The major components of CPU are:



- 64-bit processors have 64-bit ALUs, 64-bit registers, and 64-bit buses.
- A 64-bit register can address up to 2^{64} bytes of logical address.
- 64-bit processors have been with us since 1992.
- Eg: 64-bit AMD processor.

Internal Architecture

- The internal logic design of microprocessor which determines how and when various operations are performed.
- The various function performed by the microprocessor can be classified as:
  - Microprocessor initiated operations
  - Internal operations
  - Peripheral operations
- Microprocessor initiated operations mainly deal with memory and I/O read and write operations.
- Internal operations determines how and what operations can be performed with the data.The operations include:
  1. storing
  2. performing arithmetic and logical operations
  3. test for conditions
  4. store in the stack
- External initiated operations are initiated by the external devices to perform special operations like reset, interrupt, ready, etc.
- The block diagram of 64-bit microprocessor is shown below.
- The major parts of the block diagram are:

- General register unit
- Control and decoding unit
- Bus unit
- Cache memory unit
- Floating point register unit
- Issue ports

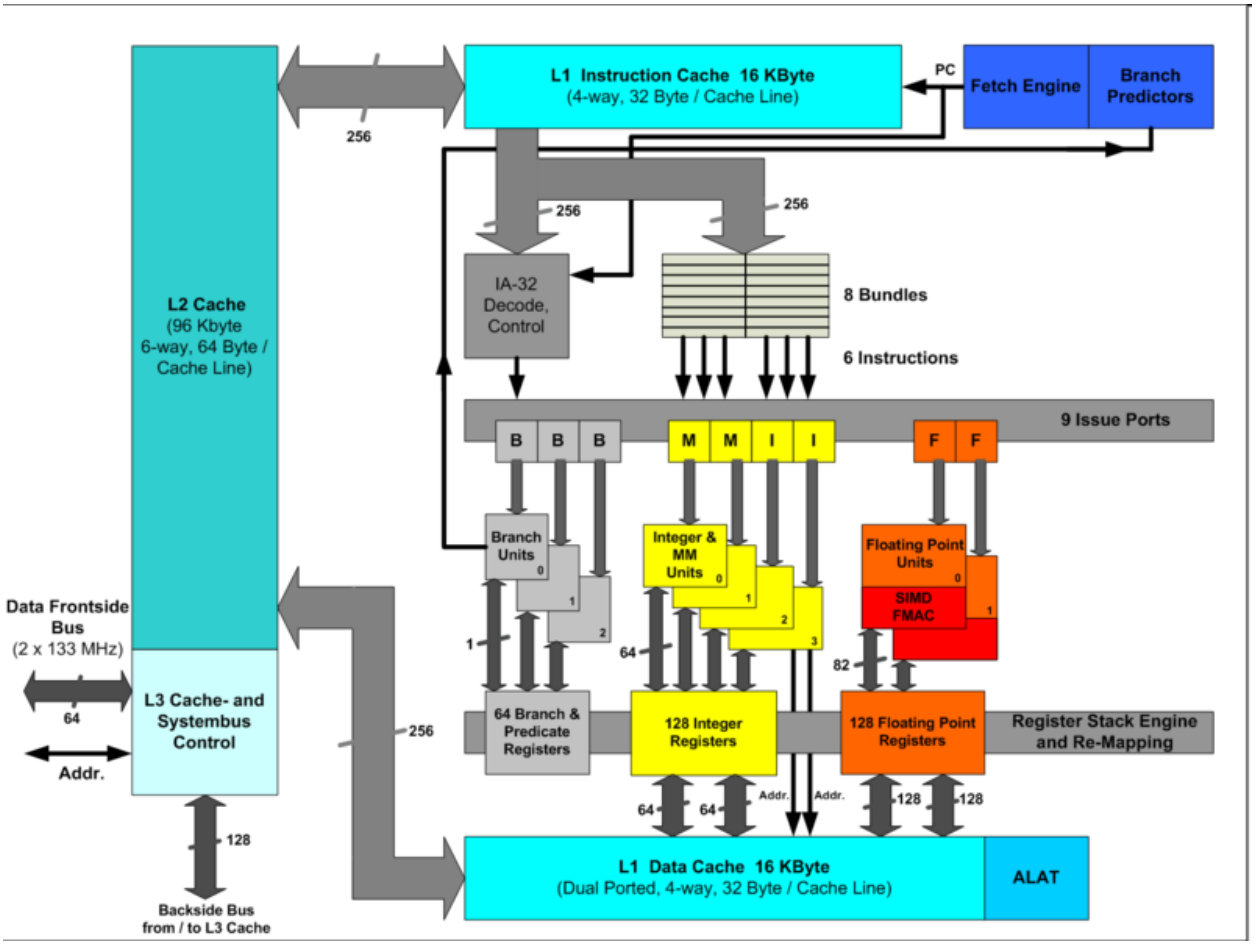


Fig: Block diagram of 64-bit internal architecture

Architecture Elements

- Addressing Modes
- General Purpose Registers
- Non-modal and modal Instructions
- New Instructions in Support of 64-bit
- New immediate Instructions

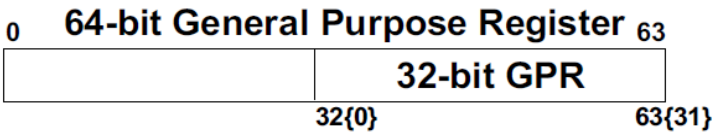
Addressing modes

- This addressing mode determines the working environment. i.e 24,32 or 64 bit mode
- PSW bits 31 and 32 designate addressing mode (out of 64 bit).
  - Addressing modes bits:00=24 bit-mode

01=32 bit-mode
11=64 bit-mode

General purposes register (GPR)

- The register is treated as 64-bits for:
  - Address generation in 64-bit mode.



- The register is treated as 32-bits for:
  - Address generation in 24/32-bit mode.

New instructions in 64-bit:

- Load Reversed - LRV, LRVR
- Multiply Logical - ML, MLR
- Divide Logical - DL, DLR
- Add Logical w/ Carry - ALC
- Subtract Logical w/ Borrow - SLB
- Store Reversed - STRV
- Rotate Left Single Logical – RLL

New immediate Instructions

- Load Logical Immediate
- Insert Logical Immediate
- AND Immediate
- OR Immediate
- Test Under Mask (High/Low)

Comparison of 64-bit with 32-bit

- Contains 32-bit data lines whereas 64-bit contains 64 data lines.
- Can address max 2^32(4 GB) of data whereas 64 bit can address 2^64(18 billion GB).
- Speed and execution is both fast in 64-bit processors.
- 64-bit processors can drive 32-bit applications even faster, by handling more data per clock cycle than a 32-bit processor.
- The table shows the basic difference between two:

Table 1. Resources required to load, add, and store two 64-bit integers

Operation	Resources on 32-bit processor	Resources on 64-bit processor	Effective improvement with 64-bit
Load two 64-bit integers	• Requires four (4) 32-bit registers to hold data • Requires 4 load instructions	• Requires two (2) 64-bit registers to hold data • Requires 2 load instructions	Reduced number of instructions to load data by one half and fewer registers consumed by one half
Add two 64-bit integers	• Requires 2 addition instructions; an add with carry and an extended to include the carry	• Requires one addition instruction	Reduced number of instructions by one half and reduced interlocking among instructions and carry status
Store two 64-bit integers	• Requires four (4) 32-bit registers to hold data • Requires 4 store instructions to save data	• Requires two (2) 64-bit registers to hold data • Requires 2 store instructions to save data	Reduced number of instructions to store data by one half and registers consumed by one half
Total resources	10 instructions issued and 4 registers plus carry field	5 instructions issued and 2 registers used	One half the instructions, less than one half the resources consumed

Advantages and disadvantages:

advantages	Disadvantages
➢ Previous processors can have max 4 Gb of physical memory but 64-bit can handle more. ➢ More general purpose registers than in older processors. ➢ Significant increase in speed due to wider data bus and processing is fast.	➢ Compatibility difficulty with existing software as they are mostly developed to the 32-bit processors. ➢ 64-bit OS must have 64-bit drivers, for working efficiently. ➢ They are costly.

Chapter – 3
Control Unit

3.1 Control Memory

- The function of the *control unit* in a digital computer is to initiate *sequences of microoperations*.
- When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be *hardwired*.
- *Microprogramming* is a second alternative for designing the control unit of a digital computer.
    - The principle of microprogramming is an elegant and systematic method for controlling the microoperation sequences in a digital computer.
- In a bus-organized systems, the control signals that specify microoperations are groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units.
- A control unit whose binary control variables are stored in memory is called a *microprogrammed control unit*.
- A memory that is part of a control unit is referred to as a *control memory*.
    - Each word in control memory contains within it a microinstruction.
    - A sequence of microinstructions constitutes a microprogram.
    - Can be either read-only memory(*ROM*) or writable control memory *(dynamic microprogramming*)
- A computer that employs a microprogrammed control unit will have two separate memories:
    - A main memory
    - A control memory
- The general configuration of a microprogrammed control unit is demonstrated in the block diagram of Fig. 3.1.
    - The *control memory* is assumed to be a ROM, within which all control information is permanently stored.
    - The *control address register* specifies the address of the microinstruction.
    - The *control data register* holds the microinstruction read from memory.
- Thus a microinstruction contains bits for initiating microoperations in the data processor part and bits that determine the address sequence for the control memory.

