CHAPTER 1: INTRODUCTION

Why do we need to study computer organization and/or architecture ?

The computer lies to the heart of computing without it most of the computing disciplines today would be branch of theorititcal mathematics. To be a professional in any field of computing today, one should not regard the computer as just a black box that executes programs by magic. All students of computing should acquire some understanding and appereciation of a computer system's fuctional componensts, their characteristcis ,their performance and their interaction. Students needs to understand computer archtecture in order to structure a program so that it runs more efficiently on real machines.
Example: we need 6 bit system and there is avaible and there is available of 2bits,4 bits or 8 bits, now one should know which will be more beneficial using 2,2,2 bits bits or 4,4 bits system or 8 bit system and design the system according.

Choosing the best alternative for right organization.
Concept used in computer organization & computer architecture can be used o other courses fields.

Computer architecture:

- Computer architecture refers to those attributes of a system visible to a programmer or that have direct impact on the logical execution of a program.
- Computer architecture is the study of the structure, behavior and design of a computer system.

Examples of architectural attributes include:

- Instruction set.
- Number of bits used to represent various data types.
- I/O mechanism and techniques for addressing memory.

Computer Organization:

Computer organization refers to the operational units and their interconnection that realize the architectural specification.

- Organization attributes include those hardware details transparent to the programmer, such as control signals, interfaces, technologies used.

For example, it is an architectural design issue whether a computer will have multiple instruction or not, it is an organizational issue whether that instruction will be implemented by a special multiply unit or by a mechanism of repeated addition. The organization decision may be based on the anticipated frequency of use of multiply instruction, the relative speed of the approaches and the cost of the physical size of special multiply unit. The computer architecture of a computer system may be same with different organization.

History of computer/ Miles stones in computer organization

Mechanical era (1623-1945):

- The idea of using machines to solve mathematical problems can be traced back to 17th century where mathematician designed and implemented calculators that were capable of addition, subtraction, multiplication & division included Wilhelm Schickhard, Blaise Pascal, Gottfried Leibnitz.

- The first multi-purpose programmable computing device was Charles Babbage's difference and analytical engine.

- First commercial use of mechanical computers was in US census Bureau by Herman Hollerith.

Two major drawbacks of mechanical computers are :

1. Speed of operation limited by the inertia of moving parts.
2. Cumbersome unreliable and expensive.

Electronic era

First Generation (1945-1956)

- Vacuum tubes for circuitry, magnetic drum for memory.

- Enormous high power.

- ENIAC(Enhanced Numeric Integrator and Computer)

Second generation (1956-1963)

- Transistors.

- Smaller, faster, cheap, more energy efficient more reliable.

- More complex arithmetic and logic unit, control units.

Third generation (1964-1971)

- Integrated circuits, SSI (Small Scale Integration)-1 million transistors.

- LSI (Large Scale Integration)-10 million transistors.

- Third generation computer through keyboards and monitors and interface with an operating system, which allowed the device to run many different applications at one time with central program that monitored the memory.

Fourth Generation (1971-present)

- Used micro-processor, fourth generation of computers as thousands of integrated circuits were built on a single silicon chip.
- Intel 4004 chip developed in 1971 located all the components of the computer from CPU to memory to I/O controls on a single chip.
- IBM introduced first computer, 1984 apple macintosh

Fifth Generation (Present)

- Based on artificial intelligence, still development phases.
- Voice recognition, self-decision, nano technology, self-organization.

Moore's law: Moore observed that the number of transistors that would be put on a single chip was doubling every year and was corrected many years. This pace slowed to a doubling every 18 months in 1970.

Von neuman machine/IAS computer:

It was designed by the Princeton institute for advanced studies (IAS)

It consists of:

- Main memory: which stores both data and instruction.
- Arithmetic and logic unit: capable of operating on binary data.
- A control unit which interprets the instruction in memory and causes them to be executed.
- Input & Output equipment operated by control unit.

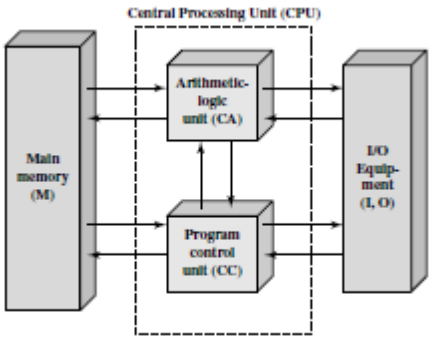


Figure 2.1 Structure of the IAS Computer

Memory location

- 1000 storage locations called words of 40 binary digits (bits).
- Both data and instruction stored.
- All numbers and data represents in binary form.
- Each number represented by 1 sign bit, 39 bits value or two 20 bits instruction consisting of 8 bits opcode, 12 bits addressing address.

