SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

COMPUTER ARCHITECTURE AND ORGANIZATION (SCSA1402)

UNIT – I – Central Processing Unit – SCSA1402

UNIT.1 INTRODUCTION

Central Processing Unit - Introduction - General Register Organization - Stack organization -- Basic computer Organization - Computer Registers - Computer Instructions - Instruction Cycle. Arithmetic, Logic, Shift Microoperations- Arithmetic Logic Shift Unit -Example Architectures: MIPS, Power PC, RISC, CISC

Central Processing Unit

The part of the computer that performs the bulk of data-processing operations is called the central processing unit CPU. The CPU is made up of three major parts, as shown in Fig.1

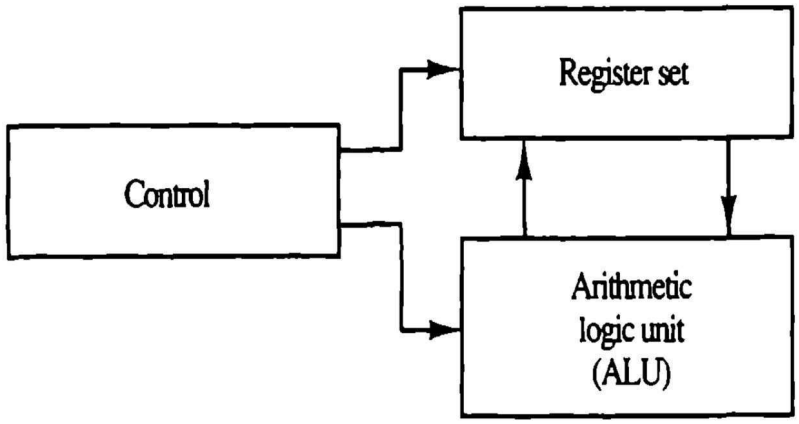


Fig 1. Major components of CPU.

- 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.

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.

Hence it is necessary to provide a common unit that can perform all the arithmetic, logic, and shift microoperations in the processor. A bus organization for seven CPU registers is shown in Fig.2.

- The output of each register is connected to two multiplexers (MUX) to form the two buses A and B.
- The selection lines in each multiplexer select one register or the input data for the particular bus. The A and B buses form the inputs to a common arithmetic logic unit (ALU).

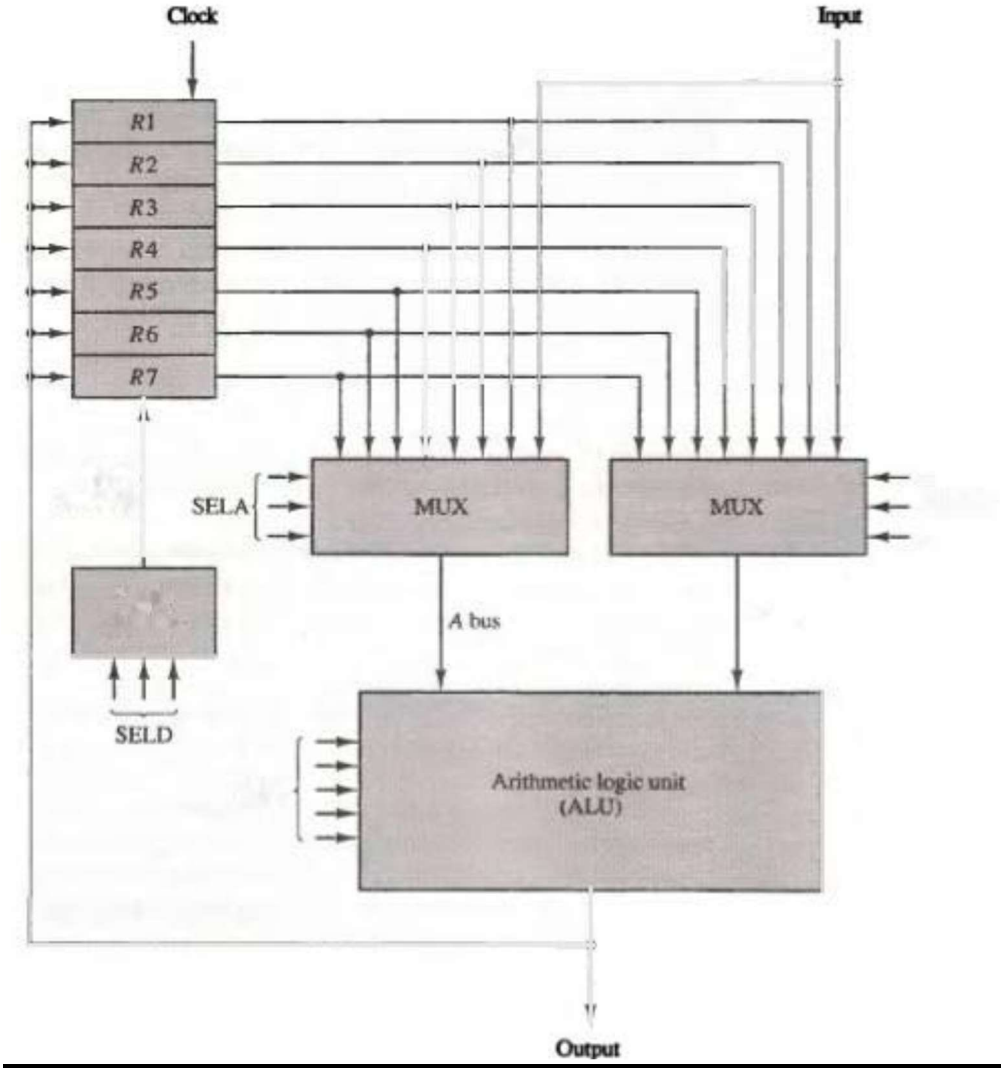


Fig 2 Register set with common ALU.