Extra Stuff:

Microprogram

- Program stored in memory that generates all the control signals required to execute the instruction set correctly
- Consists of microinstructions

Microinstruction

- ❖ Contains a control word and a sequencing word
    - Control Word - All the control information required for one clock cycle
    - Sequencing Word - Information needed to decide the next microinstruction address
- ❖ Vocabulary to write a microprogram

Control Memory (Control Storage: CS)

- Storage in the microprogrammed control unit to store the microprogram

Writeable Control Memory(Writeable Control Storage:WCS)

- ❖ CS whose contents can be modified
  - Allows the microprogram can be changed
  - Instruction set can be changed or modified

Dynamic Microprogramming

- Computer system whose control unit is implemented with a microprogram in WCS
- Microprogram can be changed by a systems programmer or a user

Microrogrammed Sequencer

- The next address generator is sometimes called a *microprogram sequencer*, as it determines the address sequence that is read from control memory.
- Typical functions of a microprogram sequencer are:
  - Incrementing the control address register by one
  - Loading into the control address register an address from control memory
  - Transferring an external address
  - Loading an initial address to start the control operations

Pipeline Register

- The data register is sometimes called a *pipeline register*.
  - It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction.
    - This configuration requires a *two-phase clock*
  - The system can operate by applying a *single-phase clock* to the address register.
    - Without the control data register
    - Thus, the control word and next-address information are taken directly from the control memory.

Advantages

- The main advantage of the microprogrammed control is the fact that once the hardware configuration is established; there should be no need for further hardware or wiring change.
- Most computers based on the reduced instruction set computer (RISC) architecture concept use *hardwired control* rather than a *control memory with a microprogram*. (Why?)

A Microprogram Control Unit that determines the Microinstruction Address to be executed in the next clock cycle

- In-line Sequencing
- Branch
- Conditional Branch
- Subroutine
- Loop
- Instruction OP-code mapping

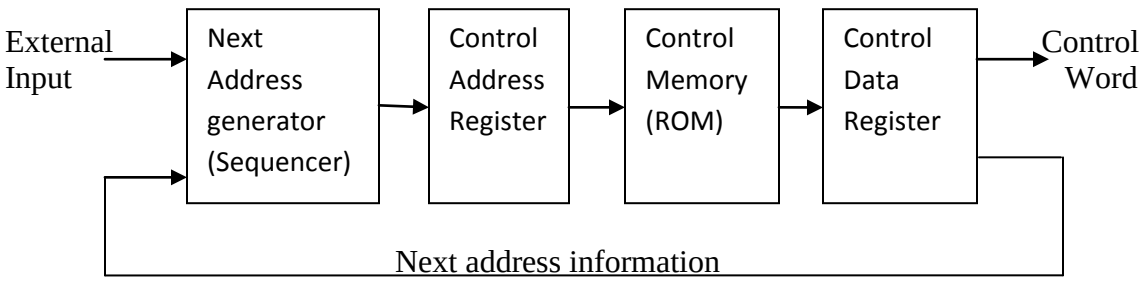


Fig 3-1: Microprogrammed Control Organization

3.2 Addressing sequencing

- Microinstructions are stored in control memory in groups, with each group specifying a *routine*.
- Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction.
- To appreciate the address sequencing in a microprogram control unit:
    - An initial address is loaded into the control address register when power is turned on in the computer.
    - This address is usually the address of the first microinstruction that activates the instruction fetch routine.
    - The control memory next must go through the routine that determines the effective address of the operand.
    - The next step is to generate the microoperations that execute the instruction fetched from memory.
- The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a *mapping* process.
- The address sequencing capabilities required in a control memory are:
    - Incrementing of the control address register
    - Unconditional branch or conditional branch, depending on status bit conditions
    - A mapping process from the bits of the instruction to an address for control memory
    - A facility for subroutine call and return
- Fig. 3-2 shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.
- The microinstruction in control memory contains
    - a set of bits to initiate microoperations in computer registers
    - Other bits to specify the method by which the next address is obtained

Sequencing Capabilities Required in Control Storage

- Incrementing of the control address register
- Unconditional and conditional branches
- A mapping process from the bits of the machine instruction to an address for control memory
- A facility for subroutine call and return

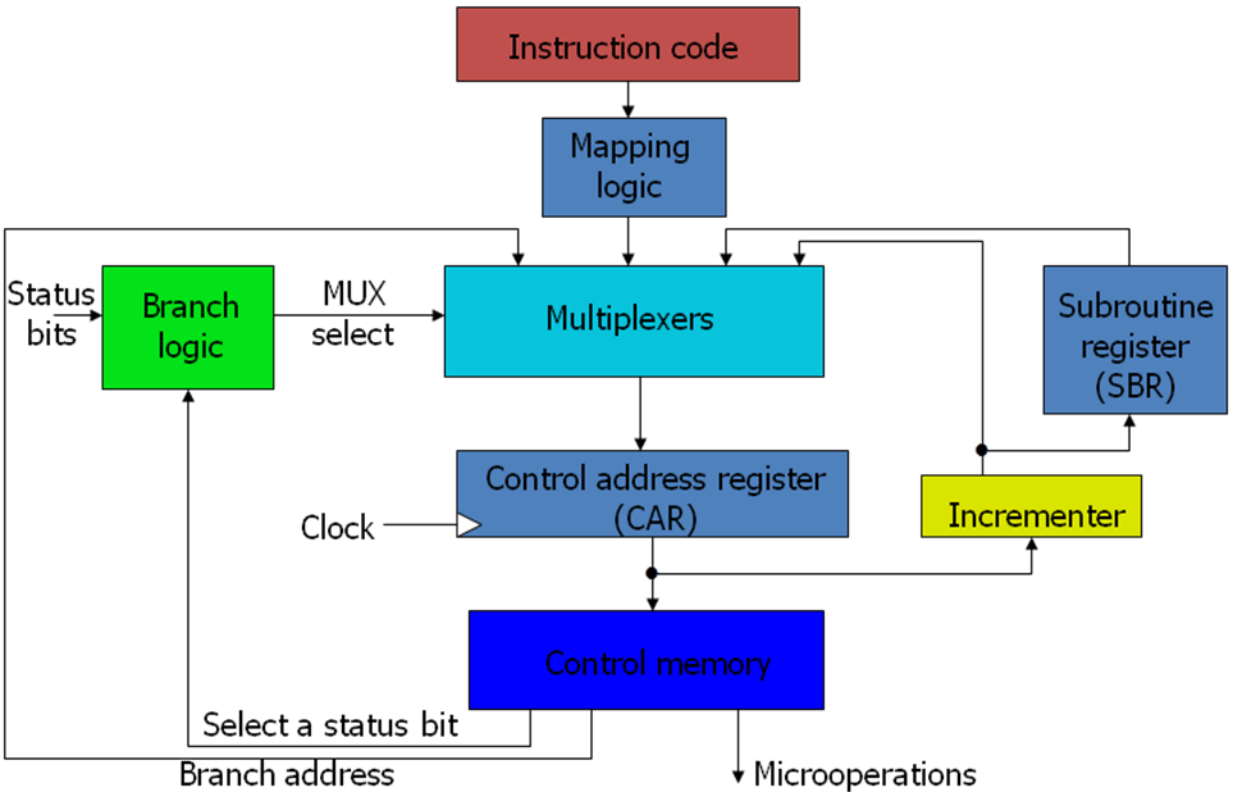


Fig 3-2: Selection of address for control memory

Conditional Branching

- The branch logic of Fig. 3-2 provides decision-making capabilities in the control unit.
- The status conditions are special bits in the system that provides parameter information.
  - e.g. the carry-out, the sign bit, the mode bits, and input or output status
- The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.
- The branch logic hardware may be implemented by multiplexer.
  - Branch to the indicated address if the condition is met;
  - Otherwise, the address register is incremented.
- An unconditional branch microinstruction can be implemented by loading the branch address from control memory into the control address register.
- If Condition is true, then Branch (address from the next address field of the current microinstruction)
  else Fall Through
- Conditions to Test: O(overflow), N(negative), Z(zero), C(carry), etc.

Unconditional Branch

- Fixing the value of one status bit at the input of the multiplexer to 1

Mapping of Instructions

- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction is located.