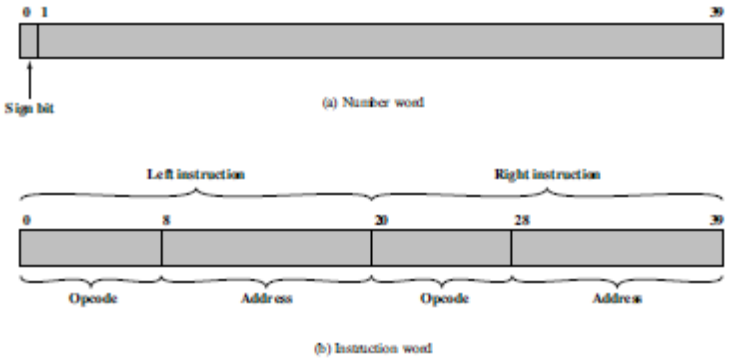


Figure 2.2 IAS Memory Formats

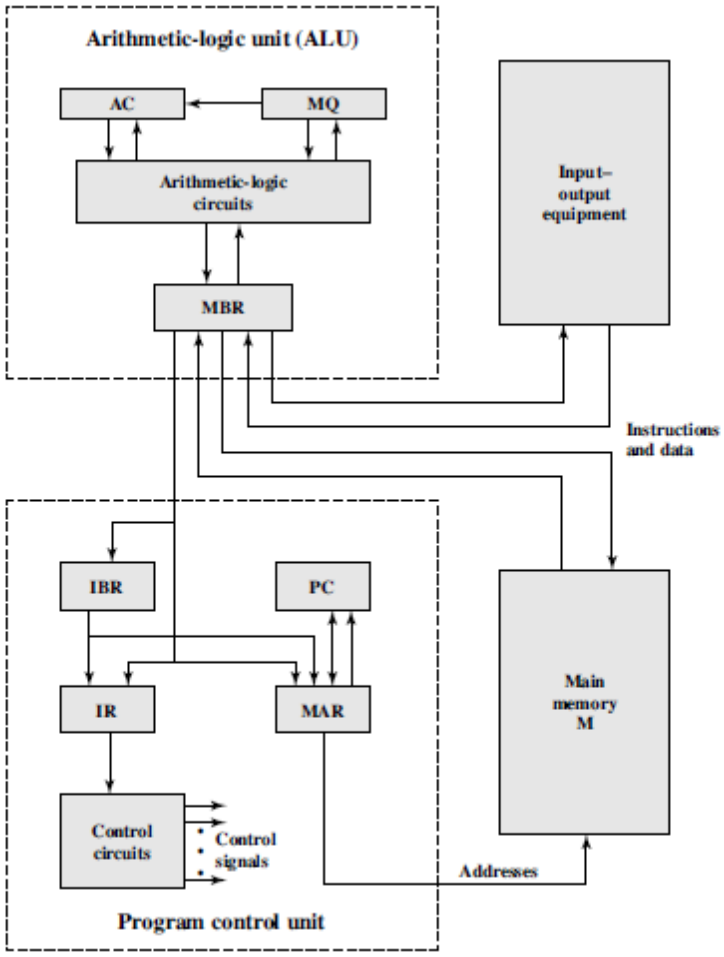


Figure 2.3 Expanded Structure of IAS Computer

Different registers

Memory buffer register (MBR): Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.

Memory address register (MAR): Specifies the address in memory of the word to be written from or read into the MBR.

Instruction register (IR): Contains the 8-bit opcode instruction being executed.

Instruction buffer register (IBR): Employed to hold temporarily the right hand instruction from a word in memory.

Program counter (PC): Contains the address of the next instruction-pair to be fetched from memory.

Accumulator (AC) and multiplier quotient (MQ): Employed to hold temporarily operands and results of ALU operations.

Operation

IAS operates by repetitively performing an instruction cycle. Instruction cycle contains fetch and execute cycle.

Fetch cycle

- The opcode of the next instruction is loaded into IR and addr. portion into MAR.
- The instruction can be taken from IBR from MBR and then to IBR, IR and MAR.

Execute cycle

- Control circuitry interprets the opcode and executes the instruction by sending out the appropriate control signal to cause data to be moved or perform ALU operations.

The total of 21 instructions that can be grouped onto:

1. **Data transfer:** between memory and/ OR ALU registers.

2. **Unconditional branch:** Normally, the control unit executes instructions in sequence from memory. This sequence can be changed by a branch instruction, which facilitates repetitive operations.

3. **Conditional branch:** The branch can be made dependent on a condition, thus allowing decision points.

4. **Arithmetic:** Operations performed by the ALU.

5. **Address modify:** Permits addresses to be computed in the ALU and then inserted into instructions stored in memory. This allows a program considerable addressing flexibility.

Table 2.1 The IAS Instruction Set

Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD −M(X)	Transfer −M(X) to the accumulator
	00000011	LOAD |M(X)|	Transfer absolute value of M(X) to the accumulator
	00000100	LOAD −|M(X)|	Transfer −|M(X)| to the accumulator
Unconditional branch	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
Conditional branch	00001111	JUMP+ M(X,0:19)	If number in the accumulator is nonnegative, take next instruction from left half of M(X)
	00010000	JUMP+ M(X,20:39)	If number in the accumulator is nonnegative, take next instruction from right half of M(X)
Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD |M(X)|	Add |M(X)| to AC; put the result in AC
	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB |M(X)|	Subtract |M(X)| from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2; i.e., shift left one bit position
	00010101	RSH	Divide accumulator by 2; i.e., shift right one position
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC

Table: IAS instruction set

** see intel/Pentium generations table from william stalling book Computer Organization and

Architecture Designing for Performance

Operations

-IAS operates by repetitively performing an instruction cycle.
-instruction cycle- fetch and execute cycle

Fetch cycle

- op-code of the next instruction is loaded into IR & address portion into MAR.
- The instruction can be taken from IBR or from MBR and then to IBR, IR and MAR

Execute cycle

- Control circuitry interprets the op-code and executes the instruction by sending out the appropriate control signals to cause data to be moved or perform ALU operations.

The total of 21 instruction can be grouped into:

1. data transfer: between memory and OR ALU register.
2. Conditional branch
3. Un-conditional branch
4. Arithmetic
5. Address modify: permits address to be computed in the ALU and then inserted into instruction in memory

Partial flowchart of IAS operation from William Stalling book.