- The operation selected in the ALU determines the arithmetic or logic microoperation that is to be performed. The result of the microoperation is available for output data and also goes into the inputs of all the registers. The register that receives the information from the output bus is selected by a decoder. The decoder activates one of the register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register.
- The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system.
- For example, to perform the operation

R 1 <--R2 + R3

1. MUX A selector (SELA): to place the content of R2 into bus A.
2. MUX B selector (SELB): to place the content o f R 3 into bus B.
3. ALU operation selector (OPR): to provide the arithmetic addition A + B.
4. Decoder destination selector (SELD): to transfer the content of the output bus into R 1.

The four control selection variables are generated i n the control unit and must be available at the beginning of a clock cycle.

Control Word

There are 14 binary selection inputs in the unit, and their combined value specifies a control word. The 14-bit control word is defined in Fig. 4.

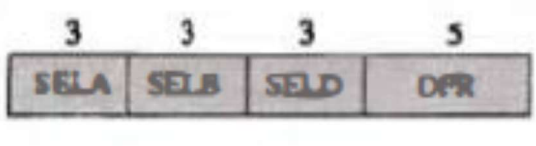


Fig 4. Control Word Format

TABLE 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

The encoding of the register selections is specified in Table 1. The 3-bit binary code listed in the first column of the table specifies the binary code for each of the three fields. The register selected by fields SELA, SELB, and SELD is the one whose decimal number is equivalent to the binary number in the code. When SELA or SELB is 000, the corresponding multiplexer selects the external input data. When SELD = 000, no destination register is selected but the contents of the output.

The ALU provides arithmetic and logic operations. The shifter may be placed in the input of the ALU to provide a preshift capability, or at the output of the ALU to provide postshifting capability. In some cases, the shift operations are included with the ALU. The function table for this ALU is listed in Fig.5. The encoding of the ALU operations for the CPU is specified in Table. The OPR field has five bits and each operation is designated with a symbolic name.

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

Fig.5 Encoding of ALU Operations

Stack Organization:

A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. The operation of a stack can be compared to a stack of trays. The last tray placed on top of the stack is the first to be taken off. The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack. The two operations of a stack are the insertion and deletion of items. The operation of insertion is called push. The operation of deletion is called pop.

Register Stack

A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure 6 shows the organization of a 64-word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C, in that order. Item C is on top of the stack so that the content of SP is now 3. To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack. Note that item C has been read out but not physically removed. This does not matter because when the stack is pushed, a new item is written in its place.

The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items. DR is the data register that holds the binary data to be written into or read out of the stack.

Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation. The push operation is implemented with the following sequence of microoperations:

DR <--M [SP] Read item from the top of stack

SP <--SP – 1 Decrement stack pointer

If (SP = 0) then (EMTY <--1) Check if stack is empty

FULL <--0 Mark the stack not full

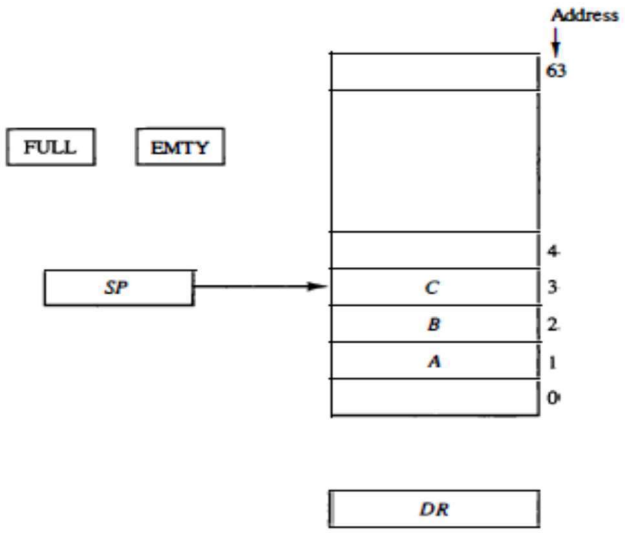


Fig.6 Block diagram of a 64 word stack.

The top item is read from the stack into DR. The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1. This condition is reached if the item read was in location L Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP.

Memory Stack:

A stack can exist as a stand-alone unit as in Fig. 6 or can be implemented in a random-access memory attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. Fig 7 shows a portion of computer memory partitioned into three segments: program, data, and stack. The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data. The stack pointer SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory. PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or pop items into or from the stack. As shown in Fig.7, the initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack is 3000. No provisions are available for stack limit checks.

We assume that the items in the stack communicate with a data register DR. A new item is inserted with the push operation as follows

SP <-SP - 1

M [SP] <-DR

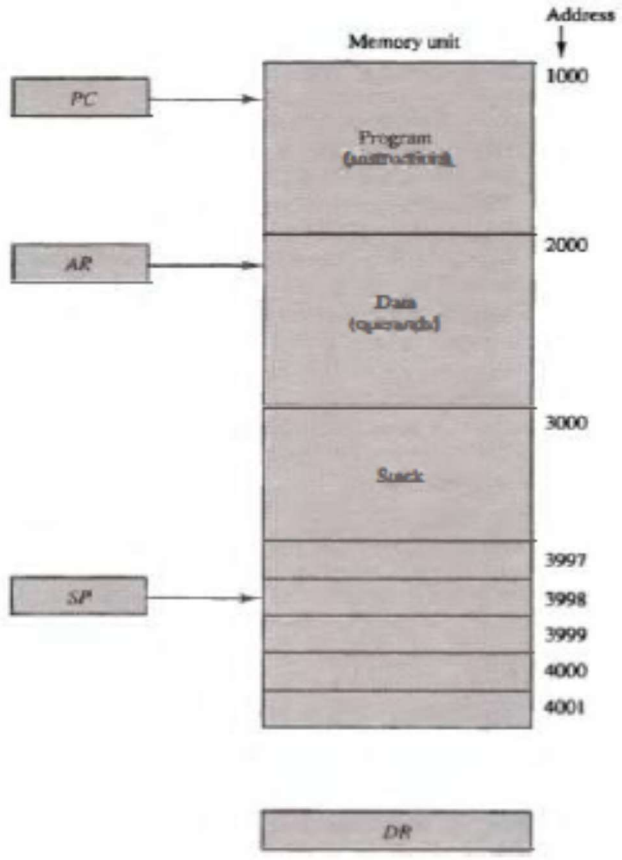


Fig .7 Computer memory with Program, data, and sack segments

The stack pointer is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack. A new item is deleted with a pop operation as follows:

DR <-M [SP]

SP <-SP + 1

Instruction Formats:

The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

1. An operation code 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.

The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift.