- The status bits for this type of branch are the bits in the operation code part of the instruction.
- One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in Fig. 3-3.
  - Placing a 0 in the most significant bit of the address
  - Transferring the four operation code bits
  - Clearing the two least significant bits of the control address register
- This provides for each computer instruction a microprogram routine with a capacity of *four microinstructions*.
  - If the routine needs *more than* four microinstructions, it can use addresses 1000000 through 1111111.
  - If it uses *fewer than* four microinstructions, the unused memory locations would be available for other routines.
- One can extend this concept to a more general mapping rule by using a *ROM* or *programmable logic device (PLD)* to specify the mapping function.

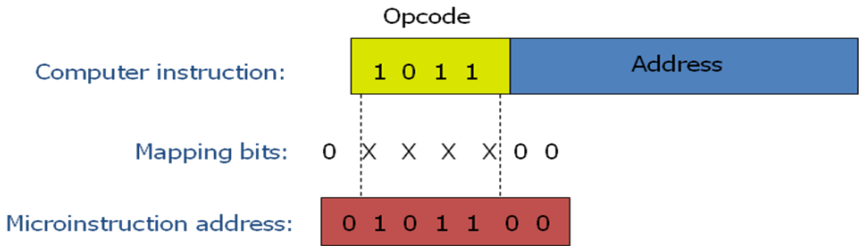


Fig 3-3: Mapping from instruction code to microinstruction address

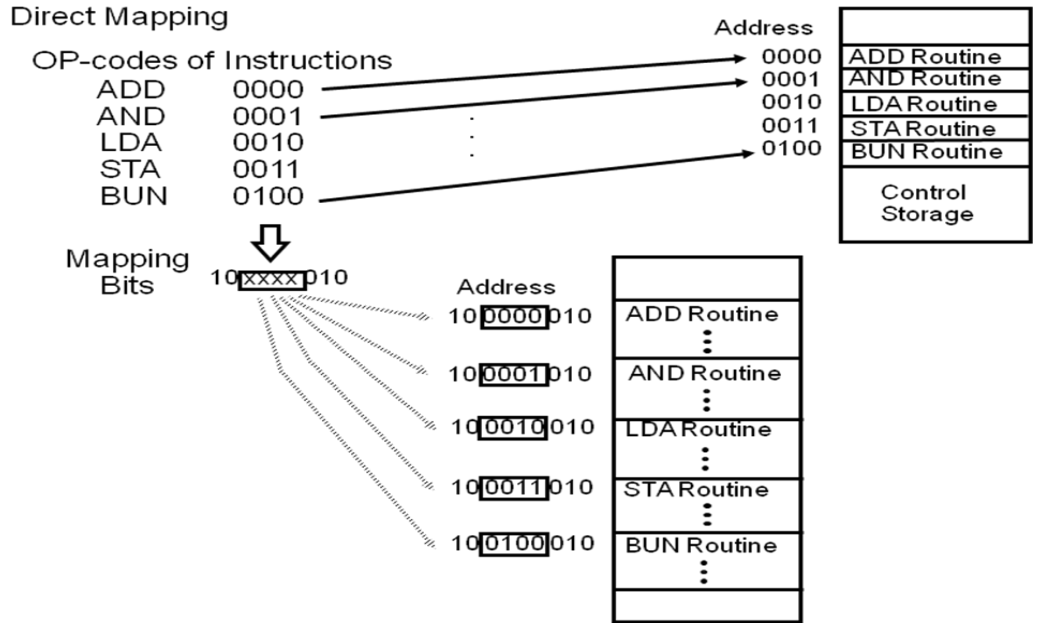


Fig 3-3 (a): Direct mapping

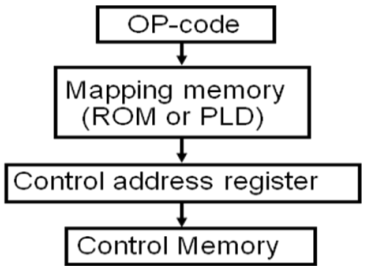


Fig 3-3 (b): Mapping Function Implemented by ROM and PLD

Mapping from the OP-code of an instruction to the address of the Microinstruction which is the starting microinstruction of its execution microprogram.

Subroutine

- Subroutines are programs that are used by other routines to accomplish a particular task.
- Microinstructions can be saved by employing subroutines that use common sections of microcode.
- e.g. effective address computation
- The subroutine register can then become the source for transferring the address for the return to the main routine.
- The best way to structure a register file that stores addresses for subroutines is to organize the registers in a last-in, first-out (LIFO) stack.

3.3 Computer configuration

- Once the configuration of a computer and its microprogrammed control unit is established, the designer's task is to generate the *microcode* for the control memory.
- This microcode generation is called *microprogramming*.
- The block diagram of the computer is shown in Below Fig.
- Two memory units
  - A main memory for storing instructions and data
  - A control memory for storing the microprogram
- Four registers are associated with the processor unit
  - Program counter *PC*, address register *AR*, data register *DR*, accumulator register *AC*
- The control unit has a control address register *CAR* and a subroutine register *SBR*.
- The control memory and its register are organized as a *microprogrammed control unit*, as shown in Fig. 3-2.
- The transfer of information among the registers in the processor is done through *multiplexers rather than a common bus*.

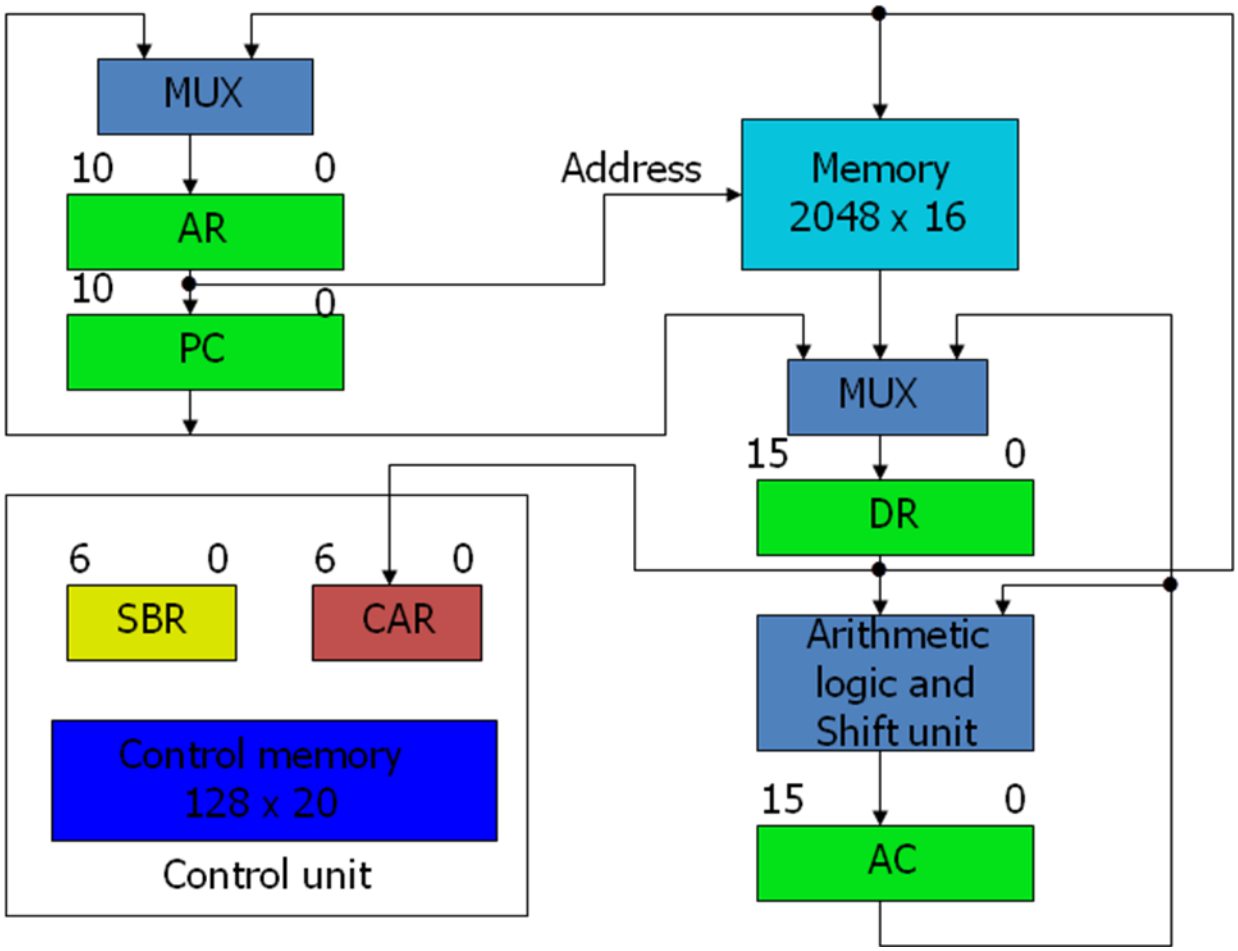


Fig 3-4: Computer hardware configuration

3.4 Microinstruction Format

Computer Instruction Format

- The computer instruction format is depicted in Fig. 3-5(a).
- It consists of three fields:
  - A 1-bit field for indirect addressing symbolized by *I*
  - A 4-bit operation code (*opcode*)
  - An 11-bit address field
- Fig. 3-5(b) lists four of the 16 possible memory-reference instructions.