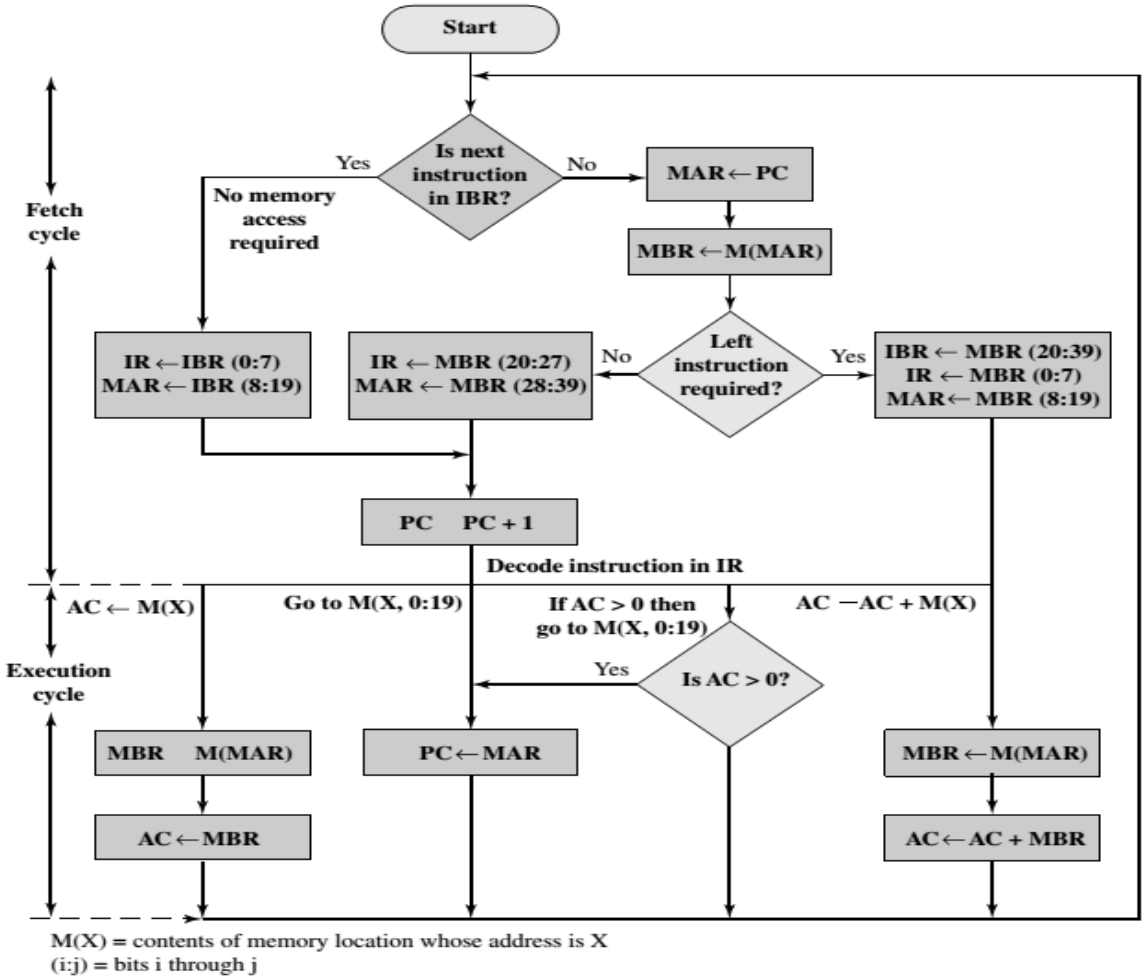


Figure 2.4 Partial Flowchart of IAS Operation

Functional view of computer

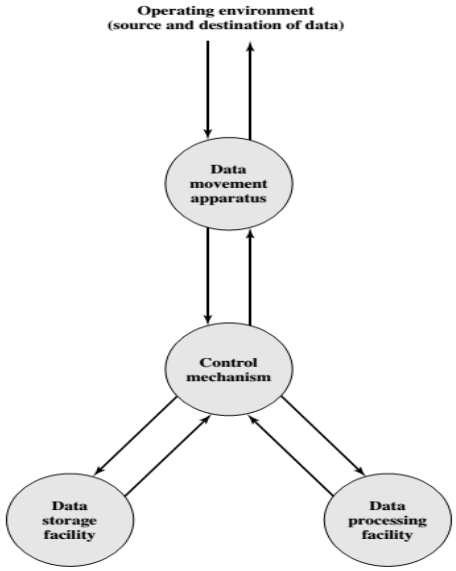


Figure 1.1 A Functional View of the Computer

Functions

- Data processing
- Data storage
- Data movement
- Control

Instruction Fetch and Execute

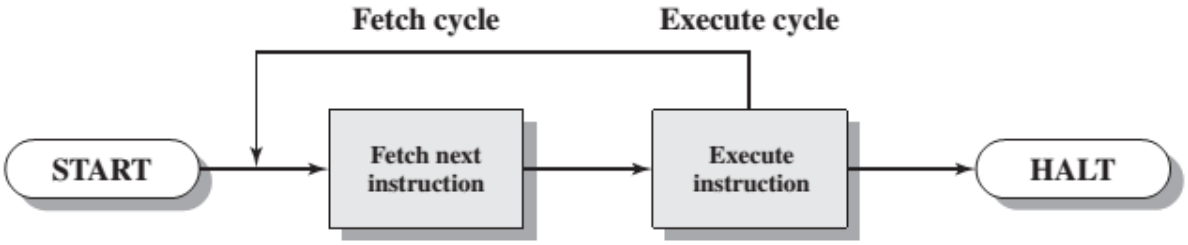


Figure 3.3 Basic Instruction Cycle

At the beginning of each instruction cycle, the processor fetches an instruction from memory. In a typical processor, a register called the program counter (PC) holds the address of the instruction to be fetched next. Unless told otherwise, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence. So, for example, consider a computer in which each instruction occupies one 16-bit word of memory. Assume that the program counter is set to location 300. The processor will next fetch the instruction at location 300. On succeeding instruction cycles, it will fetch instructions from locations 301, 302, 303, and so on. This sequence can also be altered as required. The fetched instruction is loaded into a register in the processor known as the instruction register (IR). The processor interprets the instruction and performs the required action. In general, these actions fall into four categories:

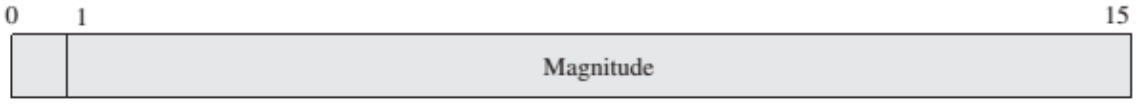
Processor-I/O: Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.

