Computer Organization & Architecture

- Architecture describes what the computer does and organization describes how it does.

Architecture → the way hardware components are connected together to form a computer system

- Attributes visible to the programmer
- Instruction set, addressing techniques, number of bits used for data representation, I/O mechanisms,
- Ex: Is there a multiply instruction?

Organization → the structural behaviour of a computer system

- how features are implemented
- Control signals, interfaces between computer and peripherals, memory technologies …
- Ex: Is there a hardware Multiply unit or is it done by repeated addition?

Central processing unit

- The part of the computer that performs the bulk of data-processing operations is called the central processing unit and is referred to as the CPU

- The CPU is made up of three major parts, as shown in Fig.
  - The register set stores intermediate data used during the execution of the instructions.
  - The arithmetic logic unit (ALU) performs the required microoperations for executing the instructions.
  - The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

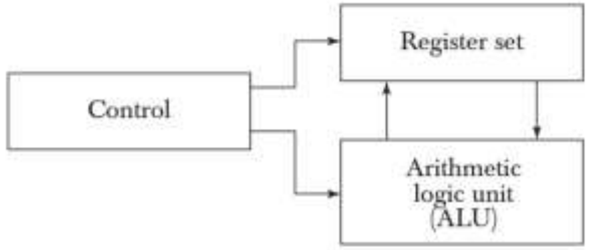
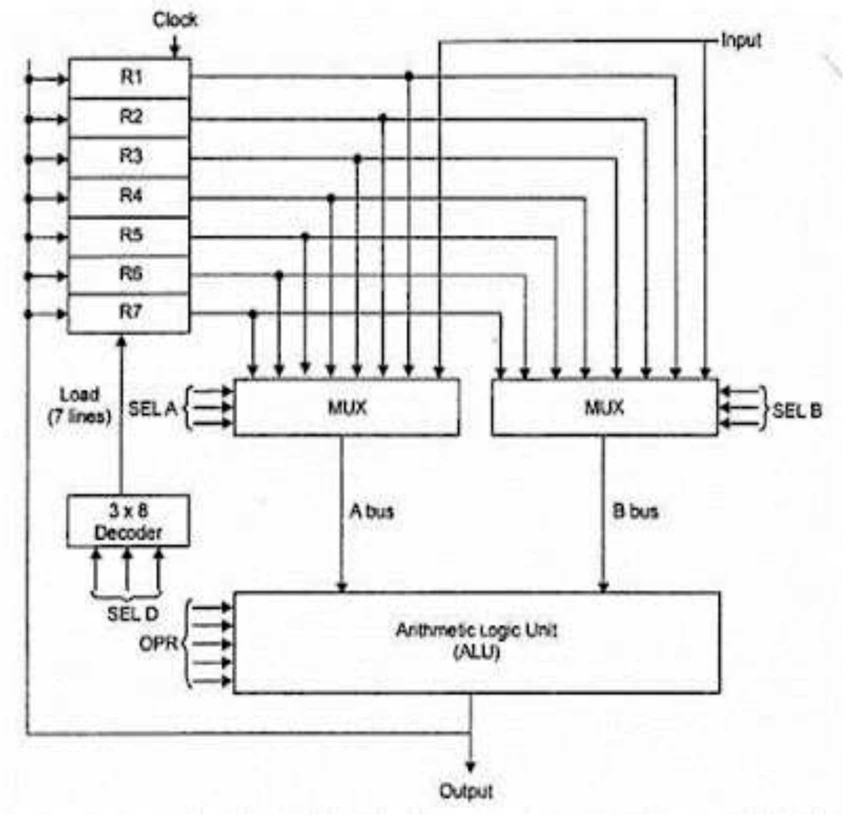


Figure 8-1 Major components of CPU.

➢ Computer architecture is sometimes defined as the computer structure and behavior as seen by the programmer that uses machine language instructions. This includes the instruction formats, addressing modes, instruction set, and the general organization of the CPU registers.

General Register Organization

- When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfers, but also while performing various microoperations.
- Register organization shows how registers are selected and how data flow between registers and ALU.
- A decoder is used to select a particular register. The output of each register is connected to two multiplexers to form the two buses A and B.
- The selection lines in each multiplexer select the input data for the particular bus. The A and B buses form the two inputs of an ALU.



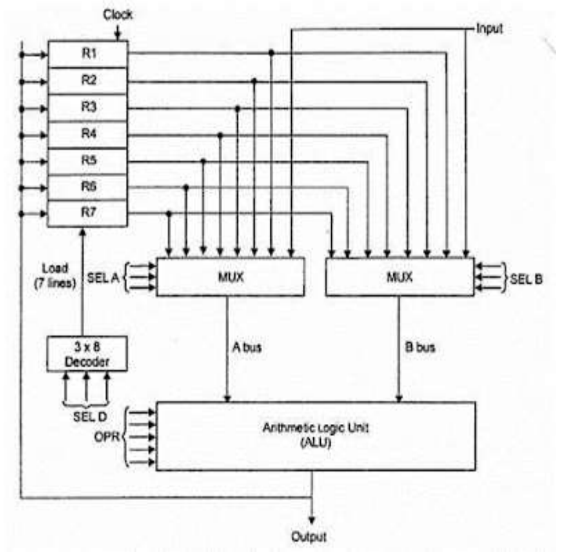
General Register Organization

- The operation select lines decide the micro operation to be performed by ALU. The result of the micro operation is available at the output bus.
- The output bus connected to the inputs of all registers, thus by selecting a destination register it is possible to store the result in it.

Example:

- To perform the operation ***R3 = R1+R2*** We have to provide following binary selection variable to the select inputs.