Fig. 3-5 (a): Instruction Format

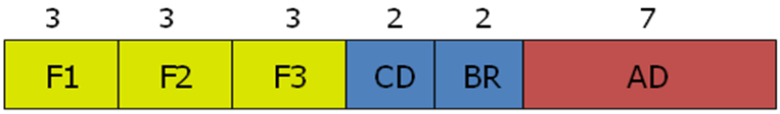
Symbol	Opcode	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	If(AC<0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

EA is the effective address

Fig 3-5 (b): Four Computer Instructions

Microinstruction Format

- The microinstruction format for the control memory is shown in Fig. 3-6.
- The 20 bits of the microinstruction are divided into four functional parts.
  - The three fields F1, F2, and F3 specify *microoperations* for the computer.
  - The CD field selects *status bit conditions*.
  - The BR field specifies *the type of branch*.
  - The AD field contains a *branch address*.



F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

Fig 3-6: Microinstruction code format

Microoperations

- The three bits in each field are encoded to specify *seven distinct microoperations* as listed in Table 3-1.
  - No more than *three* microoperations can be chosen for a microinstruction, one from each field.
  - If fewer than three microoperations are used, one or more of the fields will use the binary code 000 for no operation.
- It is important to realize that two or more conflicting microoperations cannot be specified simultaneously. e.g. 010 001 000
- Each microoperation in Table 3-1 is defined with a register transfer statement and is assigned a symbol for use in a *symbolic microprogram*.

F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC V DR	OR
011	AC ← AC ∧DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC $\oplus$ DR	XOR
010	AC ← $\overline{AC}$	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC +1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

CD	Condition	Symbol	Comments
00	Always = 1	U	Uncondition branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of *AC*
11	AC = 0	Z	Zero value in *AC*

BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1 CAR ← CAR +1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR +1 if condition = 1 CAR ← CAR +1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0, 1, 6) ← 0

Table 3-1 : Symbols and Binary code for Microinstruction Fields

Condition and Branch Field

- The CD field consists of two bits which are encoded to specify four status bit conditions as listed in Table 3-1.
- The BR field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction.
    - The *jump and call* operations depend on the value of the CD field.
    - The two operations are identical except that a call microinstruction stores the *return address* in the subroutine register SBR.
    - Note that the last two conditions in the BR field are *independent of* the values in the CD and AD fields.

3.5 Symbolic Microinstructions

- The symbols defined in Table 3-1 cab be used to specify microinstructions in symbolic form.
- Symbols are used in microinstructions as in assembly language
- The simplest and most straightforward way to formulate an assembly language for a microprogram is to define symbols for each field of the microinstruction and to give users the capability for defining their own symbolic addresses.
- A symbolic microprogram can be translated into its binary equivalent by a microprogram assembler.

Sample Format

Five fields: label; micro-ops; CD; BR; AD

- The label field: may be empty or it may specify a symbolic address terminated with a colon
- The microoperations field: of one, two, or three symbols separated by commas , the NOP symbol is used when the microinstruction has no microoperations
- The CD field: one of the letters {U, I, S, Z} can be chosen where
  - U: Unconditional Branch
  - I: Indirect address bit
  - S: Sign of AC
  - Z: Zero value in AC
- The BR field: contains one of the four symbols {JMP, CALL, RET, MAP}
- The AD field: specifies a value for the address field of the microinstruction with one of {Symbolic address, NEXT, empty}
  - When the BR field contains a RET or MAP symbol, the AD field is left empty

Fetch Subroutine

During FETCH, Read an instruction from memory and decode the instruction and update PC.

- The first 64 words are to be occupied by the routines for the 16 instructions.
- The last 64 words may be used for any other purpose.
  - A convenient starting location for the fetch routine is address 64.
- The three microinstructions that constitute the fetch routine have been listed in three different representations.
  - The register transfer representation:

```
AR ← PC
DR ← M[AR], PC ← PC + 1
AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0
```

  - The symbolic representation:

```
        ORG 64
FETCH:  PCTAR          U  JMP  NEXT
        READ, INCPC    U  JMP  NEXT
        DRTAR          U  MAP
```

  - The binary representation:

Binary address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

3.6 Symbolic Microprogram

- Control Storage: 128 20-bit words
- The first 64 words: Routines for the 16 machine instructions 0, 4, 8, …, 60 gives four words in control memory for each routine.
- The last 64 words: Used for other purpose (e.g., fetch routine and other subroutines)
- The execution of the third (MAP) microinstruction in the fetch routine results in a branch to address 0xxxx00, were xxxx are the four bits of the operation code. e.g. ADD is 0000
- In each routine we must provide microinstructions for evaluating the effective address and for executing the instruction.
- The indirect address mode is associated with all memory-reference instructions.
- A saving in the number of control memory words may be achieved if the microinstructions for the indirect address are stored as a subroutine.
- This subroutine, INDRCT, is located right after the fetch routine, as shown in Table 3-2.
- Mapping: OP-code XXXX into 0XXXX00, the first address for the 16 routines are 0(0 0000 00), 4(0 0001 00), 8, 12, 16, 20, ..., 60

- To see how the transfer and return from the indirect subroutine occurs:
  - MAP microinstruction caused a branch to address 0
  - The first microinstruction in the ADD routine calls subroutine INDRCT when *I*=1
  - The return address is stored in the subroutine register *SBR*.
  - The INDRCT subroutine has two microinstructions:

    INDRCT: READ U JMP NEXT

    DRTAR U RET
  - Therefore, the memory has to be accessed to get the effective address, which is then transferred to *AR*.
  - The execution of the ADD instruction is carried out by the microinstructions at addresses 1 and 2
  - The first microinstruction reads the operand from memory into DR.
  - The second microinstruction performs an add microoperation with the content of DR and AC and then jumps back to the beginning of the fetch routine.

Label	Microoperations	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

Table 3-2: Symbolic Microprogram for Control Memory (Partial)

Binary Microprogram

- The symbolic microprogram must be translated to binary either by means of an assembler program or by the user if the microprogram is simple.
- The equivalent binary form of the microprogram is listed in Table 7-3.
- Even though address 3 is not used, some binary value, e.g. all 0's, must be specified for each word in control memory.
- However, if some unforeseen error occurs, or if a noise signal sets CAR to the value of 3, it will be wise to jump to address 64.

Micro Routine	Address		Binary microinstruction					
	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

Table 3-3: Binary Microprogram for control memory (Partial)

Control Memory

- When a ROM is used for the control memory,the microprogram binary list provides the truth table for fabricating the unit.
    - To modify the instruction set of the computer, it is necessary to generate a new microprogram and mask a new ROM.
- The advantage of employing a RAM for the control memory is that the microprogram can be altered simply by writing a new pattern of 1's and 0's without resorting to hardware procedure.
- However, most microprogram systems use a ROM for the control memory because it is cheaper and faster than a RAM.

3.7 Control Unit Operation

Microoperations

- A computer executes a program consisting instructions. Each instruction is made up of shorter sub-cycles as fetch, indirect, execute cycle, interrupt.
- Performance of each cycle has a number of shorter operations called micro-operations.
- Called so because each step is very simple and does very little.
- Thus micro-operations are functional atomic operation of CPU.

- Hence events of any instruction cycle can be described as a sequence of micro-operations

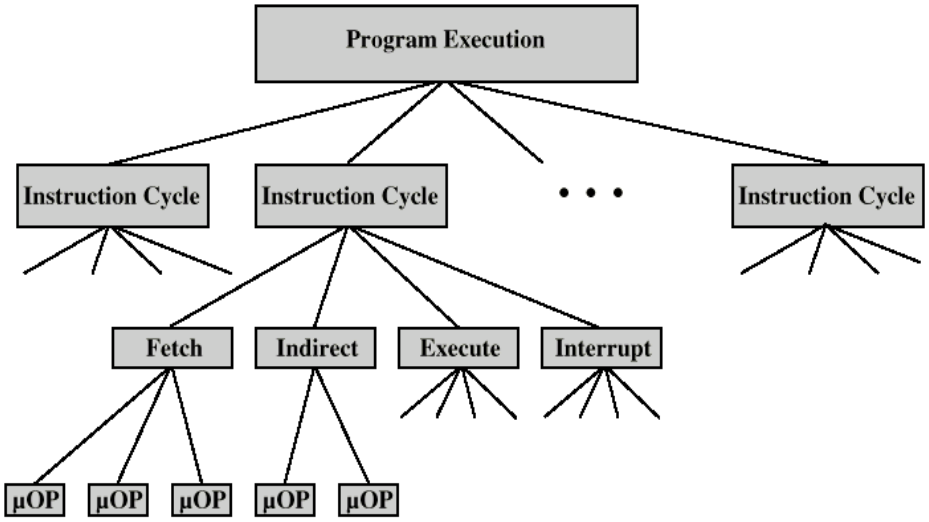


Fig 3-7: Constituent Elements of Program Execution

Steps leading to characterization of CU

- Define basic elements of processor
- Describe micro-operations processor performs
- Determine functions control unit must perform