Three-Address Instructions

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) • (C + D) is shown below, together with comments that explain the register transfer operation of each instruction.

```
ADD     R 1, A, B    R 1 <--M [A] + M [B]
ADD     R2, C, D     R2 <--M [C] + M [D]
MUL     X, R 1, R 2  M [X] <--R 1 • R 2
```

It is assumed that the computer has two processor registers, R 1 and R2. The symbol M [A] denotes the operand at memory address symbolized by A. The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

Two-Address Instructions

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. The program to evaluate X = (A + B) • (C + D) is as follows:

```
MOV     R 1, A   R 1 <--M [A]
ADD     R 1, B   R 1 <--R 1 + M [B]
```

MOV R 2, C R2 <--M [C]

ADD R2, D R2 <--R 2 + M [D]

MUL R1, R2 R 1 <--R 1 • R 2

MOV X, R 1 M [X] <--R 1

The MOV instruction moves or transfers the operands t o and from memory and processor registers. The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

One-Address Instructions

One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. The program to evaluate

X = (A + B) • (C + D) is

L O A D A AC <- M [A]

A DD B A C <- A C + M [B]

S T O R E T M [T] <- A C

L O A D C A C <- M [C]

A D D D A C <- A C + M [D]

M U L T A C <- A C • M [T]

S T O R E X M [X] <- A C

Zero-Address Instructions

A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A + B) • (C + D) will be written for a stack organized computer. (TOS stands for top of stack.)

P U S H A T O S <- 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

Computer Registers

The memory unit has a capacity of 4096 words and each word contains 16 bits. Twelve bits of an instruction word are needed to specify the address of an operand. This leaves three bits for the operation part of the instruction and a bit to specify a direct or indirect address. The data register (DR) holds the operand read from memory. The accumulator (AC) register is a general purpose processing register. The instruction read from memory is placed in the instruction register (IR). The temporary register (TR) is used for holding temporary data during the processing.

The memory address register (AR) has 12 bits since this is the width of a memory address. The program counter (PC) also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed. The PC goes through a counting sequence and causes the computer to read sequential instructions previously stored in memory. Instruction words are read and executed in sequence unless a branch instruction is encountered. A branch instruction calls for a transfer to a nonconsecutive instruction in the program. The address part of a branch instruction is transferred to PC to become the address of the next instruction. To read an instruction, the content of PC is taken as the address for memory and a memory read cycle is initiated. PC is then incremented by one, so it holds the address of the next instruction in sequence. Two registers are used for input and output. The input register (INPR) receives an 8-bit character from an input device. The output register (OUTR) holds an 8-bit character for an output device.

Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

Fig .8 List of Registers for the Basic Computer

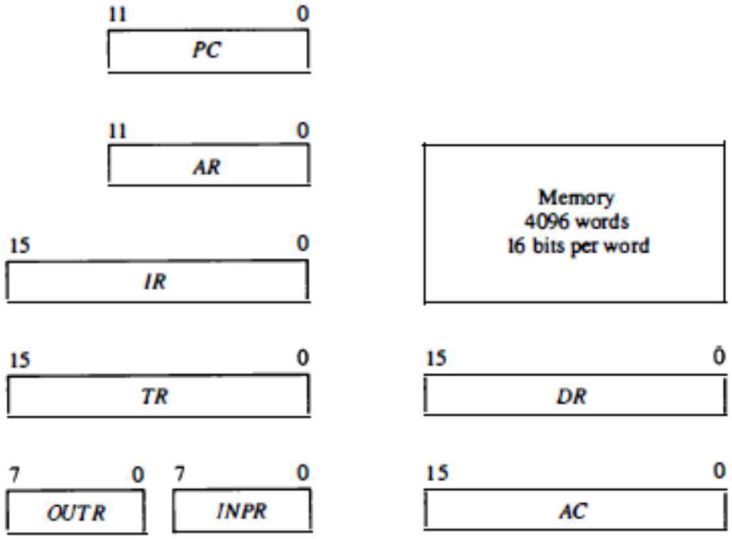


Figure 9. Basic computer registers and memory.

Computer Instructions:

The basic computer has three instruction code formats, as shown in Fig. 10. Each format has 16 bits. The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.

- A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address. The register reference instructions are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction.

- A register-reference instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed.
- A input-output instruction does not need a reference to memory and is recognized by the operation code Ill with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.

The type of instruction is recognized by the computer control from the four bits in positions 12 through 15 of the instruction. If the three opcode bits in positions 12 through 14 are not equal to 111, the instruction is a memory-reference type and the bit in position 15 is taken as the addressing mode I. If the 3-bit opcode is equal to 111, control then inspects the bit in position 15. If this bit is 0, the instruction is a register-reference type. If the bit is 1, the instruction is an input -output type. Note that the bit in position 15 of the instruction code is designated by the symbol I but is not used as a mode bit when the operation code is equal to 111.