1. **SEL A** : **001** -To place the contents of R1 into bus A.
2. **SEL B** : **010** - to place the contents of R2 into bus B
3. **SEL OPR** : **10010** – to perform the arithmetic addition A+B
4. **SEL REG or SEL D : 011** – to place the result available on output bus in R3.



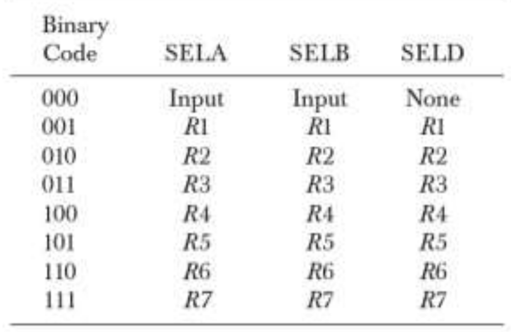


TABLE 8-1 Encoding of Register Selection Fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	*R1*	*R1*	*R1*
010	*R2*	*R2*	*R2*
011	*R3*	*R3*	*R3*
100	*R4*	*R4*	*R4*
101	*R5*	*R5*	*R5*
110	*R6*	*R6*	*R6*
111	*R7*	*R7*	*R7*

General Register Organization

- The operation select lines decide the micro operation to be performed by ALU. The result of the micro operation is available at the output bus.
- The output bus connected to the inputs of all registers, thus by selecting a destination register it is possible to store the result in it.

Example:

- To perform the operation ***R3 = R1+R2*** We have to provide following binary selection variable to the select inputs.

1. **SEL A** : **001** -To place the contents of R1 into bus A.
2. **SEL B** : **010** - to place the contents of R2 into bus B
3. **SEL OPR** : **10010** – to perform the arithmetic addition A+B
4. **SEL REG or SEL D : 011** – to place the result available on output bus in R3.

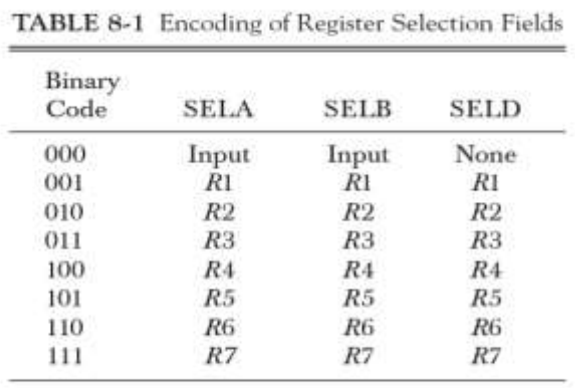


TABLE 8-1 Encoding of Register Selection Fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	*R*1	*R*1	*R*1
010	*R*2	*R*2	*R*2
011	*R*3	*R*3	*R*3
100	*R*4	*R*4	*R*4
101	*R*5	*R*5	*R*5
110	*R*6	*R*6	*R*6
111	*R*7	*R*7	*R*7

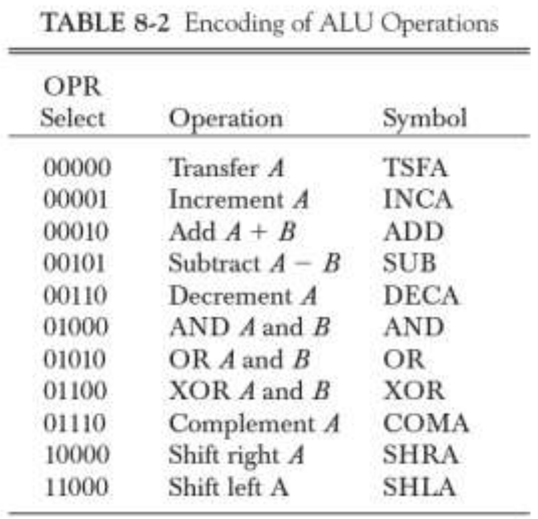
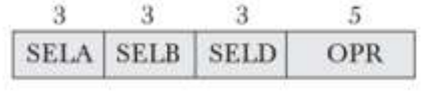


TABLE 8-2 Encoding of ALU Operations

OPR Select	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Add *A* + *B*	ADD
00101	Subtract *A* − *B*	SUB
00110	Decrement *A*	DECA
01000	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left A	SHLA

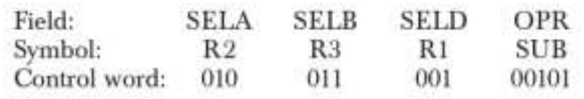
What is CONTROL WORD?

- The combined value of a binary selection inputs specifies the control word.
- There are 14 binary selection inputs in the unit, and their combined value specifies a control word.
- It consist of four fields SELA, SELB, and SELD contains three bit each and SEL-OPR field contains four bits thus the total bits in the control word are 14-bits.
  1. The three bit of SELA select a source registers of the a input of the ALU.
  2. The three bits of SELB select a source registers of the b input of the ALU.
  3. The three bits of SELED or SELREG select a destination register using the decoder.
  4. The four bits of SELOPR select the operation to be performed by ALU.