Data processing: The processor may perform some arithmetic or logical operations on data.

Control: An instruction may specify that the sequence of execution be altered.

Example of fetch and execute cycle

(a) Instruction format



(b) Integer format

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

(d) Partial list of opcodes

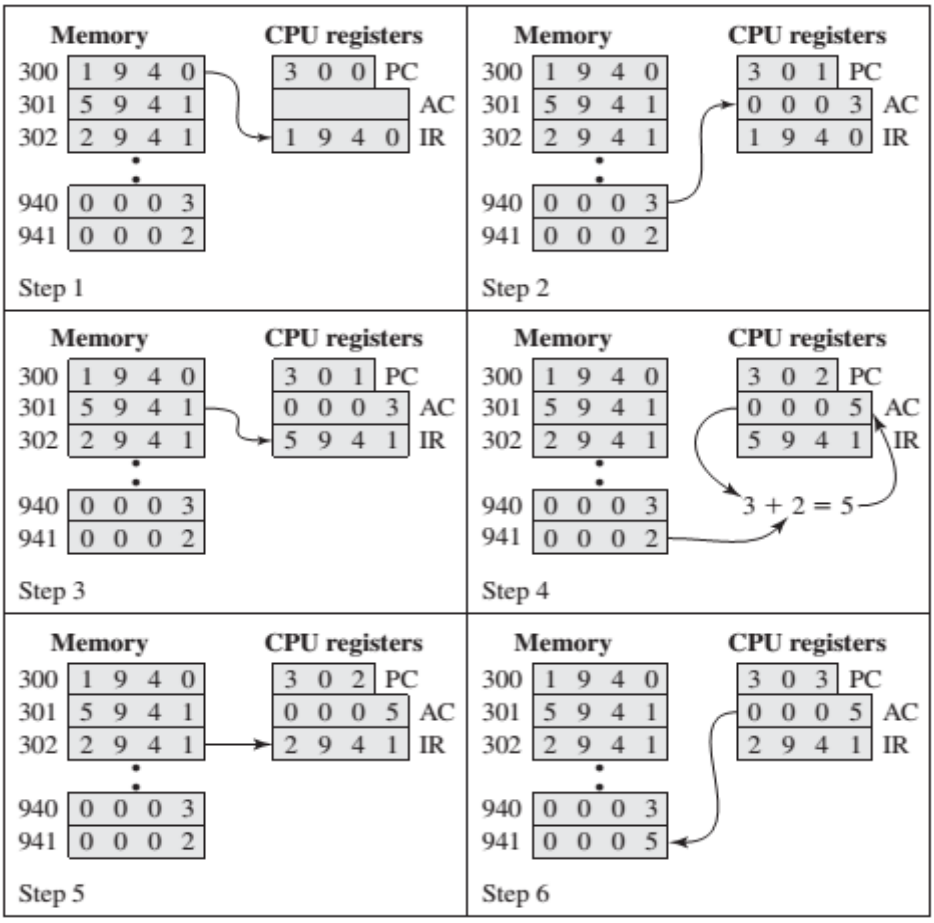


Figure 3.5 Example of Program Execution (contents of memory and registers in hexadecimal)

Figure 3.5: Example of Program Execution (contents of memory and registers in hexadecimal) Figure 3.5 illustrates a partial program execution, showing the relevant portions of memory and processor registers. The program fragment shown adds the contents of the
memory word at address 940 to the contents of the memory word at address 941 and stores the result in the latter location. Three instructions, which can be described as three fetch and three execute cycles, are required:

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the instruction register IR and the PC is incremented. Note that this process involves the use of a memory address register (MAR) and a memory buffer register (MBR). For simplicity, these intermediate registers are ignored.
2. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is to be loaded. The remaining 12 bits (three hexadecimal digits) specify the address (940) from which data are to be loaded.
3. The next instruction (5941) is fetched from location 301 and the PC is incremented.
4. The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.

5. The next instruction (2941) is fetched from location 302 and the PC is incremented.
6. The contents of the AC are stored in location 941.

Instruction cycle state diagram:

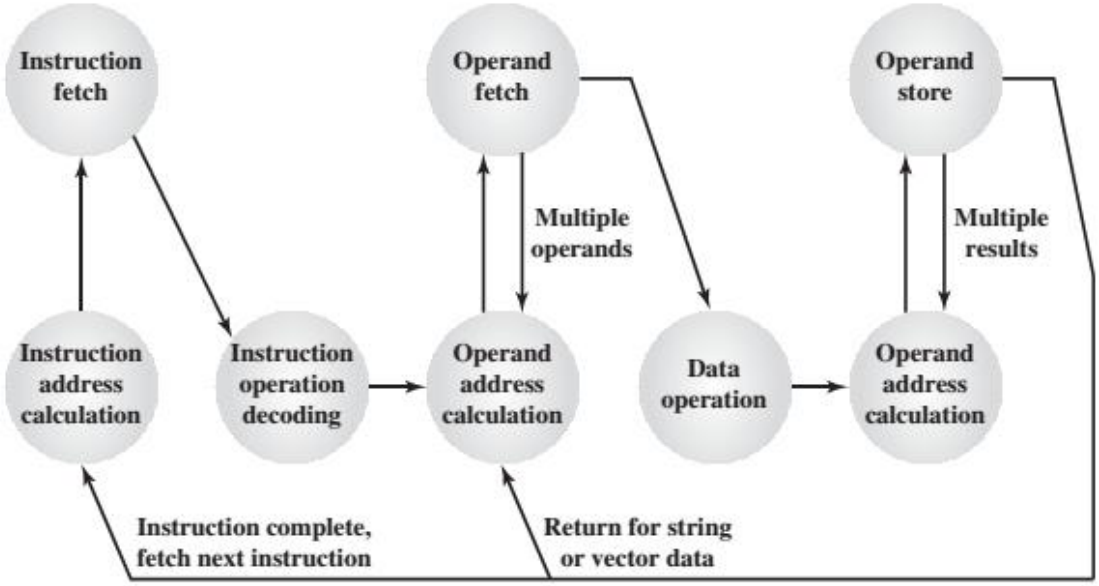


Figure 3.6 Instruction Cycle State Diagram

The above figure shows the sate diagram of instruction cycle, for any given instruction cycle. Some states may be null and other may be visited more than once. The different states can be described as:

- ***Instruction address calculation (iac):*** Determine the address of the next instruction to be executed. Usually, this involves adding a fixed number to the address of the previous instruction. For example, if each instruction is 16 bits long and memory is organized into 16-bit words, then add 1 to the previous address. If, instead, memory is organized as individually addressable 8-bit bytes, then add 2 to the previous address.
- **Instruction fetch (if):** Read instruction from its memory location into the processor.
- ***Instruction operation decoding (iod):*** Analyze instruction to determine type of operation to be performed and operand(s) to be used.
- **Operand address calculation (oac):** If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand.
- ***Operand fetch (of):*** Fetch the operand from memory or read it in from I/O.
- ***Data operation (do):*** Perform the operation indicated in the instruction.
- ***Operand store (os):*** Write the result into memory or out to I/O.

States in the upper part of Figure 3.6 involve an exchange between the processor and either memory or an I/O module. States in the lower part of the diagram involve only internal processor operations. The oac state appears twice, because an instruction may involve a read, a write, or both. However, the action performed during that state is fundamentally the

same in both cases, and so only a single state identifier is needed. Also note that the diagram allows for multiple operands and multiple results, because some instructions on some machines require this.

Interrupts in instruction cycle

Interrupts is an asynchronous service request from hardware or software to CPU. Interrupts make the CPU execution of it's normal operation to pause to service external devices or errors. The processor and the OS are responsible for recognizing in interrupt suspending the user program, servicing the interrupt and then resuming the user program. The instruction cycle with interrupts is as shown below:

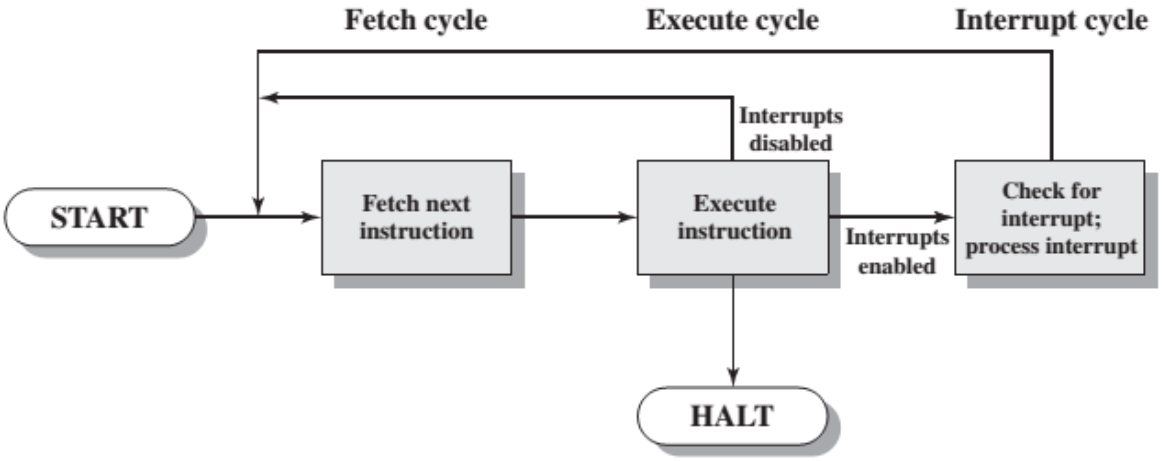


Figure 3.9 Instruction Cycle with Interrupts

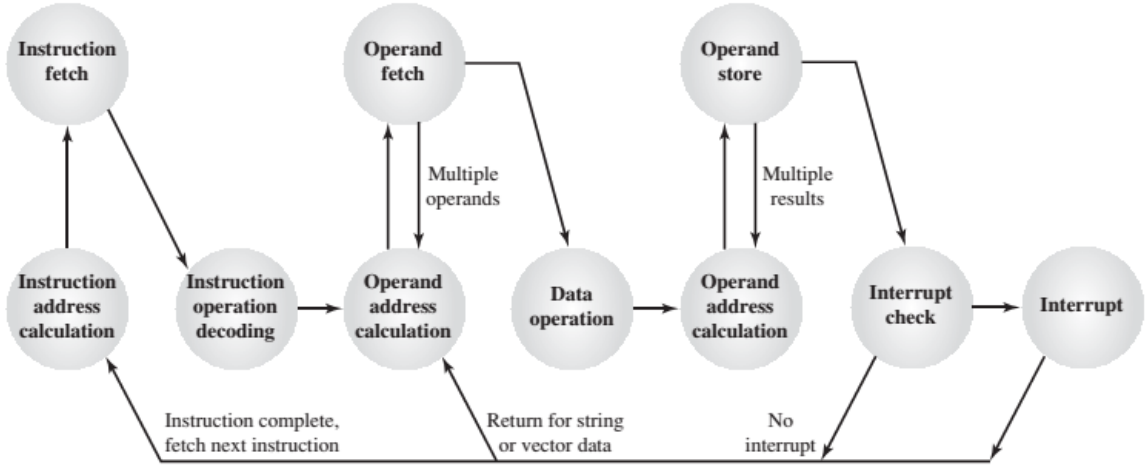


Figure 3.12 Instruction Cycle State Diagram, with Interrupts

Instruction format

An instruction format defines the layout of the bits of an instruction, in terms of its constituent fields. An instruction format must include an opcode and, implicitly or

explicitly, zero or more operands. Each explicit operand is referenced using one of the addressing modes. The format must, implicitly or explicitly, indicate the addressing mode for each operand. Some of the key design issue are:

Instruction Length

The decision of instruction format length is affected by, memory size, memory organization, bus structure, processor complexity, and processor speed. The most obvious trade-off here is between the desire for a powerful instruction and a need to save space. Programmers want more opcodes, more operands, more addressing modes, and greater address range. More opcodes and more operands make life easier for the programmer, because shorter programs can be written to accomplish given tasks. Similarly, more addressing modes give the programmer greater flexibility in implementing certain functions, such as table manipulations and multiple-way branching. And, of course, with the increase in main memory size and the increasing use of virtual memory, programmers want to be able to address larger memory ranges. All of these things (opcodes, operands, addressing modes, address range) require bits and push in the direction of longer instruction lengths. But longer instruction length may be wasteful. Another trade off, consideration may be either instruction length should be equal to the memory transfer length or multiple, otherwise we will not get an integral number of instruction during fetch cycle.

Allocation of Bits

For a given instruction length, there is clearly a trade-off between the number of op-codes and the power of the addressing capability. More op-codes obviously mean more bits in the op-code field. For an instruction format of a given length, this reduces the number of bits available for addressing. There is one interesting refinement to this trade-off, and that is the use of variable-length op-codes. In this approach, there is a minimum op-code length but, for some op-codes, additional operations may be specified by using additional bits in the instruction. For a fixed-length instruction, this leaves fewer bits for addressing. Thus, this feature is used for those instructions that require fewer operands and/or less powerful addressing. The following interrelated factors go into determining the use of the addressing bits.

- **Number of addressing modes:** Sometimes an addressing mode can be indicated implicitly. For example, certain op-codes might always call for indexing. In other cases, the addressing modes must be explicit, and one or more mode bits will be needed.

Number of operands: Typical instructions on today's machines provide for two operands. Each operand address in the instruction might require its own mode indicator, or the use of a mode indicator could be limited to just one of the address fields.

- **Register versus memory:** With a single user-visible register (usually called the accumulator), one operand address is implicit and consumes no instruction bits. However, single-register programming is awkward and requires many instructions. Even with multiple registers, only a few bits are needed to specify the register. The more that registers can be used for operand references, the fewer bits are needed.
- **Number of register sets:** Machines have one set of general purpose registers, with typically 32 or more registers in the set. These registers can be used to store data and can be used to store addresses for displacement addressing. Some architectures have a collection of two or more specialized sets (such as data and displacement). One advantage

of this latter approach is that, for a fixed number of registers, a functional split requires fewer bits to be used in the instruction. For example, with two sets of eight registers, only 3 bits are required to identify a register; the op-code or mode register will determine which set of registers is being referenced.

• **Address range:** the range of addresses that can be referenced is related to the number of address bits. Because this imposes a severe limitation, direct addressing is rarely used. With displacement addressing, the range is opened up to the length of the address register. Even so, it is still convenient to allow rather large displacements from the register address, which requires a relatively large number of address bits in the instruction.

• **Address granularity:** In a system with 16 or 32 bits word or address can reference a word or a byte at designers choice. Byte addressing is convenient for character manipulation but required for a fixed sixe of memory more address bits.

Variable length instruction: The instruction discussed so far have single fixed length, but the designer may choose instead to provide a variety of instruction formats of different lengths, this tactic makes it easy to provide a large no. of op-codes with different lengths and the address length can also be varied with variable length instruction, these many variations can be provided efficiently and compactly.

Instruction representation

Within the computer each instruction is represented by a sequence of bits. The instruction is divided into fields, corresponding to the constituents elements of the instruction. It is difficult for both the programmer and reader to deal with binary representation, so it is

common to use symbolic representation. Code are represented by abbreviations called nmemonics that indicate the operations.

ADD: Add, SUB: Subtract, MUL: Multiply, DIV: Divide.

Instruction types

An instruction set should be functionally complete. It should formulate any high level data processing task. Such operations can be grouped as:

1. Data processing: ALU
2. Data movement: I/O instruction
3. Data storage: memory instruction
4. Control operations: test and branch instruction.

Good instruction set should meet following standards:

1. Completeness: to be able to construct a machine level program to evaluate any computable function.
2. Efficient: frequently performed instruction should be done quickly with few instructions.
3. Regular and complete class of instruction: provide logical set of operations.
4. Orthogonal: define instruction, data types and addressing independently.
5. Compatible: with existing hardware and software product of that time.

Number of address/ address in an instruction

In a typical arithmetic or logical instruction 3 address is required, 2 for operands and 1 for result. These address can be explicitly given or implied by instruction. Example: compare instruction Y= (A-B)/[C+ (D*E)] with one, two and three instructions

3-address instruction

Both operands and the destination for the result are explicitly contained in the instruction word.

Instruction	Comment
SUB Y,A,B	Y←A-B
MUL T,D,E	T←D*E
ADD T,T,C	T←T+C
DIV Y,Y,T	Y←Y/T

This format is rarely used due to the length of address themselves and resulting length of the instruction word.

2-address instruction

One of the address is used to specify both an operand and result location. Very common in instruction set.

Instruction	Comment
MOV Y,A	Y←A
SUB Y,B	Y←Y-B
MOV T,D	T←D
MUL T,E	T←T*E
ADD T,C	T←T+C
DIV Y,T	Y←Y/T

1-address instruction

Traditional accumulator based operations, ACC←ACC+X

Instruction	Comment
LOAD D	AC←D
MUL E	AC←AC*E
ADD C	AC←AC+C
STOR Y	Y←AC
LOAD A	AC←A
SUB B	AC←AC-B
DIV Y	A←AC/Y
STOR Y	Y←AC

0 address instruction

Zero address instruction are applicable to a special memory organization called stack. A stack is a last-in-first-out set of locations. Zero address instruction would reference the top of the two stack elements.