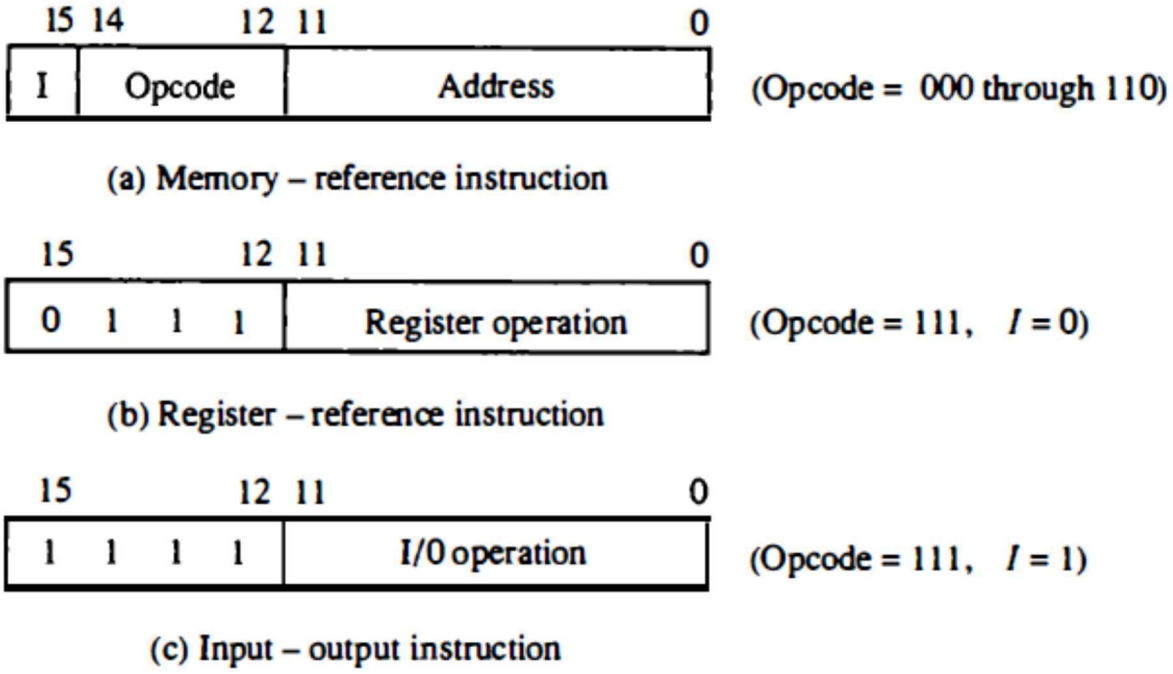


Fig 10. Basic computer instruction formats.

Symbol	Hexadecimal code $I = 0$	$I = 1$	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Fig 11.Basic Computer Instructions

Stack Organization:

Stack is a storage device that stores information in a way that the item is stored last is the first to be retrieved (LIFO). Stack in computers is actually a memory unit with address register (stack pointer SP) that can count only. SP value always points at top item in stack.

The two operations done on stack are,

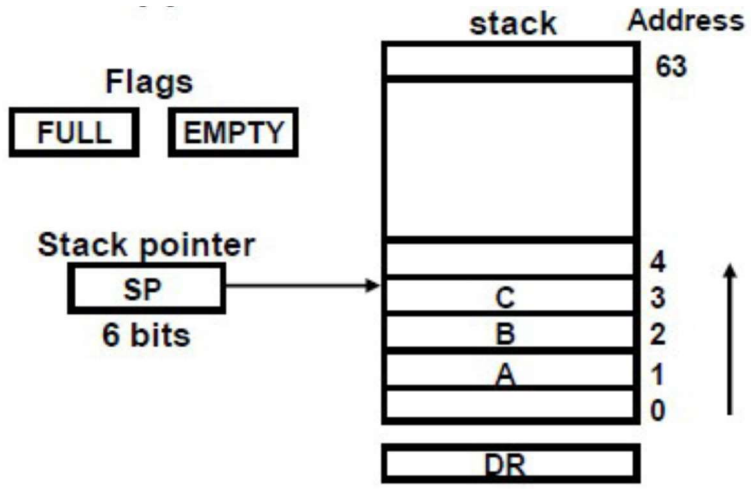
PUSH (Push Down), operation of insertion of items into stack

POP (Pop Up), operation of deletion item from stack

Those operation are simulated by INC and DEC stack register (SP).

1. Register stack:

A stand alone unit that consists of collection of finite number of registers. The next example shows 64 location stack unit with SP that stores address of the word that is currently on the top of stack.



Note that 3 items are placed in the stack A, B, and C. Item C is in top of stack so that SP holds 3 which the address of item C. To remove top item from stack (popping stack) we start by reading content of address 3 and decrementing the content of SP. Item B is now in top of stack holding address 2.

To insert new item (pushing the stack) we start by incrementing SP then writing a new word where SP now points to (top of stack).

Note that in 64-word stack we need to have SP of 6 bits only (from 000000 to 111111). If 111111 is reached then at next push SP will be 000000, that is when the stack is FULL. Similarly, when SP is 000001 then at next pop SP will go to 000000 that is when the stack is EMTY.

Initially, SP = 0, EMPTY = 1, FULL =0

Procedures for pushing stack

SP <- SP + 1

M[SP] <- DR

IF (SP = 0) THEN (FULL = 1)

EMTY <- 0

Note that:

1. Always we use DR to pass word into stack
2. M[SP] memory word specified by address currently in SP
3. First item stored in stack is at address 1
4. Last item stored in stack is at address 0. That is FULL = 1
5. Any push to stack means EMTY = 0

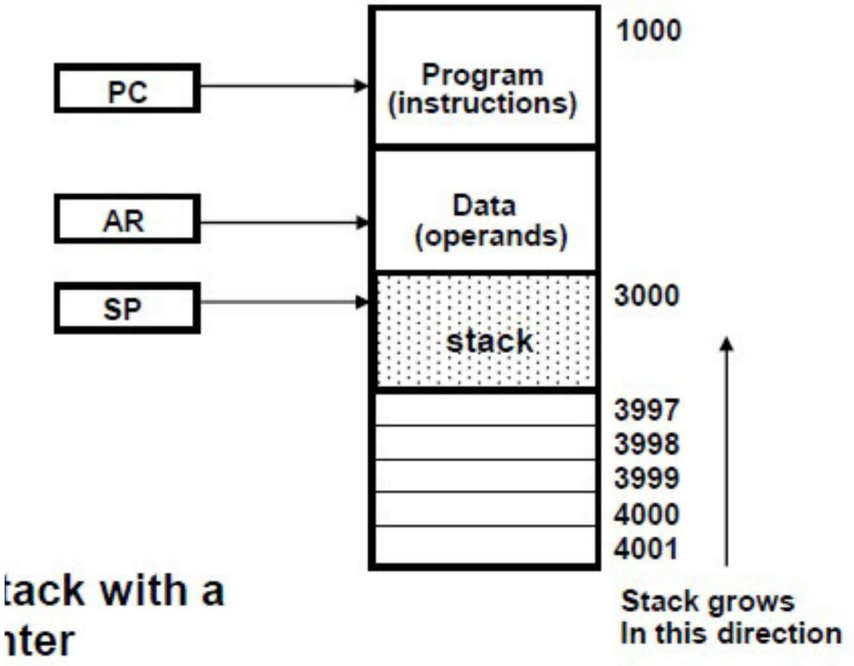
2. Memory Stack :
   - Stack can be implemented in RAM memory attached to CPU. Only by assigning special part of it for stack operations.
   - Next figure shows of main memory divided into program, data, and stack.
   - PC points to next instruction in instruction part
   - AR points to array of data of operands
   - SP points to top of stack All are connected to common address bus
   - Stack grows (pushed) with decreasing address and empties (pops) with increasing address.
   - New item is inserted with push operation by decrementing SP then a write to SP address is done