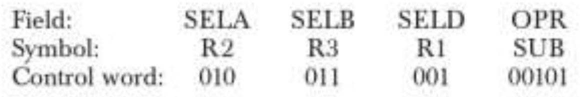
(b) Control word

the operation ***R3 = R1+R2***



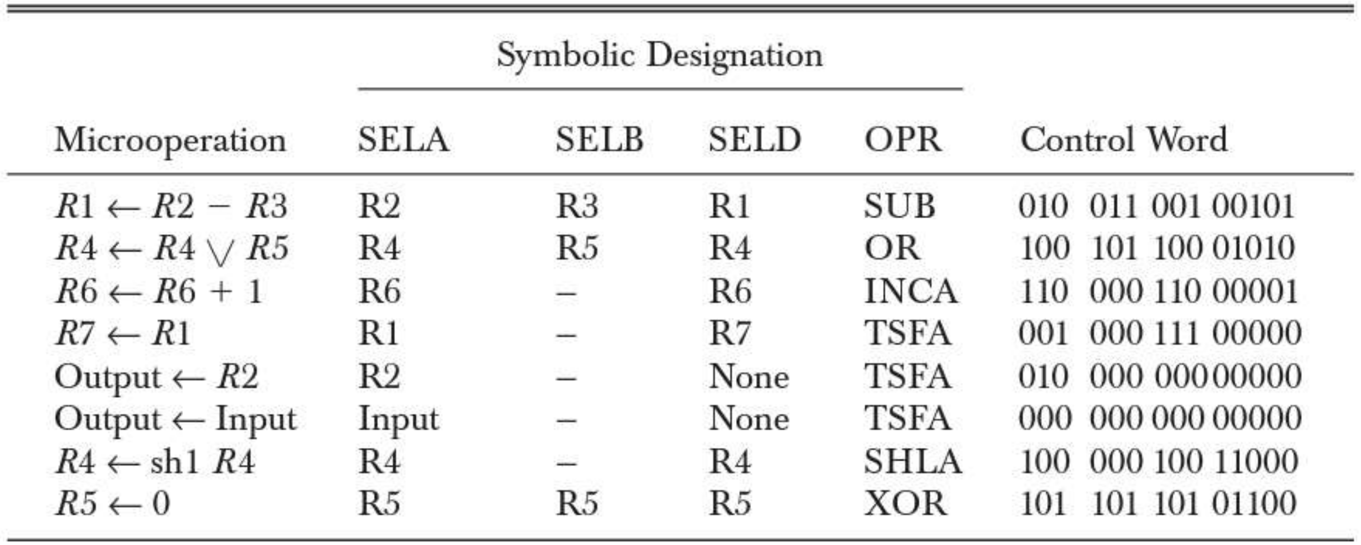
Field:	SELA	SELB	SELD	OPR
Symbol:	R2	R3	R1	SUB
Control word:	010	011	001	00101

CONTROL WORD?



Field:	SELA	SELB	SELD	OPR
Symbol:	R2	R3	R1	SUB
Control word:	010	011	001	00101

TABLE 8-3 Examples of Microoperations for the CPU



	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	–	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	–	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	–	None	TSFA	010 000 00000000
Output $\leftarrow$ Input	Input	–	None	TSFA	000 000 000 00000
$R4 \leftarrow$ sh1 $R4$	R4	–	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

Instruction Formats

- The bits of the instruction are divided into groups called fields.
- The most common fields found in instruction formats are:

1.An operation code(**opcode**) field that specifies the operation to be performed.

2.An **address** field that designates a memory address or a processor register.

3.A **mode** field that specifies the way the operand or the effective address is determined.

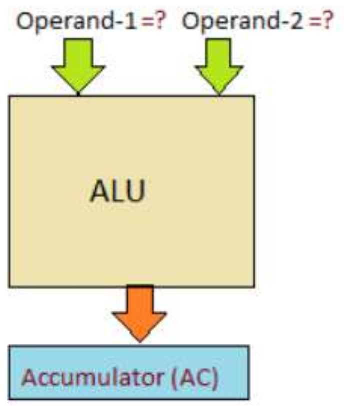
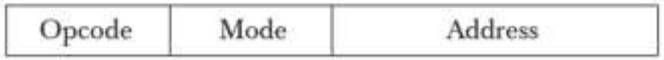


Figure- Instruction format and ALU

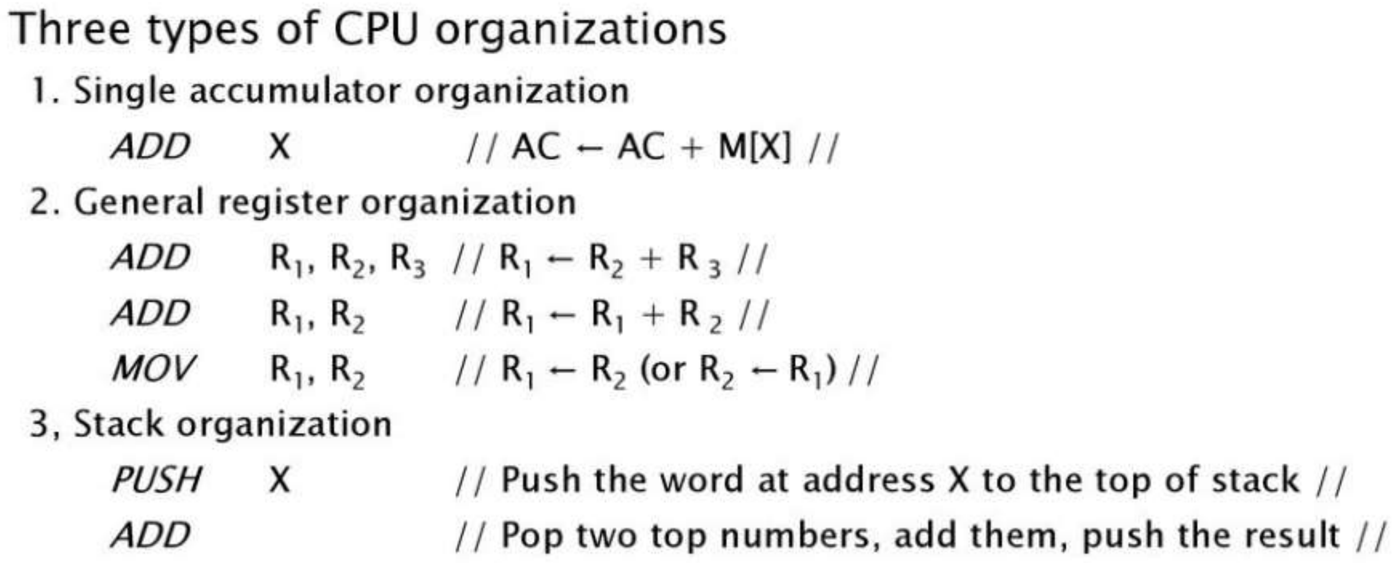
Figure 8-6 Instruction format with mode field.