PUSH A	TOS←A
PUSH B	TOS←B
SUB	TOS←(A-B)
PUSH D	TOS←D
PUSH E	TOS←E
MUL	TOS←D*E
PUSH C	TOS←C
ADD	TOS←C+(D*E)
DIV	TOS←(A-B)/(C+D*E)
POP Y	Y←TOS

Addressing mode

The manner in which each address field specifies memory location is called addressing modes. That can be of following

1. Immediate mode:

The operand is contained within the instruction itself. Data is a constant at run time. No additional memory reference are required after the fetch of the instruction. Size of the operand is limited, i.e range of value limited.

Operand A

2. Direct addressing mode:

The address field of the instruction contains the effective address of the operand. No calculation required. One additional memory access is required to fetch the operand. Address range is limited by the width of the field that contains the address reference. Address is a constant at run time but data itself can be changed during the program execution.

EA=A

3. Indirect addressing:

The address filed in the instruction specified a memory location which contains the address of the data. In indirect addressing, address field refer to the address of a word in memory which in-turn contains a full length address of operand.

4. Register based addressing modes

Register addressing: similar to direct addressing the only difference is that the address field refers to a register rather than a main memory address.

EA=R

Register indirect: like indirect, but address filed specifies a register that contains the effective address.

Advantage

i. Only a small address filed is needed in the instruction.
ii. No time consuming memory reference are required.

Dis-advantage

i. If registers are heavily used, this will limit the performance of the processor.
ii. If frequently used there will be more immediate steps involved. So only used if the operand in a register remains in use for multiple operations.

5. Displacement or address relative addressing

EA=A+(R)

Displacement address requires that the instruction have two address filed at least one of which is explicit (definite). The value contained in one address field (value=A) is directly added to a register to produce effective address. The most commonly used displacement addressing:

a. Relative addressing:

- A is added to the program counter contents to cause a branch operations in fetching the next instruction.

b. Base register addressing:

- The referenced register contains a main memory address and the address register field contains a displacement (usually unsigned integer) from that address.

c. Indexing:

The address field references a main memory address and the referenced register contains a positive displacement from that address.

6. Stack addressing:

A stack is a linear array of location. It is sometime referred to as a pushdown list or last-in-first-out queue. Items are appended to the top of the stack so that at any given time the block is partially filled. Associated with stack is a pointer whose value is the address of the top of the stack. Alternatively, the top two elements of the stack may be in processor register, in which case the stack pointer references the third element of the stack.

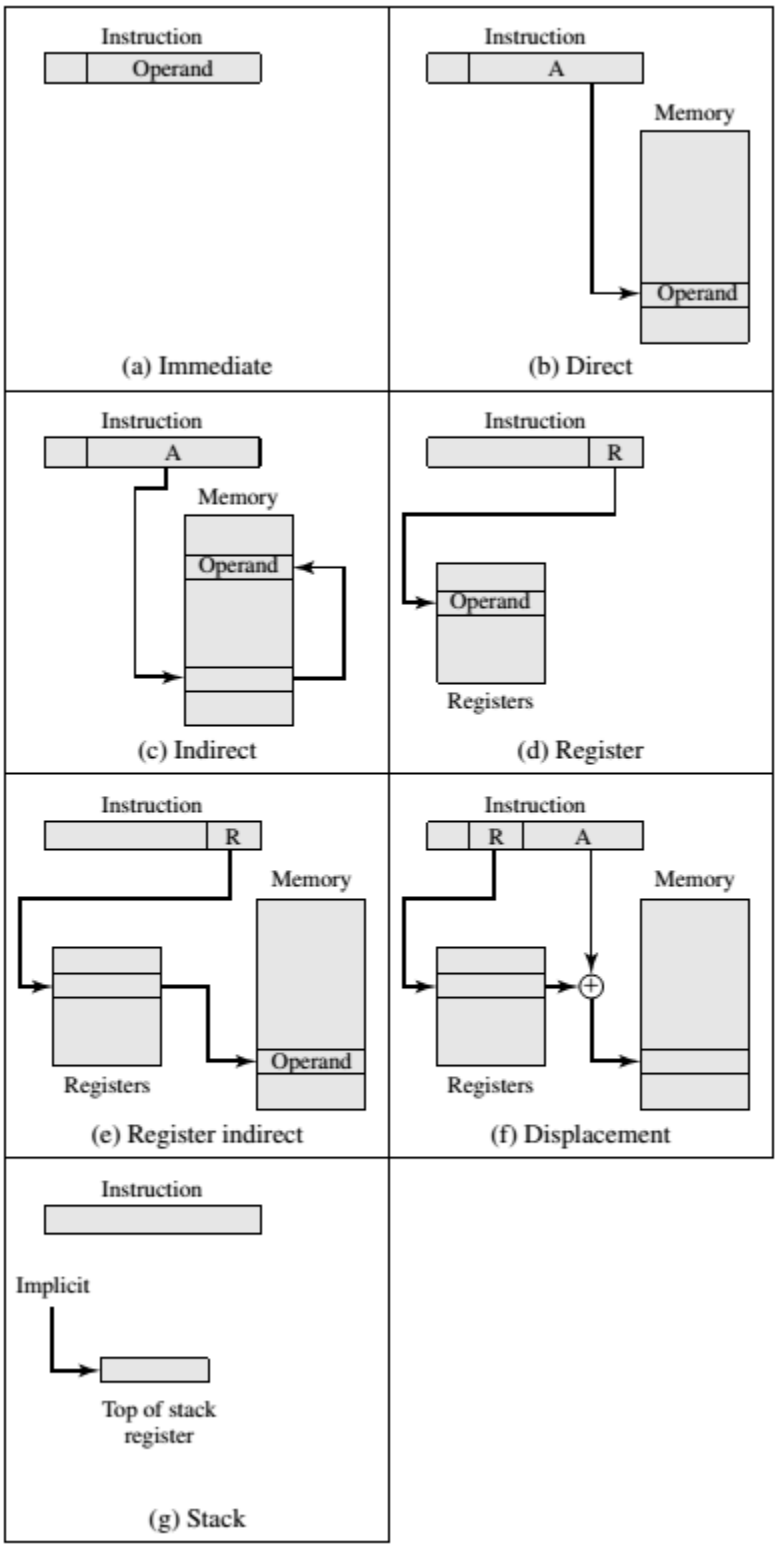


Figure 11.1 Addressing Modes

Stack, reverse polish notation
A+B: infix notation
+AB: prefix or polish notation
AB+: postfix or reverse polish notation
(A*B+C*D) in reverse polish notation is AB*CD*+

Scan the expression from let to right, when an operator is reached perform the operation with operands found on the left side of the operator. Remove the two operands and the operator and replace them by the number obtained from the result of the operation. Continue to scan the expression and repeat the procedure for every operator encountered until there are no more operators.