Types of Micro-operation

- Transfer data between registers
- Transfer data from register to external interface
- Transfer data from external interface to register
- Perform arithmetic/logical ops with register for i/p, o/p

Functions of Control Unit

- Sequencing
- Causing the CPU to step through a series of micro-operations
- Execution
- Causing the performance of each micro-op

These are done using Control Signals

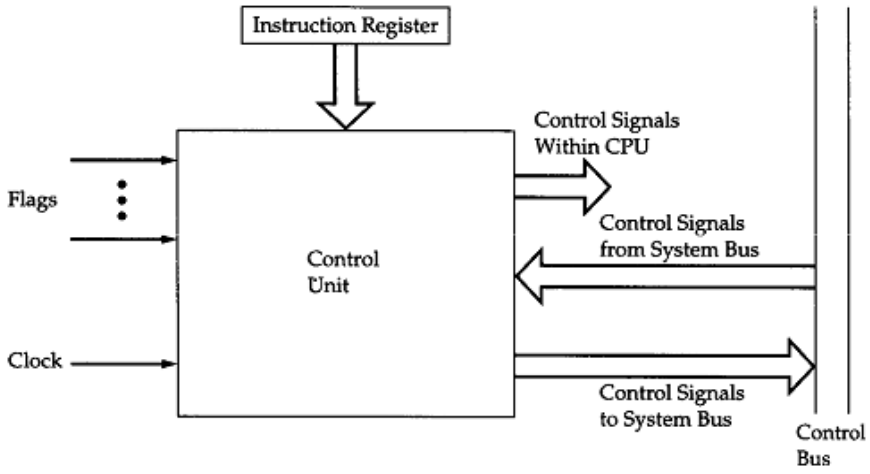


Fig 3-8: Control Unit Layout

Inputs to Control Unit

- Clock
  - CU causes one micro-instruction (or set of parallel micro-instructions) per clock cycle
- Instruction register
  - Op-code for current instruction determines which micro-instructions are performed
- Flags
  - State of CPU
  - Results of previous operations
- From control bus
  - Interrupts
  - Acknowledgements

CU Outputs (Control Signals)

- Within CPU(two types)
  - Cause data movement
  - Activate specific ALU functions
- Via control bus(two types)
  - To memory
  - To I/O modules
- Types of Control Signals
  - Those that activate an ALU
  - Those that activate a data path
  - Those that are signal on external system bus or other external interface.
- All these are applied as binary i/p to individual logic gates

Hardwired Implementation

- In this implementation, CU is essentially a combinational circuit. Its i/p signals are transformed into set of o/p logic signal which are control signals.
- Control unit inputs
- Flags and control bus
  - Each bit means something
- Instruction register
  - Op-code causes different control signals for each different instruction
  - Unique logic for each op-code
  - Decoder takes encoded input and produces single output
  - Each decoder i/p will activate a single unique o/p
- Clock
  - Repetitive sequence of pulses
  - Useful for measuring duration of micro-ops
  - Must be long enough to allow signal propagation along data paths and through processor circuitry
  - Different control signals at different times within instruction cycle
  - Need a counter as i/p to control unit with different control signals being used for t1, t2 etc.
  - At end of instruction cycle, counter is re-initialised

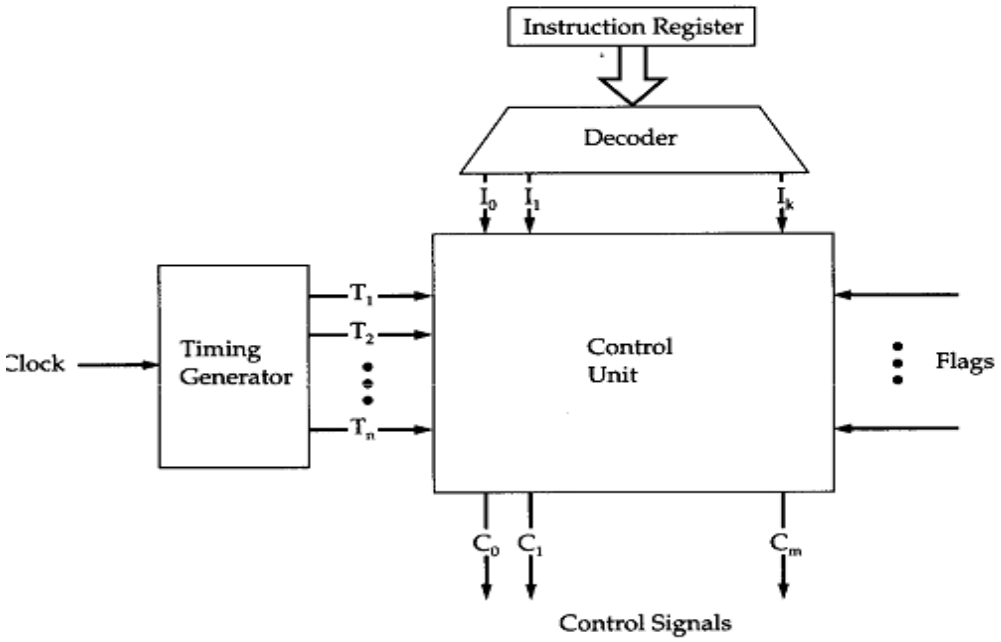


Fig 3-9: Control Unit With Decoded Input

Implementation

- For each control signal, a Boolean expression of that signal as a function of the inputs is derived
- With that the combinatorial circuit is realized as control unit.

Problems With Hard Wired Designs

- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Difficult to add new instructions

Micro-programmed Implementation

- An alternative to hardwired CU
- Common in contemporary CISC processors
- Use sequences of instructions to perform control operations performed by micro operations called micro-programming or firmware

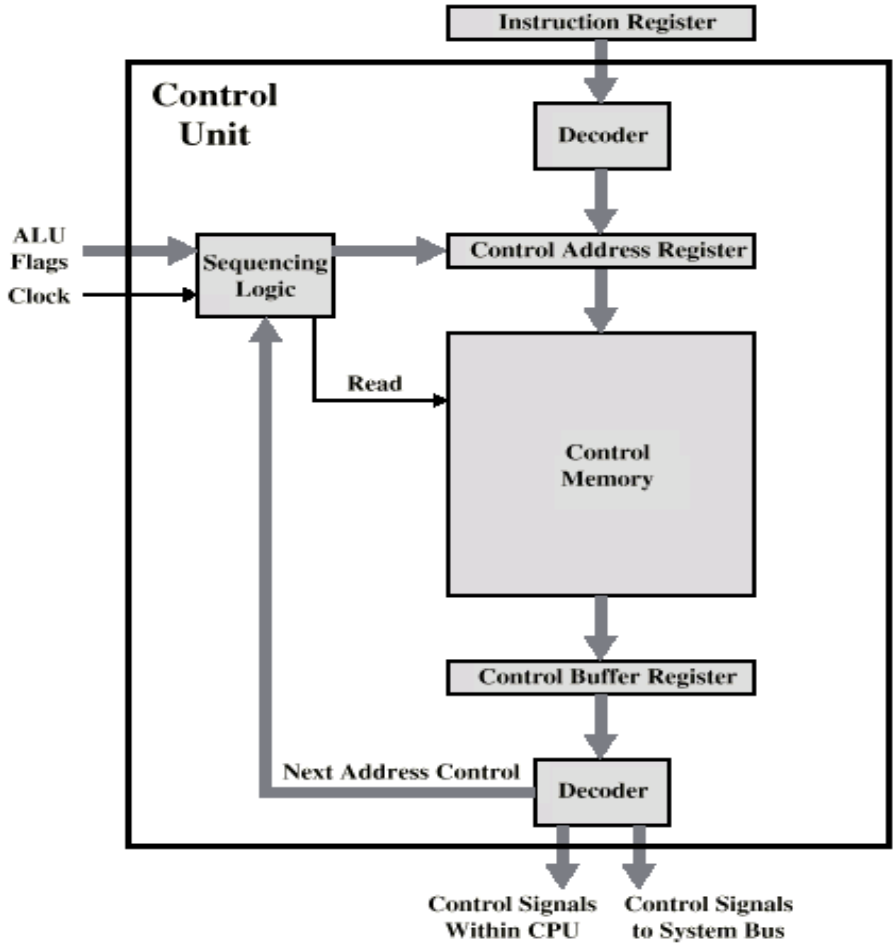


Fig 3-10: Microprogrammed Control Unit

- Set of microinsrurctions are stored in control memory
- Control address register contains the address of the next microinstruction to be read
- As it is read, it is transferred to control buffer register.
- For horizontal micro instructions, reading a microinstruction is same as executing it.
- Sequencing unit loads the control address register and issues a read command