Opcode	Mode	Address

Types of CPU organizations

1. Single accumulator organization.
2.General register organization.
3. Stack organization.



Three types of CPU organizations

1. Single accumulator organization

ADD	X	// $AC \leftarrow AC + M[X]$ //

2. General register organization

ADD	R_1, R_2, R_3	// $R_1 \leftarrow R_2 + R_3$ //
ADD	R_1, R_2	// $R_1 \leftarrow R_1 + R_2$ //
MOV	R_1, R_2	// $R_1 \leftarrow R_2$ (or $R_2 \leftarrow R_1$) //

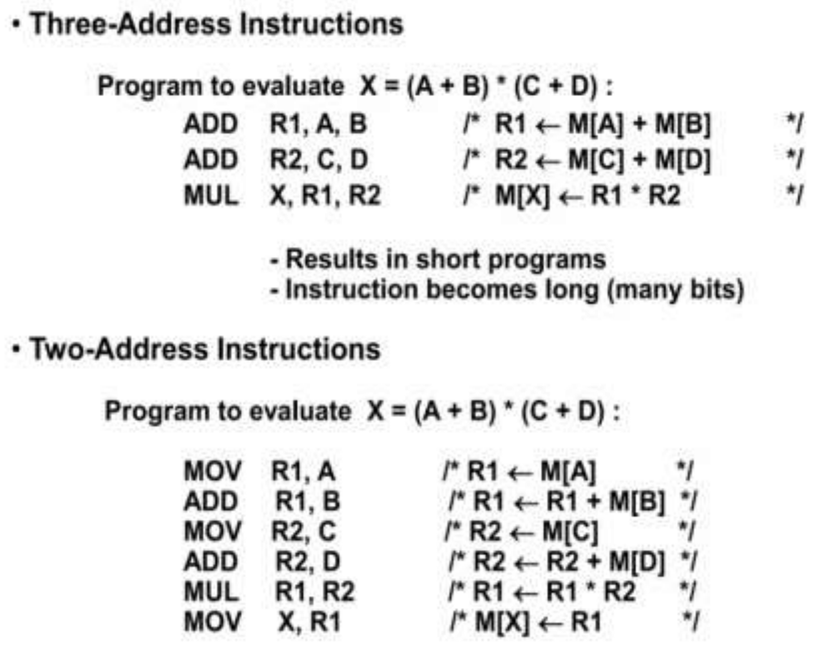
3, Stack organization

PUSH	X	// Push the word at address X to the top of stack //
ADD		// Pop two top numbers, add them, push the result //

Instruction Formats

- To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement

$$X = (A + B) * (C + D)$$



- **Three-Address Instructions**

Program to evaluate X = (A + B) * (C + D) :

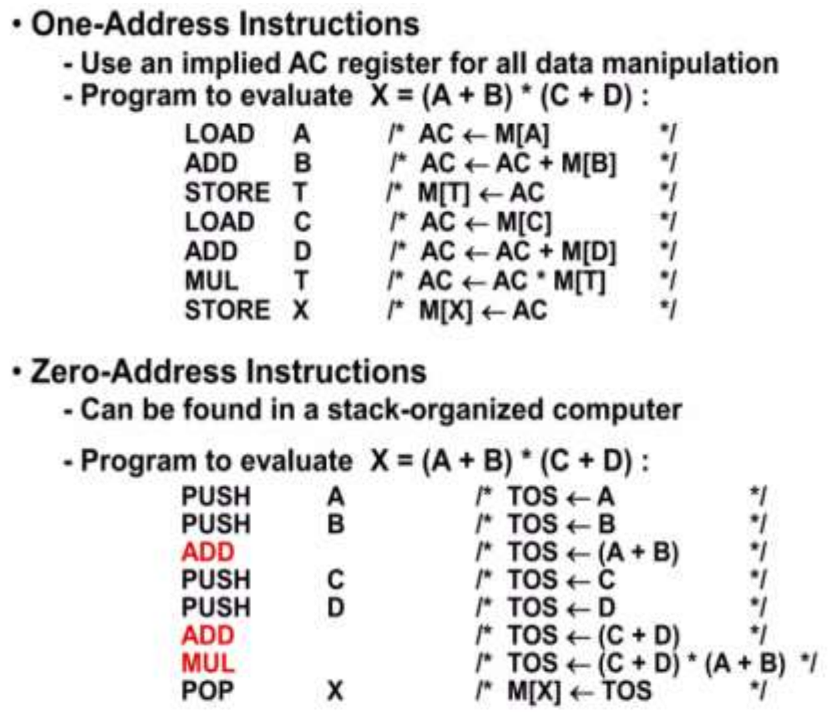
```
ADD    R1, A, B      /*  R1 ← M[A] + M[B]      */
ADD    R2, C, D      /*  R2 ← M[C] + M[D]      */
MUL    X, R1, R2     /*  M[X] ← R1 * R2        */
```