Example: (A+B)*[C*(D+E) +F]

Convert to reverse polish notation
- First perform all arithmetic inside the inner parameters.
- Then inside outer parentheses.
- Do multiplication and division before addition and subtraction operations.

Final result is: AB+DE+C*F+*

****Note:** examples of computer families, see table 2.3 for IBM 700/7000 series, IBM system/360

The evolution of intel x86 architecture, see table 2.6 from William stallings

Future trends in computer

- ARM, embedded system

Designing computer for performance
Year by year, the cost of computer systems continues to drop dramatically, while the performance and capacity of those systems continue to rise equally dramatically. Most of the today's computer requires some of the following capabilities:
• Image processing
• Speech recognition
• Videoconferencing
• Multimedia authoring
• Voice and video annotation of files
• Simulation modeling

Basically the building blocks for today's computer are virtually same as those of the IAS computer, but the techniques for making performance high has improved a lot and sophisticated. Now we highlight some of the driving factors behind the need to design for performance.

- **Microprocessor Speed**

What gives these processors such mind-boggling power is the relentless pursuit of speed by processor chip manufacturers. The evolution of these machines continues to bear out Moore's law, mentioned previously. So long as this law holds, chipmakers can unleash a new generation of chips every three

Years with four times as many transistors. In memory chips, this has quadrupled the capacity of dynamic random-access memory (DRAM), still the basic technology for computer main memory, every three years. In microprocessors, the addition of new circuits, and the speed boost that comes from reducing the distances between them, has improved performance four or fivefold every three years. But the raw speed of the microprocessor will not achieve its potential unless it is fed a constant stream of work to do in the form of computer instructions. Anything that gets in the way of that smooth flow undermines the power of the processor. Accordingly, while the chipmakers have been busy learning how to fabricate chips of greater and greater density, the processor designers must come up with ever more elaborate techniques for feeding the monster. Among the techniques built into contemporary processors are the following:

- **Branch prediction:** The processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, are likely to be processed next. If the processor guesses right most of the time, it can prefetch the correct instructions and buffer them so that the processor is kept busy. The more sophisticated examples of this strategy predict not just the next branch but multiple branches ahead. Thus, branch prediction increases the amount of work available for the processor to execute.
- **Data flow analysis:** The processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions. In fact, instructions are scheduled to be executed when ready, independent of the original program order. This prevents unnecessary 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. This enables the processor to keep its execution engines as busy as possible by executing instructions that are likely to be needed.
- **Performance Balance**

  While processor power has raced ahead at breakneck speed, other critical components of the computer have not kept up. The result is a need to look for performance balance: an adjusting of the organization and architecture to compensate for the mismatch among the capabilities of the various components.

Consider an example the speed of processor has increased tremendously or rapidly but not the speed of data transfer from main memory to processor, so the processor has to wait for the data from memory to come causing processor stall or wait for data causing overall slow performance. The no of ways the system architect can solve this problem are:

- Increasing the number of bits that are retrieved from memory.
- Introducing a cache or other buffering schemes.
- Reducing the access to memory incorporating more caches.
- Using high speed buses for faster data transfer.

- Another example I/O peripheral devices. The key idea here is to balance the throughput and processing demands of the processor, main memory, I/O devices, inter connecting structures.

Improvements in Chip Organization and Architecture

As designers wrestle with the challenge of balancing processor performance with that of main memory and other computer components, the need to increase processor speed remains. There are three approaches to achieving increased processor speed:

- **Increase the hardware speed of the processor:** This increase is fundamentally due to shrinking the size of the logic gates on the processor chip, so that more gates can be packed together more tightly and to increasing the clock rate. With gates closer together, the propagation time for signals is significantly reduced, enabling a speeding up of the processor. An increase in clock rate means that individual operations are executed more rapidly.

- Increase the size and speed of caches that are interposed between the processor and main memory. In particular, by dedicating a portion of the processor chip itself to the cache, cache access times drop significantly.

- Make changes to the processor organization and architecture that increase the effective speed of instruction execution. Typically, this involves using parallelism in one form or another.

Here are some obstacles or factors that limit or hold back in increasing the performance using above method.

- **Power:** As the density of logic and the clock speed on a chip increase, so does the power density (Watts/cm2). The difficulty of dissipating the heat generated on high-density, high-speed chips is becoming a serious design issue.

- **RC delay:** The speed at which electrons can flow on a chip between transistors is limited by the resistance and capacitance of the metal wires connecting them; specifically, delay increases as the RC product increases. As components on the chip decrease in size, the wire interconnects become thinner, increasing resistance. Also, the wires are closer together, increasing capacitance.

- **Memory latency:** Memory speeds lag processor speeds. Difference between access time of processor and memory.

Thus, there will be more emphasis on organization and architectural approaches to improving performance. Two main strategies that have been used to increase performance beyond what can be achieved simply by increasing clock speed are:

- Increasing cache capacity, using two or more levels of caches.
- Using parallel execution, like pipelining & superscalar methods.

Evolution of Intel X86 architecture see William stallings book on computer organization and architecture designing for performance.