CU functions as follows to execute an instruction:

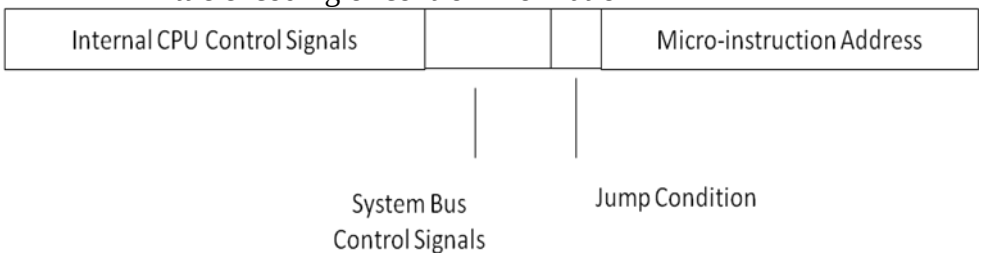
- Sequencing logic issues read command to control memory
- Word whose address is in control address register is read into control buffer register.
- Content of control buffer register generates control signals and next address instruction for the sequencing logic unit.
- Sequencing logic unit loads new address into control address register depending upon the value of ALU flags, control buffer register.
- One of following decision is made:
  - add 1 to control address register
  - load address from address field of control buffer register
  - load the control address register based on opcode in IR
- Upper decoder translates the opcode of IR into control memory address.
- Lower decoder used for veritcal micro instructions.

Micro-instruction Types

- Each micro-instruction specifies single or few micro-operations to be performed - *vertical* micro-programming
- Each micro-instruction specifies many different micro-operations to be performed in parallel - *horizontal* micro-programming

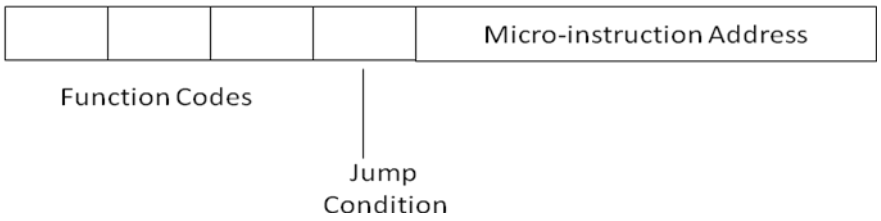
Horizontal Micro-programming

- Wide memory word
- High degree of parallel operations possible
- Little encoding of control information



Vertical Micro-programming

- Width is narrow
- n control signals encoded into $\log_2 n$ bits
- Limited ability to express parallelism
- Considerable encoding of control information requires external memory word decoder to identify the exact control line being manipulated



3.8 Design of Control Unit

- The bits of the microinstruction are usually divided into fields, with each field defining a distinct, separate function.
- The various fields encountered in instruction formats provide:
  - Control bits to initiate microoperations in the system
  - Special bits to specify the way that the next address is to be evaluated
  - An address field for branching
- The number of control bits that initiate microoperations can be reduced by grouping *mutually exclusive* variables into fields by encoding the *k* bits in each field to provide 2^k microoperations.
- Each field requires a decoder to produce the corresponding control signals.
  - Reduces the size of the microinstruction bits
  - Requires additional hardware external to the control memory
  - Increases the delay time of the control signals
- Fig. 3-11 shows the three decoders and some of the connections that must be made from their outputs.
- Outputs 5 or 6 of decoder F1 are connected to the load input of AR so that when either one of these outputs is active; information from the multiplexers is transferred to AR.
- The transfer into *AR* occurs with a clock pulse transition only when output 5 (from DR (0-10) to AR i.e. DRTAR) or output 6 (from PC to AR i.e. PCTAR) of the decoder are active.
- The arithmetic logic shift unit can be designed instead of using gates to generate the control signals; it comes from the outputs of the decoders.

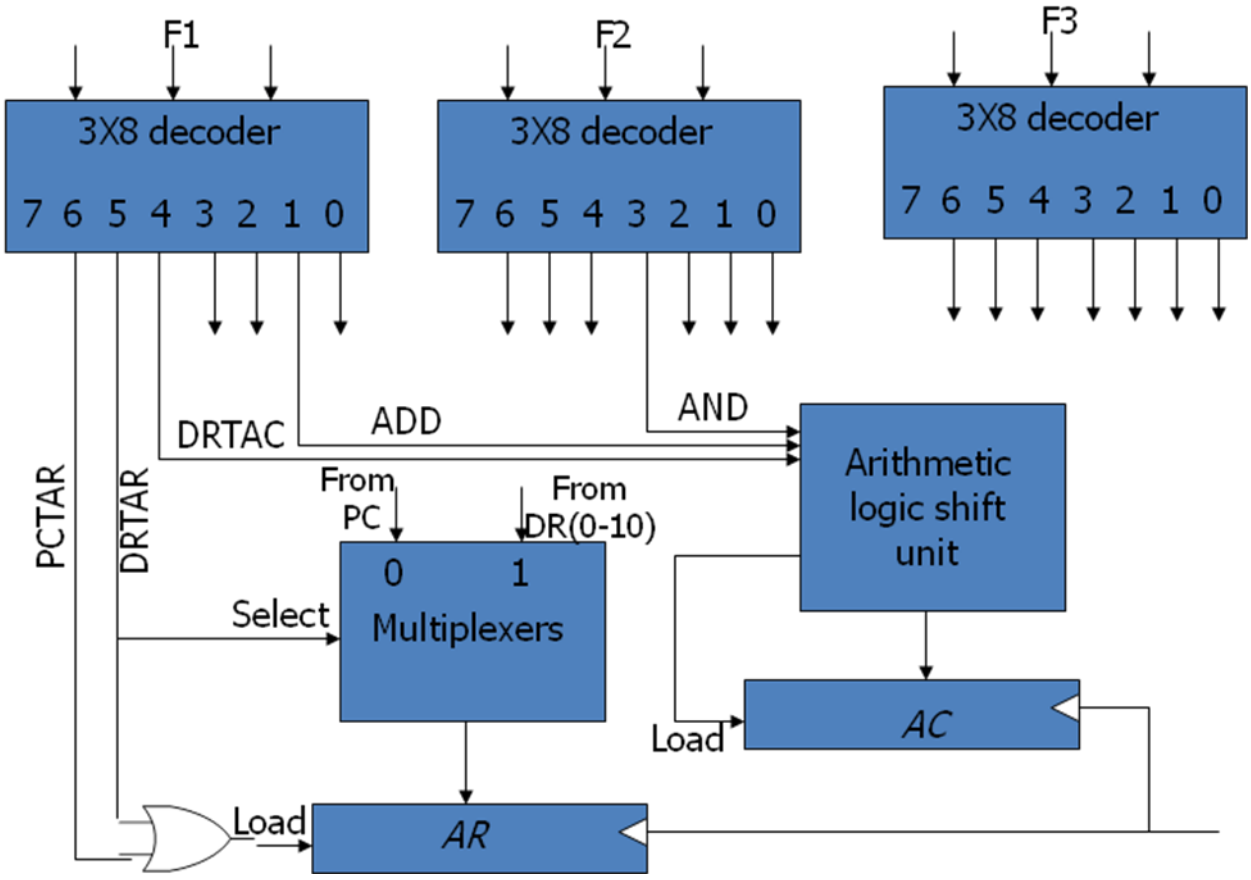


Fig 3-11: Decoding of Microoperation fields

Microprogram Sequencer

- The basic components of a microprogrammed control unit are the *control memory* and *the circuits that select the next address*.
- The address selection part is called a *microprogram sequencer*.
- A microprogram sequencer can be constructed with *digital functions* to suit a particular application.
- To guarantee a wide range of acceptability, an *integrated circuit sequencer* must provide an internal organization that can be adapted to a wide range of application.
- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- The block diagram of the microprogram sequencer is shown in Fig. 3-12.
    - The control memory is included to show the interaction between the sequencer and the attached to it.
    - There are two multiplexers in the circuit; first multiplexer selects an address from one of the four sources and routes to CAR, second multiplexer tests the value of the selected status bit and result is applied to an input logic circuit.
    - The output from CAR provides the address for control memory, contents of CAR incremented and applied to one of the multiplexers input and to the SBR.

- Although the diagram shows a *single subroutine register*, a typical sequencer will have a *register stack* about four to eight levels deep. In this way, a push, pop operation and stack pointer operates for subroutine call and return instructions.
- The CD (Condition) field of the microinstruction selects one of the status bits in the second multiplexer.
- The Test variable (either 1 or 0) i.e. T value together with the two bits from the BR (Branch) field go to an input logic circuit.
- The input logic circuit determines the type of the operation.