     SP <-SP -1

     M [SP]<- DR
   - Last item is removed from stack with pop operation by removing item by reading from memory location addressed by SP then SP is incremented.

     DR <- M [SP]

     SP <- SP +1



- As shown in figure initial value of SP is 4001 and first item when pushed in stack stores at address 4000 and second one stores at address 3999. The last address pushed into will be 3000. (See limitation danger?)
- Most computers are not supported by hardware to sense stack overflow and underflow. But can be implemented by saving the 2 limits in 2 registers. After each push or pop the SP is compared with the limit to see if stack has reached its limits. So must be taking care of using software.
- Always in this way we load SP with bottom address of stack portion of memory

Reverse Polish Notation:

- Very useful notation to utilize stacks to evaluate arithmetic expressions.

We write in infix notation such as:

$$A*B + C*D$$

We compute A*B, store product, compute C*D, then sum two products. So we have to scam back and forth to see which operation comes first.

The 3 notations to evaluate expressions

1. A + B Infix notation
2. +AB Prefix notation (Polish notation)
3. AB+ Postfix notation (reverse Polish

Reverse Polish Notation is in a form suitable for stack manipulation. Starts by scanning expression from left to right. When operator is found then perform

Instruction Format:

Operation with 2 operands in left of operator and replace result place of 2 operands and operator. Then you can continue this until you reach final answer.

Example

Expression A*B + C*D is written in RPN as AB*CD*+. And will computed as

(A*B) CD *+

(A*B)(C*D) +

Example

Convert infix notation expression (A + B)*(C * (D + E) + F) to RPN?

AB+ DE+ C * F+*.

Will be computed as

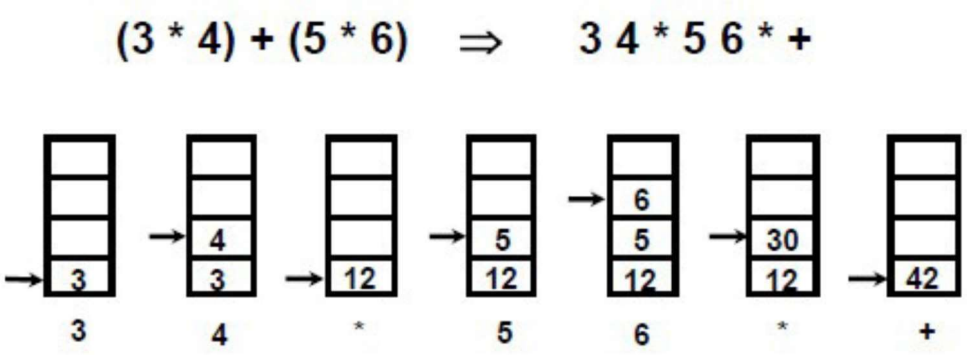
(A+B) (D+E) C * F + *

- Reverse polish notation combined with stack comprised of registers is most efficient way to evaluate expression. Stacks are good for handling long and complex problems involving chain calculations. But need first to convert arithmetic expressions into parenthesis-free reverse polish notation.
- This procedure is employed in some scientific calculators and some computers.

Example

Convert (3*4) (5*6) to RPN

34*56*+



The Instruction coding fields in today's computers follow the next format

1. Operation code field to specify operation
2. Address field that specifies operand address field or register
3. Mode field to specify effective address

In general, most processors are organized in one of 3 ways

- Single register (Accumulator) organization
  - Basic Computer is a good example
  - Accumulator is the only general-purpose register
- General register organization
  - Used by most modern computer processors
  - Any of the registers can be used as the source or destination for computer operations
- Stack organization
  - All operations are done using the hardware stack
  - For example, an OR instruction will pop the two top elements from the stack, do a logical OR on them, and push the result on the stack
- We are interested with address field of instructions with multiple address fields in instructions. The number of address fields in the instruction format depends on the internal organization of CPU. Some CPU combines features from more of one structure.

Instruction Cycle:

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of subcycles or phases. In the basic computer each instruction cycle consists of the following phases:

1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

Fetch and Decode:

Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal To. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2, and so on. The micro-operations for the fetch and decode phases can be specified by the following register transfer statements.