- Results in short programs
- Instruction becomes long (many bits)

- **Two-Address Instructions**

Program to evaluate X = (A + B) * (C + D) :

```
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          */
```



- **One-Address Instructions**
  - Use an implied AC register for all data manipulation
  - Program to evaluate X = (A + B) * (C + D) :

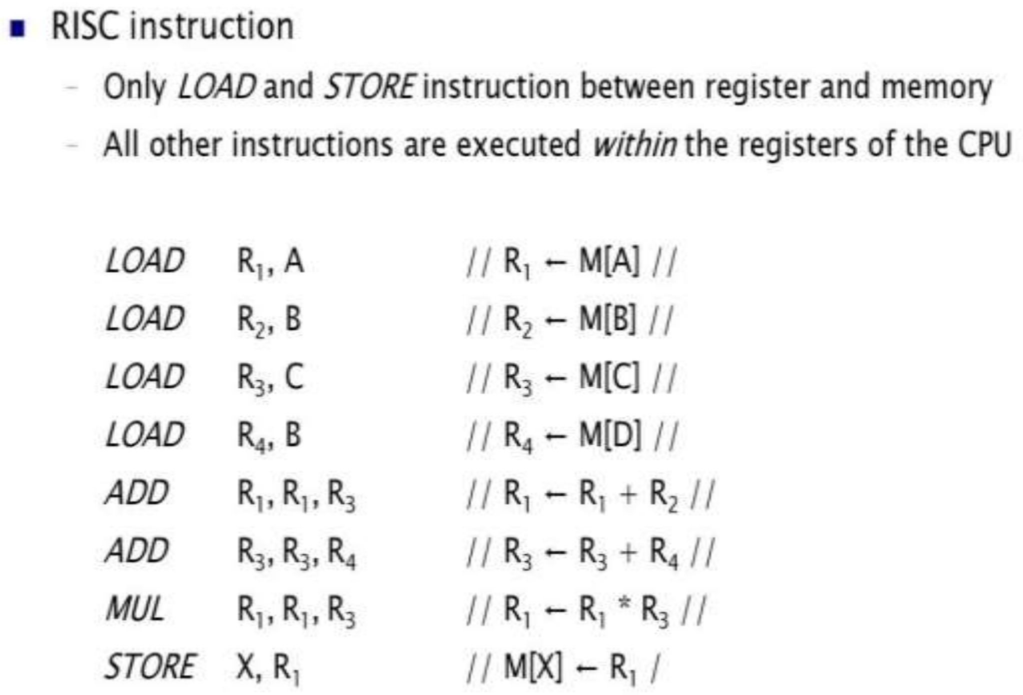
```
LOAD    A     /*  AC ← M[A]          */
ADD     B     /*  AC ← AC + M[B]     */
STORE   T     /*  M[T] ← AC          */
LOAD    C     /*  AC ← M[C]          */
ADD     D     /*  AC ← AC + M[D]     */
MUL     T     /*  AC ← AC * M[T]     */
STORE   X     /*  M[X] ← AC          */
```

- **Zero-Address Instructions**
  - Can be found in a stack-organized computer
  - Program to evaluate X = (A + B) * (C + D) :

```
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                 */
```

Instruction Formats

- The load instructions transfer the operands from memory to CPU registers. The add and multiply operations are executed with data in the registers without accessing memory. The result of the computations is then stored in memory with a store instruction.



- RISC instruction
  - Only *LOAD* and *STORE* instruction between register and memory
  - All other instructions are executed *within* the registers of the CPU

LOAD	R_1, A	// $R_1 \leftarrow M[A]$ //
LOAD	R_2, B	// $R_2 \leftarrow M[B]$ //
LOAD	R_3, C	// $R_3 \leftarrow M[C]$ //
LOAD	R_4, B	// $R_4 \leftarrow M[D]$ //
ADD	R_1, R_1, R_3	// $R_1 \leftarrow R_1 + R_2$ //
ADD	R_3, R_3, R_4	// $R_3 \leftarrow R_3 + R_4$ //
MUL	R_1, R_1, R_3	// $R_1 \leftarrow R_1 * R_3$ //
STORE	X, R_1	// $M[X] \leftarrow R_1$ /

Addressing Modes

- The operation field of an instruction specifies the operation to be performed.
- This operation must be executed on some data stored in computer registers or memory words.
- The way the operands are chosen during program execution is dependent on the addressing mode of the instruction.
- The addressing mode 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 one or both of the following provisions:

1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
2. To reduce the number of bits in the addressing field of the instruction

1. Implied Mode

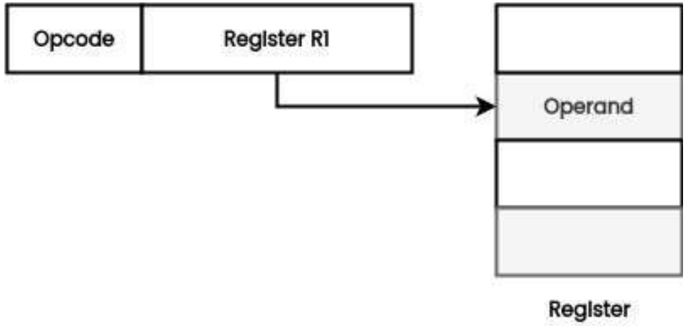
- The operands are specified implicitly in the definition of the instruction.
    - Eg: CMA - Complement Accumulator – operand in AC is implied in the definition of instruction.
- All register reference instructions that use an accumulator are implied-mode instructions.
- Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