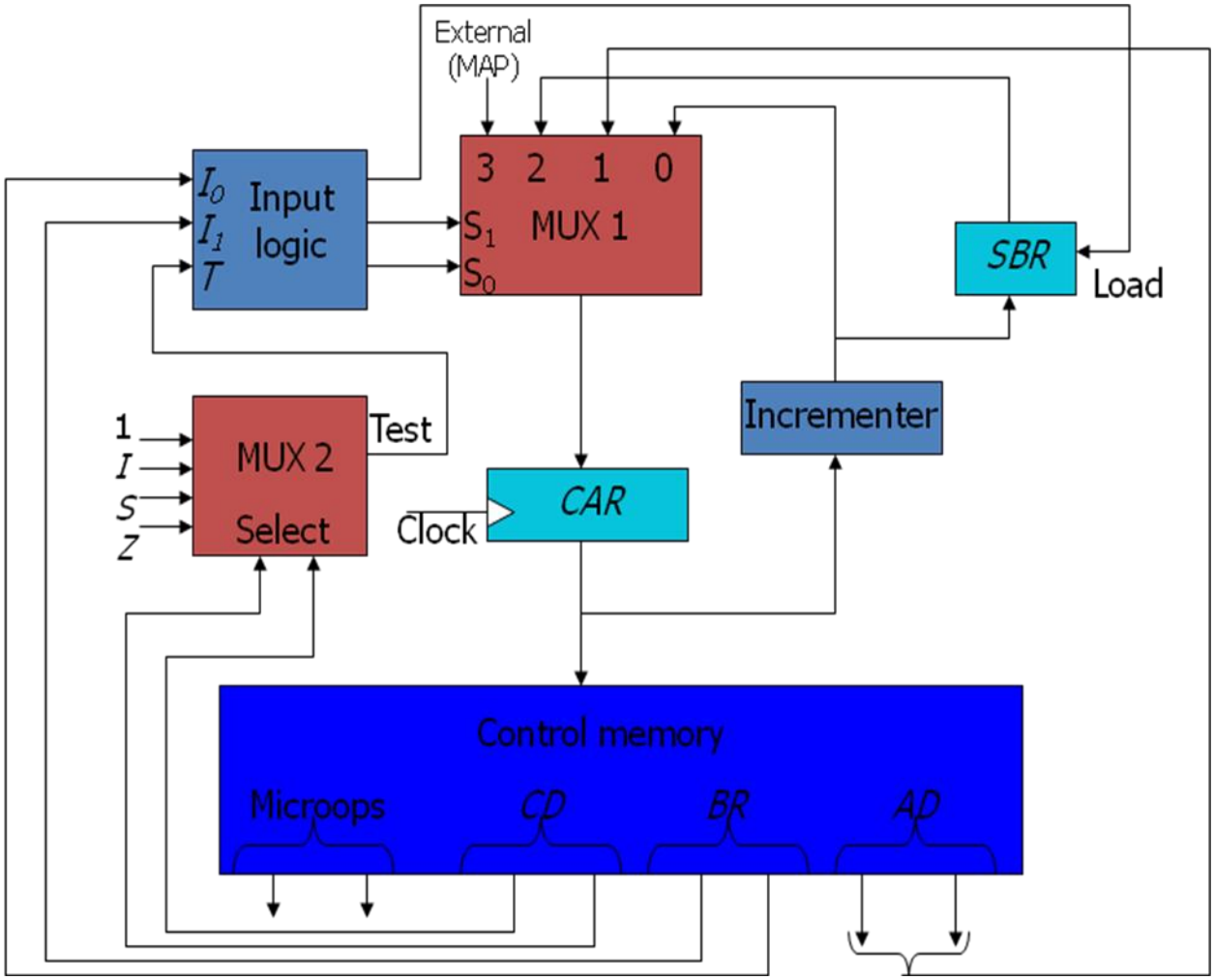


Fig 3-12: Microprom Sequencer for a Control Memory

Design of Input Logic

- The input logic in a particular sequencer will determine the type of operations that are available in the unit.
- Typical sequencer operations are: *increment, branch or jump, call and return from subroutine, load an external address, push or pop the stack, and other address sequencing operations*.

- Based on the function listed in each entry was defined in Table 3-1, the truth table for the input logic circuit is shown in Table 3-4.
- Therefore, the simplified Boolean functions for the input logic circuit can be given as:

$S_1 = I_1$

$S_0 = I_1 I_0 + I_1' T$

$L = I_1' I_0 T$

BR Field	Input I_1	I_0	T	MUX 1 S_1	S_0	Load *SBR* L
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	0	1	1
1	0	1	0	X	1	0
1	1	1	1	X	1	0

Table 3-4: Input logic truth table for microprogram sequencer

  - The bit values for S1 and S0 are determined from the stated function and the path in the multiplexer that establishes the required transfer.
  - Note that the incrementer circuit in the sequencer of Fig. 7-12 is not a counter constructed with flip-flops but rather a combinational circuit constructed with gates.

KARPAGAM ACADEMY OF HIGHER EDUCATION
COIMBATORE-21
Faculty of Engineering
Department of Computer Science and Engineering

POSSIBLE QUESTIONS

Title of the paper : COA

Answer All the Questions (5*2=18)

1. Define pipelining.
2. What is a Cache memory?
3. Define parallel processing
4. What is memory interleaving?
5. Define SSID.
6. What are the characteristics of memory systems?
7. Define Exceptions.
8. What is a data hazard?
9. Define memory latency.

ANSWER all the Questions(3*14=42)

10. a. What is branch hazard? Describe the method for dealing with the branch hazard? (14)

(OR)

b. Draw and explain data path modified for pipelined execution. (14)

11. a. Explain Discuss any six ways of improving the cache performance. (14)

(OR)

b. What do you mean by virtual memory? Discuss how paging helps in implementing virtual memory. (14)

12.a) Discuss the different mapping techniques used in cache memories and their relative merits and demerits. (14)

(OR)

b). What is Memory Interleaving? Explain the addressing of multiple modules Memory system. (14)

MULTIPLE CHOICE QUESTIONS - COA

1. In Reverse Polish notation, expression A*B+C*D is written as
(A) AB*CD*+ (B) A*BCD*+ (C) AB*CD+* (D) A*B*CD+
Ans: A

2. SIMD represents an organization that _____.
(A) refers to a computer system capable of processing several programs at the same time.
(B) represents organization of single computer containing a control unit, processor unit and a memory unit.
(C) includes many processing units under the supervision of a common control unit
(D) none of the above.
Ans: C

3. Floating point representation is used to store
(A) Boolean values (B) whole numbers (C) real integers (D) integers
Ans: C

4. Suppose that a bus has 16 data lines and requires 4 cycles of 250 nsecs each to transfer data. The bandwidth of this bus would be 2 Megabytes/sec. If the cycle time of the bus was reduced to 125 nsecs and the number of cycles required for transfer stayed the same what would the bandwidth of the bus?
(A) 1 Megabyte/sec (B) 4 Megabytes/sec (C) 8 Megabytes/sec (D) 2 Megabytes/sec
Ans: D

5. Assembly language
(A) uses alphabetic codes in place of binary numbers used in machine language
(B) is the easiest language to write programs
(C) need not be translated into machine language
(D) None of these
Ans: A

6. In computers, subtraction is generally carried out by
(A) 9's complement (B) 10's complement (C) 1's complement (D) 2's complement **Ans:** D

7. The amount of time required to read a block of data from a disk into memory is composed of seek time, rotational latency, and transfer time. Rotational latency refers to (A) the time its takes for the platter to make a full rotation
(B) the time it takes for the read-write head to move into position over the appropriate track
(C) the time it takes for the platter to rotate the correct sector under the head
(D) none of the above **Ans:** A

8. What characteristic of RAM memory makes it not suitable for permanent storage?
(A) too slow (B) unreliable (C) it is volatile (D) too bulky **Ans:** C

9. Computers use addressing mode techniques for _____.
(A) giving programming versatility to the user by providing facilities as pointers to memory counters for loop control
(B) to reduce no. of bits in the field of instruction
(C) specifying rules for modifying or interpreting address field of the instruction
(D) All the above
Ans: D

10. The circuit used to store one bit of data is known as
(A) Register (B) Encoder (C) Decoder (D) Flip Flop
Ans: D

11. (2FAOC) 16 is equivalent to
(A) (195 084) 10 (B) (001011111010 0000 1100) 2 (C) Both (A) and (B) (D) None of these
Ans: B

12. The average time required to reach a storage location in memory and obtain its contents is called the
(A) seek time (B) turnaround time (C) access time (D) transfer time
Ans: C

13. Which of the following is not a weighted code?
(A) Decimal Number system (B) Excess 3-cod
(C) Binary number System (D) None of these
Ans: B

14. The idea of cache memory is based
(A) on the property of locality of reference (B) on the heuristic 90-10 rule
(C) on the fact that references generally tend to cluster (D) all of the above
Ans: A

15. Which of the following is lowest in memory hierarchy? (A) Cache memory (B) Secondary memory (C) Registers (D) RAM (E) None of these
Ans (B)