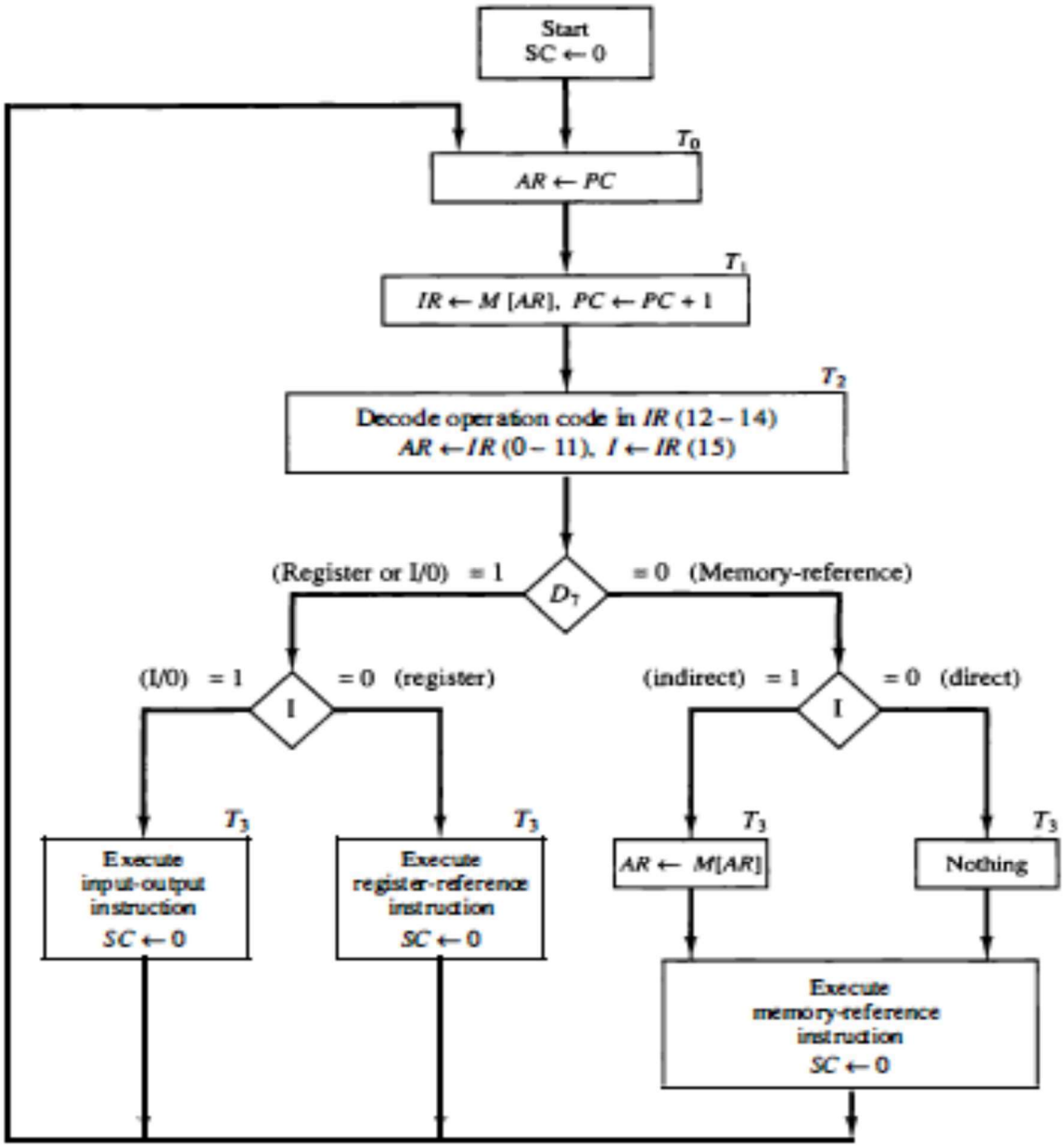
T0: AR <- PC

T,: IR <-M[AR], PC <- PC + 1

T2: D0, • • • , D7 <- Decode IR(12-14), AR <--- IR(0-11), 1 <--- IR(lS)

Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T0• The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1•

At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program. At time T2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR . Note that SC is incremented after each clock pulse to produce the sequence To, T1, and T2•



TEXT / REFERENCE BOOKS

1. M.Morris Mano, ;Computer System Architecture",Prentice-Hall Publishers,Third Edition.
2. John P Hayes , 'Computer Architecture and Organization', McGraw Hill international edition, Third Edition.
3. Kai Hwang and Faye A Briggs ,'Computer Architecture and Parallel Processing', McGraw Hill international edition,1995.

RISC Architecture

RISC (Reduced Instruction Set Computer) is used in portable devices due to its power efficiency. For Example, Apple iPod and Nintendo DS. RISC is a type of microprocessor architecture that uses highly-optimized set of instructions. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program Pipelining is one of the unique feature of RISC. It is performed by overlapping the execution of several instructions in a pipeline fashion. It has a high performance advantage over CISC.

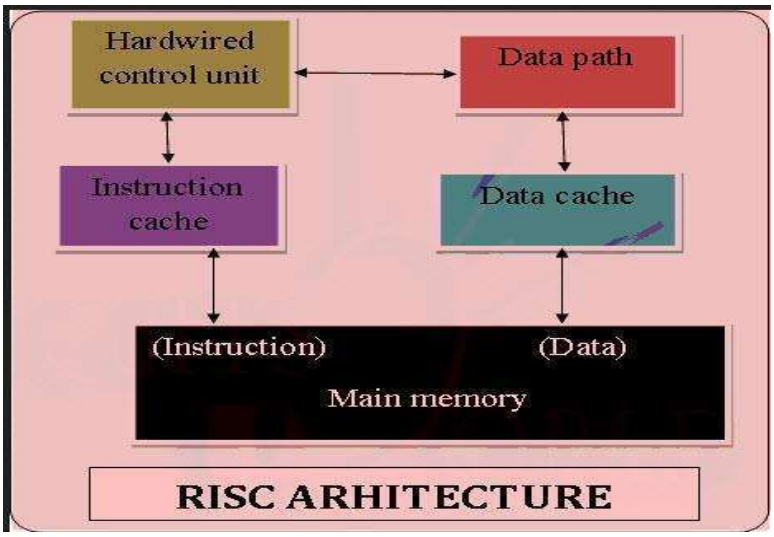


Figure : RISC Architecture

RISC ARCHITECTURE CHARACTERISTICS

Simple Instructions are used in RISC architecture. RISC helps and supports few simple data types and synthesize complex data types. RISC utilizes simple addressing modes and fixed length instructions for pipelining. RISC permits any register to use in any context. One Cycle Execution Time The amount of work that a computer can perform is reduced by separating "LOAD" and "STORE" instructions. RISC contains Large Number of Registers in order to prevent various number of interactions with memory. In RISC, Pipelining is easy as the execution of all instructions will be done in a uniform interval of time i.e. one click. In RISC, more RAM is required to store assembly level instructions. Reduced instructions need a less number of transistors in RISC. RISC uses Harvard memory model means it is Harvard Architecture. A compiler is used to perform the conversion operation means to convert a high-level language statement into the code of its form.