2. Immediate Mode

- The operand is specified in the instruction itself.
- Immediate mode 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.
- They are are useful for initializing registers to a constant value.
  - Eg: ADD 7

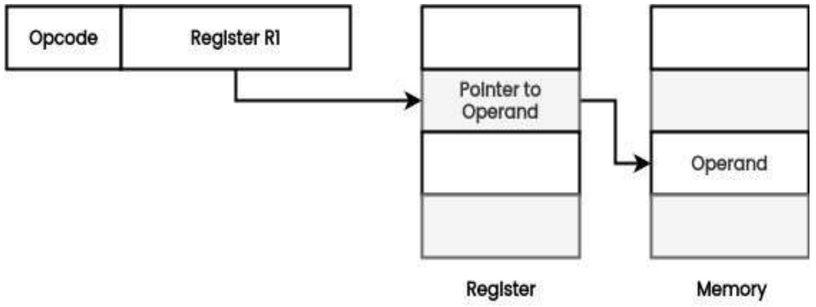
3. Register Mode

- When the address field specifies a processor register, instruction is in register mode.
- In this mode, the operands are in registers that reside within the CPU.
- The particular register is selected from a register field in the instruction.
    - Eg: ADD R1



4. Register Indirect Mode

- The instruction specifies a CPU register whose contents give the address of operand in memory.
- The selected register contains the address of the operand rather than 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.
- Advantage - The address field of the instruction uses fewer bits to select a register than to specify a memory address directly.

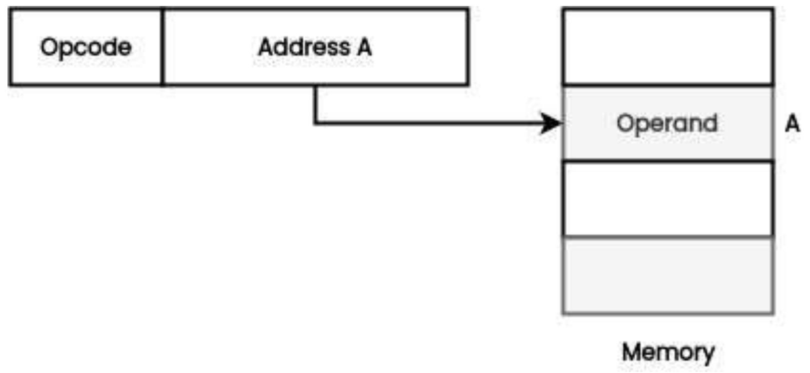


5. Autoincrement or Autodecrement Mode

- Similar to the 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 register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.
- This can be achieved by using the increment or decrement instruction.
- The address field of instruction is used by the CU of CPU to obtain the operand from memory.

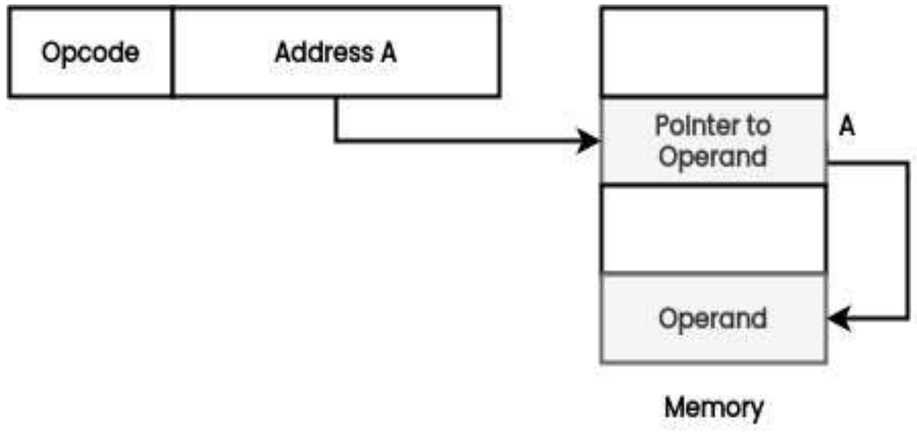
6.Direct Address Mode

- 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.
- In a branch-type instruction the address field specifies the actual branch address.



7. Indirect Address Mode

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



8. Relative Address Mode

- Content of PC is added to address part of instruction in order to obtain the effective address.
- The address part of instruction is usually a signed number (positive or negative) which is added to the content of the PC, producing an effective address whose position in memory is relative to the address of next instruction.
- It is often used with branch-type instructions.

Example:

- Let PC =825 and address part of the instruction contains the number 24.
- The instruction at location 825 is read from memory during fetch phase and PC is incremented by one to 826.
- The effective address computation for the relative address mode is 826 + 24 = 850.
- This is 24 locations forward from the address of next instruction