16. The addressing mode used in an instruction of the form ADD X Y, is
(A) Absolute (B) indirect (C) index (D) none of these
Ans: C

17. If memory access takes 20 ns with cache and 110 ns with out it, then the ratio (cache uses a 10 ns memory) is
(A) 93% (B) 90% (C) 88% (D) 87%
Ans: B

18. In a memory-mapped I/O system, which of the following will not be there?
(A) LDA (B) IN (C) ADD (D) OUT

Ans: A

19. In a vectored interrupt.
(A) the branch address is assigned to a fixed location in memory.
(B) the interrupting source supplies the branch information to the processor through an interrupt vector.
(C) the branch address is obtained from a register in the processor
(D) none of the above
Ans: B

20. Von Neumann architecture is
(A) SISD (B) SIMD (C) MIMD (D) MISD
Ans: A

21. The circuit used to store one bit of data is known as
(A) Encoder (B) OR gate (C) Flip Flop (D) Decoder
Ans: C

22. Cache memory acts between
(A) CPU and RAM (B) RAM and ROM (C) CPU and Hard Disk (D) None of these
Ans: A

23. Write Through technique is used in which memory for updating the data
(A) Virtual memory (B) Main memory
(C) Auxiliary memory (D) Cache memory
Ans: D

24. Generally Dynamic RAM is used as main memory in a computer system as it
(A) Consumes less power (B) has higher speed
(C) has lower cell density (D) needs refreshing circuitary
Ans: B

25. In signed-magnitude binary division, if the dividend is (11100) 2 and divisor is (10011) 2 then the result is
(A) (00100) 2 (B) (10100) 2 (C) (11001) 2 (D) (01100) 2
Ans: B

26. Virtual memory consists of
(A) Static RAM (B) Dynamic RAM
(C) Magnetic memory (D) None of these
Ans: A

27. In a program using subroutine call instruction, it is necessary
(A) initialise program counter (B) Clear the accumulator
(C) Reset the microprocessor (D) Clear the instruction register
Ans: D

28. A Stack-organised Computer uses instruction of
(A) Indirect addressing (B) Two-addressing (C) Zero addressing (D) Index addressing
Ans: C

29. If the main memory is of 8K bytes and the cache memory is of 2K words. It uses associative mapping. Then each word of cache memory shall be
(A) 11 bits (B) 21 bits (C) 16 bits (D) 20 bits
Ans: C

30 A-Flip Flop can be converted into T-Flip Flop by using additional logic circuit
(A) n TQD =• (B) T D = (C) D = T . Q n (D) n TQD =⊕
Ans: D

31. Logic X-OR operation of (4ACO) H & (B53F) H results
(A) AACB (B) 0000 (C) FFFF (D) ABCD
Ans: C

32. When CPU is executing a Program that is part of the Operating System, it is said to be in (A) Interrupt mode (B) System mode (C) Half mode (D) Simplex mode
Ans: B

33. An n-bit microprocessor has
(A) n-bit program counter (B) n-bit address register
(C) n-bit ALU (D) n-bit instruction register
Ans: D

34. Cache memory works on the principle of
(A) Locality of data (B) Locality of memory
(C) Locality of reference (D) Locality of reference & memory
Ans: C

35. The main memory in a Personal Computer (PC) is made of
(A) cache memory. (B) static RAM
(C) Dynamic Ram (D) both (A) and (B) .
Ans: D

36. In computers, subtraction is carried out generally by
(A) 1's complement method (B) 2's complement method
(C) signed magnitude method (D) BCD subtraction method
Ans: B

37. PSW is saved in stack when there is a
(A) interrupt recognised (B) execution of RST instruction
(C) Execution of CALL instruction (D) All of these
Ans: A

38. The multiplicand register & multiplier register of a hardware circuit implementing booth's algorithm have (11101) & (1100). The result shall be
(A) (812) 10 (B) (-12) 10 (C) (12) 10 (D) (-812) 10
Ans: A

39. The circuit converting binary data in to decimal is
(A) Encoder (B) Multiplexer (C) Decoder (D) Code converter
Ans: D

40. A three input NOR gate gives logic high output only when
(A) one input is high (B) one input is low
(C) two input are low (D) all input are high
Ans: D

41. n bits in operation code imply that there are _____ possible distinct operators (A) 2n (B) 2n (C) n/2 (D) n2
Ans: B

42. _____ register keeps tracks of the instructions stored in program stored in memory.
(A) AR (Address Register) (B) XR (Index Register)
(C) PC (Program Counter) (D) AC (Accumulator)
Ans: C

43. Memory unit accessed by content is called
(A) Read only memory (B) Programmable Memory
(C) Virtual Memory (D) Associative Memory
Ans: D

44. 'Aging registers' are
(A) Counters which indicate how long ago their associated pages have been referenced.
(B) Registers which keep track of when the program was last accessed.
(C) Counters to keep track of last accessed instruction.
(D) Counters to keep track of the latest data structures referred.
Ans: A

45 The instruction 'ORG O' is a
(A) Machine Instruction. (B) Pseudo instruction.
(C) High level instruction. (D) Memory instruction.
Ans: B

46 Translation from symbolic program into Binary is done in
(A) Two passes. (B) Directly (C) Three passes. (D) Four passes.
Ans: A

47 A floating point number that has a O in the MSB of mantissa is said to have

(A) Overflow (B) Underflow (C) Important number (D) Undefined
Ans: B

48 The BSA instruction is
(A) Branch and store accumulator (B) Branch and save return address
(C) Branch and shift address (D) Branch and show accumulator
Ans: B

49 State whether True or False.
(i) Arithmetic operations with fixed point numbers take longer time for execution as compared to with floating point numbers.
Ans: True.

(ii) An arithmetic shift left multiplies a signed binary number by 2. **Ans: False.**

50 Logic gates with a set of input and outputs is arrangement of
(A) Combinational circuit (B) Logic circuit (C) Design circuits (D) Register
Ans: A

51. MIMD stands for
(A) Multiple instruction multiple data (B) Multiple instruction memory data
(C) Memory instruction multiple data (D) Multiple information memory data
Ans: A

52 A k-bit field can specify any one of
(A) 3k registers (B) 2k registers
(C) K2 registers (D) K3 registers
Ans: B

53 The time interval between adjacent bits is called the
(A) Word-time (B) Bit-time (C) Turn around time (D) Slice time
Ans: B

54 A group of bits that tell the computer to perform a specific operation is known as
(A) Instruction code (B) Micro-operation (C) Accumulator (D) Register
Ans: A

55 The load instruction is mostly used to designate a transfer from memory to a processor register known as
(A) Accumulator (B) Instruction Register
(C) Program counter (D) Memory address Register
Ans: A

56 The communication between the components in a microcomputer takes place via the address and
(A) I/O bus (B) Data bus (C) Address bus (D) Control lines

Ans: B

57 An instruction pipeline can be implemented by means of
(A) LIFO buffer (B) FIFO buffer (C) Stack (D) None of the above
Ans: B

58 Data input command is just the opposite of a
(A) Test command (B) Control command (C) Data output (D) Data channel
Ans: C

59 A microprogram sequencer
(A) generates the address of next micro instruction to be executed.
(B) generates the control signals to execute a microinstruction.
(C) sequentially averages all microinstructions in the control memory.
(D) enables the efficient handling of a micro program subroutine.
Ans: A

60 . A binary digit is called a
(A) Bit (B) Byte (C) Number (D) Character
Ans: A

61 A flip-flop is a binary cell capable of storing information of
(A) One bit (B) Byte (C) Zero bit (D) Eight bit
Ans: A

62 The operation executed on data stored in registers is called
(A) Macro-operation (B) Micro-operation
(C) Bit-operation (D) Byte-operation
Ans: B

63 MRI indicates
(A) Memory Reference Information. (B) Memory Reference Instruction.
(C) Memory Registers Instruction. (D) Memory Register information
Ans: B

64 Self-contained sequence of instructions that performs a given computational task is called
(A) Function (B) Procedure (C) Subroutine (D) Routine
Ans: A

65 Microinstructions are stored in control memory groups, with each group specifying a (A) Routine
(B) Subroutine (C) Vector (D) Address
Ans: A

66 An interface that provides a method for transferring binary information between internal storage and external devices is called
(A) I/O interface (B) Input interface (C) Output interface (D) I/O bus

Ans: A

67 Status bit is also called
(A) Binary bit (B) Flag bit (C) Signed bit (D) Unsigned bit
Ans: B

68 An address in main memory is called
(A) Physical address (B) Logical address (C) Memory address (D) Word address
Ans: A

69 If the value V(x) of the target operand is contained in the address field itself, the addressing mode is
(A) immediate. (B) direct. (C) indirect. (D) implied.
Ans: B

70 can be represented in a signed magnitude format and in a 1's complement format as (A) 111011 & 100100 (B) 100100 & 111011
(C) 011011 & 100100 (D) 100100 & 011011
Ans: A