RISC & CISC COMPARISON

CISC	RISC
It is prominent on Hardware	It is prominent on the Software
It has high cycles per second	It has low cycles per second
It has transistors used for storing Instructions which are complex	More transistors are used for storing memory
LOAD and STORE memory-to-memory is induced in instructions	LOAD and STORE register-register are independent
It has multi-clock	It has a single - clock

MUL instruction is divided into three instructions
"LOAD" – moves data from the memory bank to a register
"PROD" – finds product of two operands located within the registers
"STORE" – moves data from a register to the memory banks
The main difference between RISC and CISC is the number of instructions and its complexity.

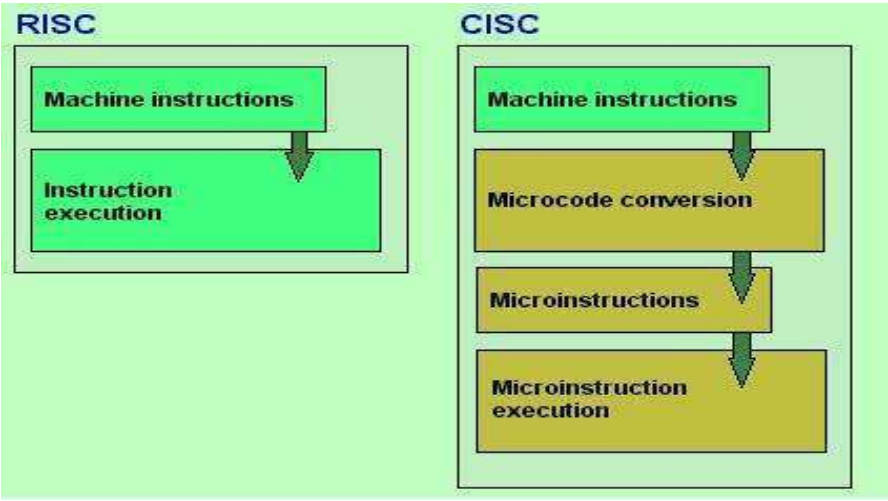
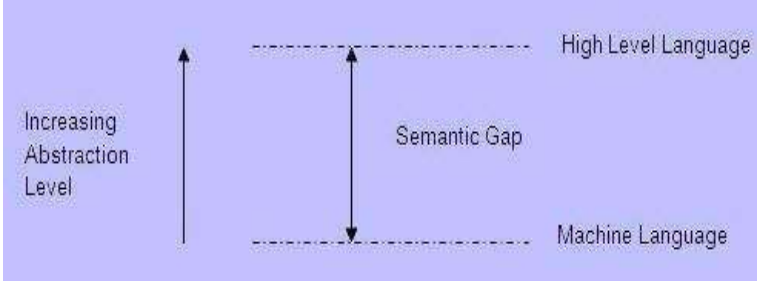


Figure: RISC & CISC COMPARISON

SEMANTIC GAP

Both RISC and CISC architectures have been developed as an attempt to cover the semantic gap.



With an objective of improving efficiency of software development, several powerful programming languages have come up, viz., Ada, C, C++, Java, etc. They provide a high level of abstraction, conciseness and power. By this evolution the semantic gap grows. To enable efficient compilation of high level language programs, CISC and RISC designs are the two options.

The features of RISC include the following

- The demand of decoding is less
- Uniform instruction set
- Few data types in hardware
- General purpose register Identical
- Simple addressing nodes

Advantages of RISC architecture:

- RISC(Reduced instruction set computing)architecture has a set of instructions, so high-level language compilers can produce more efficient code
- It allows freedom of using the space on microprocessors because of its simplicity.
- Many RISC processors use the registers for passing arguments and holding the local variables.
- RISC functions use only a few parameters, and the RISC processors cannot use the call instructions, and therefore, use a fixed length instruction which is easy to pipeline.
- The speed of the operation can be maximized and the execution time can be minimized. Very less number of instructional formats, a few numbers of instructions and a few addressing modes are needed.

The Disadvantages of RISC architecture:

- Mostly, the performance of the RISC processors depends on the programmer or compiler as the knowledge of the compiler plays a vital role while changing the CISC code to a RISC code

- While rearranging the CISC code to a RISC code, termed as a code expansion, will increase the size. And, the quality of this code expansion will again depend on the compiler, and also on the machine's instruction set.
- The first level cache of the RISC processors is also a disadvantage of the RISC, in which these processors have large memory caches on the chip itself. For feeding the instructions, they require very fast memory systems.

CISC Architecture

- The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. Computers based on the CISC architecture are designed to decrease the memory cost. Because, the large programs need more storage, thus increasing the memory cost and large memory becomes more expensive. To solve these problems, the number of instructions per program can be reduced by embedding the number of operations in a single instruction, thereby making the instructions more complex.

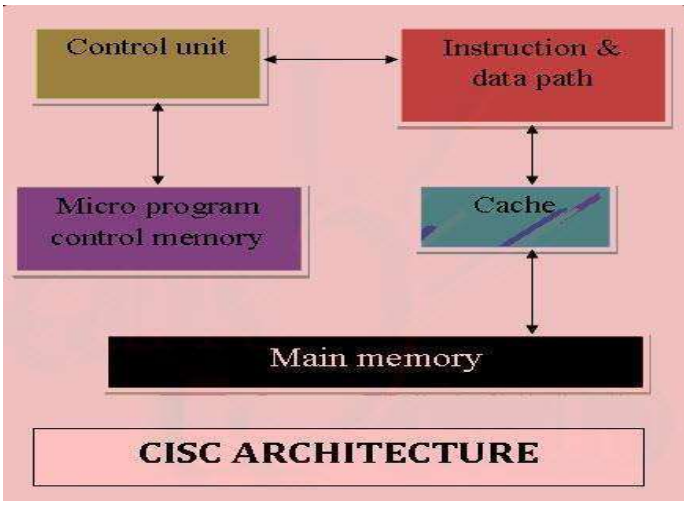


Figure: CISC Architecture

MUL loads two values from the memory into separate registers in CISC. CISC uses minimum possible instructions by implementing hardware and executes operations. Instruction Set Architecture is a medium to permit communication between the programmer and the hardware. Data execution part, copying of data, deleting or editing is the user commands used in the microprocessor and with this microprocessor the Instruction set architecture is operated. The main keywords used in the above Instruction Set Architecture are as below

Instruction Set: Group of instructions given to execute the program and they direct the computer by manipulating the data. Instructions are in the form – Opcode (operational code) and

Operand. Where, opcode is the instruction applied to load and store data, etc. The operand is a memory register where instruction applied.