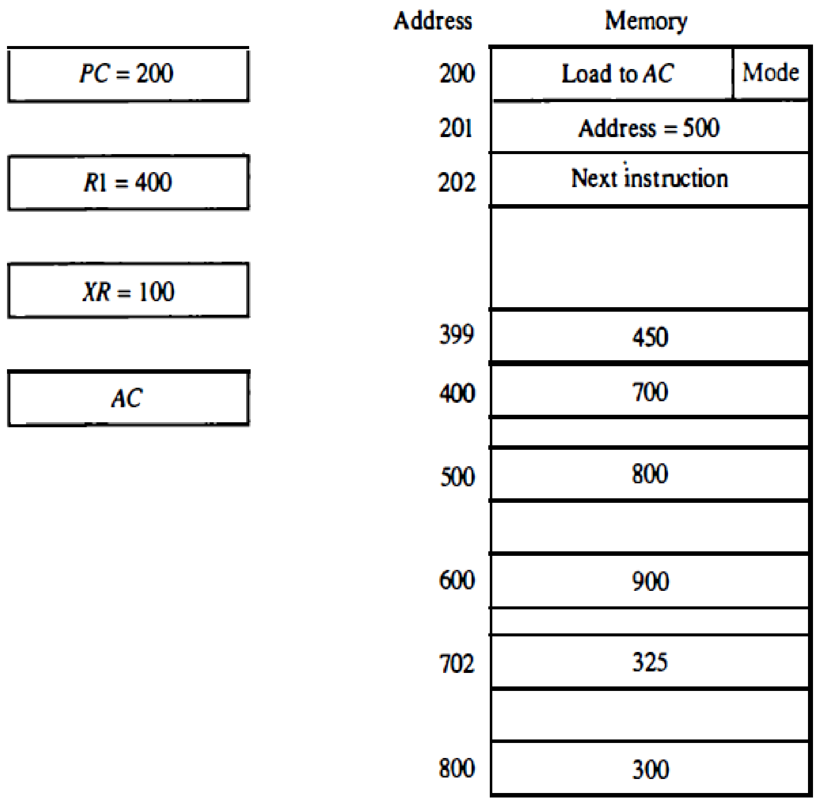
9. Indexed Addressing Mode

- Content of an index register 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.
- The address field of the instruction defines the beginning address of a data array in memory.
- The distance between the beginning address and the address of the operand is the index value stored in the index register.
- Any operand in the array can be accessed with the same instruction if the index register contains the correct index value.
- The index register can be incremented to facilitate access to consecutive operands.

10. Base Register Addressing Mode

- The content of base register is added to address part of the instruction to obtain the effective address.
- Similar to indexed addressing mode, except that register is called a base register.
- Difference between two modes is in the way they are used rather than the way they are computed.
- A base register holds base address and the address field of the instruction gives a displacement relative to this base address. This mode is used to facilitate the relocation of programs in memory.
- When programs and data are moved from one segment of memory to another, as required in multiprogramming systems, the address values of instructions must reflect this change of position.
- Here only the value of the base register requires updating to reflect the beginning of a new memory segment.

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

The two word instruction at location 200 and 201 is a "load to AC" with an address field equal to 500

here PC = 200 = program counter

R_1 = 400 = processor register

XR = 100 = index register

AC receives operand after instruction is executed

Mode field specifies one of the addressing mode.

a) Direct Addressing mode

$EA = M[201] = 500$

$\Rightarrow$ operand $= M[500] = 800$

$\Rightarrow AC = 800$.

b) Immediate Mode

operand $= M[201] = 500 = AC$ [$\because$ No address field]

c) Indirect Mode

$EA = M[M[201]] = M[500] = 800$

$\Rightarrow$ operand $= M[800] = 300 = AC$

d) Relative mode

$EA = PC + M[201] = 202 + 500 = 702$ (after fetch $PC = 202$)

$\Rightarrow$ operand $= M[702] = 325 = AC$

e) Index Mode

$EA = XR + M[201] = 100 + 500 = 600$

$\Rightarrow$ operand $= M[600] = 900 = AC$

4) Register Mode

$EA =$ Register R_1

$\Rightarrow$ operand $= R_1 = 400$

g) Register indirect Mode

EA = content of R_1 = 400

$\Rightarrow$ operand = M [400] = 700 = AC

h) autoincrement mode

EA = content of R_1 = 400

$\Rightarrow$ operand = M [400] = 700 = AC and $R_1 \leftarrow R_1 + 1 \Rightarrow R_1 = 401$

i) autodecrement mode

② EA = content of R_1 = 399 ① $R_1 \leftarrow R_1 - 1 \Rightarrow R_1 = 399$

$\Rightarrow$ operand = M [399] = 450

1st decrement R_1 Then get The operand

Data Transfer and Manipulation

- Three categories of computer instructions
  1. Data transfer instruction
  2. Data Manipulation instruction
  3. Program control instruction
- Data transfer instructions (Tab. 8-5, 8-6)

  LD ADR // $AC \leftarrow M[ADR]$ (direct address //

  ST @ADR // $M[ADR] \leftarrow AC$ (indirect address //
- Data manipulation instructions (Tab. 8-7, 8-8, 8-9)
  1. Arithmetic instructions

     INC, DEC, ADD, MUL, NEG, etc.
  2. Logical and bit manipulation instructions

     CLR, COM, AND, OR, XOR, etc.
  3. Shift instructions

     SHR, SHL, ROR, ROL, etc.

- **Typical Data Transfer Instructions**

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

- **Data Transfer Instructions with Different Addressing Modes**

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$
Autodecrement	LD -(R1)	$R1 \leftarrow R1 - 1, AC \leftarrow M[R1]$

- **Three Basic Types:** Arithmetic instructions
  Logical and bit manipulation instructions
  Shift instructions

- **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

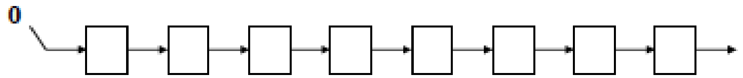
- **Logical and Bit Manipulation Instructions**

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

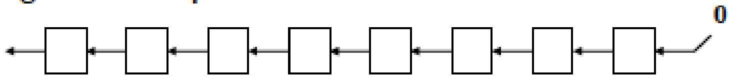
- **Shift Instructions**

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 thru carry	RORC
Rotate left thru carry	ROLC

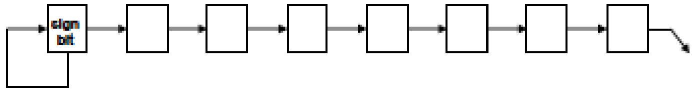
A right logical shift operation:



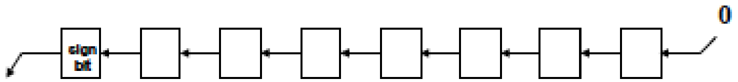
A left logical shift operation:



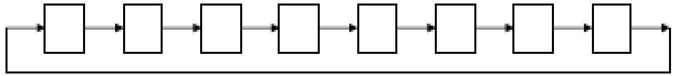
A right arithmetic shift operation:



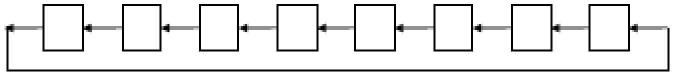
A left arithmetic shift operation:



A right circular shift operation:



A left circular shift operation:



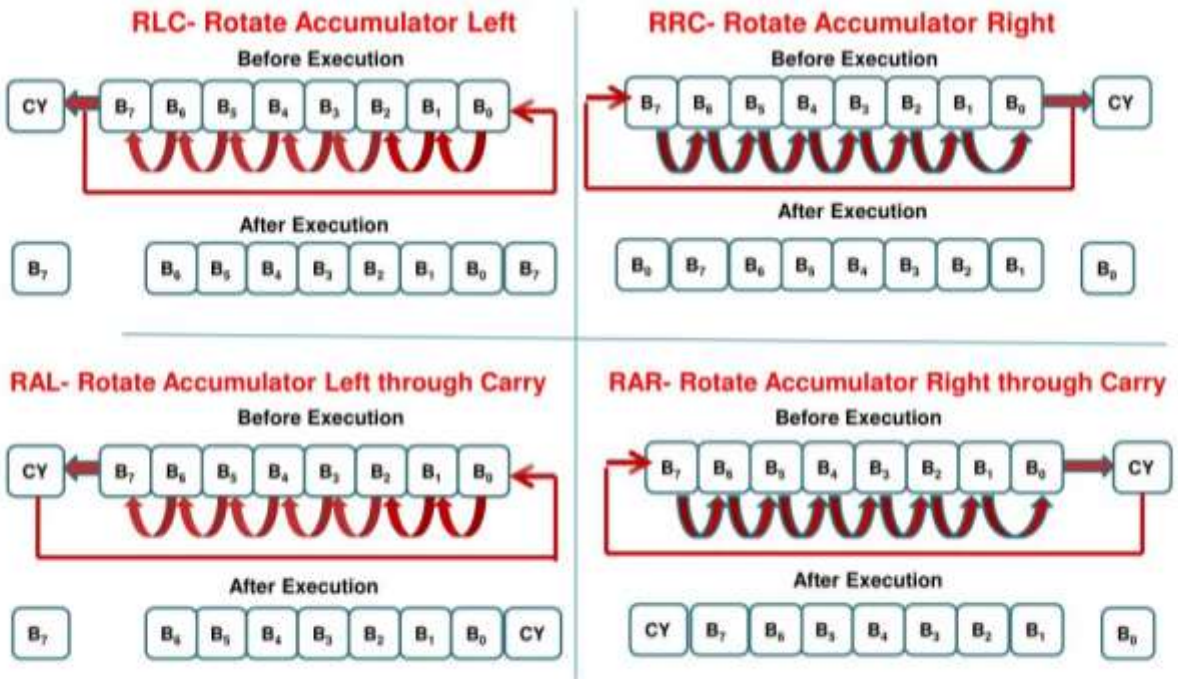
- **Shift Instructions**

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 thru carry	RORC
Rotate left thru carry	ROLC

- **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 thru carry	RORC
Rotate left thru carry	ROLC

Rotate Group


RLC- Rotate Accumulator Left
Before Execution
CY B7 B6 B5 B4 B3 B2 B1 B0
After Execution
B7 B6 B5 B4 B3 B2 B1 B0 B7
RRC- Rotate Accumulator Right
Before Execution
B7 B6 B5 B4 B3 B2 B1 B0 CY
After Execution
B0 B7 B6 B5 B4 B3 B2 B1 B0
RAL- Rotate Accumulator Left through Carry
Before Execution
CY B7 B6 B5 B4 B3 B2 B1 B0
After Execution
B7 B6 B5 B4 B3 B2 B1 B0 CY
RAR- Rotate Accumulator Right through Carry
Before Execution
B7 B6 B5 B4 B3 B2 B1 B0 CY
After Execution
CY B7 B6 B5 B4 B3 B2 B1 B0

CISC and RISC architectures

- CISC characteristics
  1. A large number of instruction : 100 to 250
  2. Some instructions that perform specialized tasks and are used infrequently
  3. A large variety of addressing modes : 5 to 20
  4. Variable-length instruction formats
  5. Instructions that manipulate operands in memory
- RISC characteristics
  1. Relatively few instructions
  2. Relatively few addressing modes
  3. Memory access limited to load and store instructions
  4. All operations done within the registers of the CPU
  5. Fixed-length, easily decoded instruction formats
  6. Single-cycle instruction execution
  7. Hardwired rather than μprogrammed control

CISC and RISC architectures

- CISC characteristics

1. A large number of instruction : 100 to 250
2. Some instructions that perform specialized tasks and are used infrequently
3. A large variety of addressing modes : 5 to 20
4. Variable-length instruction formats
5. Instructions that manipulate operands in memory

RISC architectures

1. Relatively few instructions
2. Relatively few addressing modes
3. Memory access limited to load and store instructions
4. All operations done within the registers of the CPU
5. Fixed-length, easily decoded instruction formats
6. Single-cycle instruction execution
7. Hardwired rather than μprogrammed control