Addressing Modes: Addressing modes are the manner in the data is accessed. Depending upon the type of instruction applied, addressing modes are of various types such as direct mode where straight data is accessed or indirect mode where the location of the data is accessed. Processors having identical ISA may be very different in organization. Processors with identical ISA and nearly identical organization is still not nearly identical.

CPU performance is given by the fundamental law

$$CPU\ Time = \frac{Seconds}{Program} = \frac{Instructions}{Program} X \frac{Cycles}{Instructions} X \frac{Seconds}{Cycle}$$

Thus, CPU performance is dependent upon Instruction Count, CPI (Cycles per instruction) and Clock cycle time. And all three are affected by the instruction set architecture.

Instruction Count of the CPU

	Instruction Count	CPI	Clock
Program	X		
Compiler	X	X	
Instruction Set Architecture	X	X	X
Microarchitecture		X	X
Physical Design			X

Examples of CISC PROCESSORS

IBM 370/168 – It was introduced in the year 1970. CISC design is a 32 bit processor and four 64-bit floating point registers.
VAX 11/780 – CISC design is a 32-bit processor and it supports many numbers of addressing modes and machine instructions which is from Digital Equipment Corporation.
Intel 80486 – It was launched in the year 1989 and it is a CISC processor, which has instructions varying lengths from 1 to 11 and it will have 235 instructions.

CHARACTERISTICS OF CISC ARCHITECTURE

Instruction-decoding logic will be Complex. One instruction is required to support multiple addressing modes. Less chip space is enough for general purpose registers for the instructions that are 0operated directly on memory. Various CISC designs are set up two special registers for the stack pointer, handling interrupts, etc. MUL is referred to as a "complex instruction" and requires the programmer for storing functions. CISC designs involve very complex architectures, including a large number of instructions and addressing modes, whereas RISC designs involve simplified instruction set and adapt it to the real requirements of user programs.

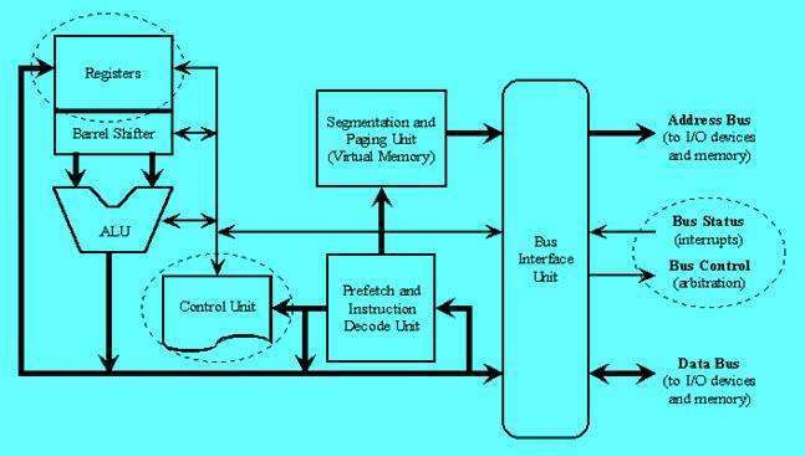


Figure: CISC and RISC Design

Advantages of CISC architecture

- Microprogramming is easy assembly language to implement, and less expensive than hard wiring a control unit.
- The ease of micro coding new instructions allowed designers to make CISC machines upwardly compatible:
- As each instruction became more accomplished, fewer instructions could be used to implement a given task.

Disadvantages of CISC architecture

- The performance of the machine slows down due to the amount of clock time taken by different instructions will be dissimilar
- Only 20% of the existing instructions is used in a typical programming event, even though there are various specialized instructions in reality which are not even used frequently.
- The conditional codes are set by the CISC instructions as a side effect of each instruction which takes time for this setting – and, as the subsequent instruction changes the

condition code bits – so, the compiler has to examine the condition code bits before this happens.

Memory Unit

RISC has no memory unit and uses a separate hardware to implement instructions. CISC has a memory unit to implement complex instructions

Program

RISC has a hard-wired unit of programming. CISC has a microprogramming unit

Design

RISC is a complex compiler design. CISC is an easy compiler design

Calculations

RISC calculations are faster and more precise. CISC calculations are slow and precise

Decoding

RISC decoding of instructions is simple. CISC decoding of instructions is complex

Time

Execution time is very less in RISC. Execution time is very high in CISC.

External memory

RISC does not require external memory for calculations. CISC requires external memory for calculations.

Pipelining

RISC Pipelining does function correctly. CISC Pipelining does not function correctly.

Stalling

RISC stalling is mostly reduced in processors. CISC processors often stall.

Code Expansion

Code expansion can be a problem in RISC whereas, in CISC, Code expansion is not a problem.

Disc space

Space is saved in RISC whereas in CISC space is wasted. The best examples of CISC instruction set architecture include VAX, PDP-11, Motorola 68k,And your desktop PCs on Intel's x86

architecture, whereas the best examples of RISC architecture include DEC Alpha, ARC, AMD 29k, Atmel AVR, Intel i860, Blackfin, i960, Motorola 88000, MIPS, PA-RISC, Power, SPARC, SuperH, and ARM too.