Chapter – 2
Central Processing Unit

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

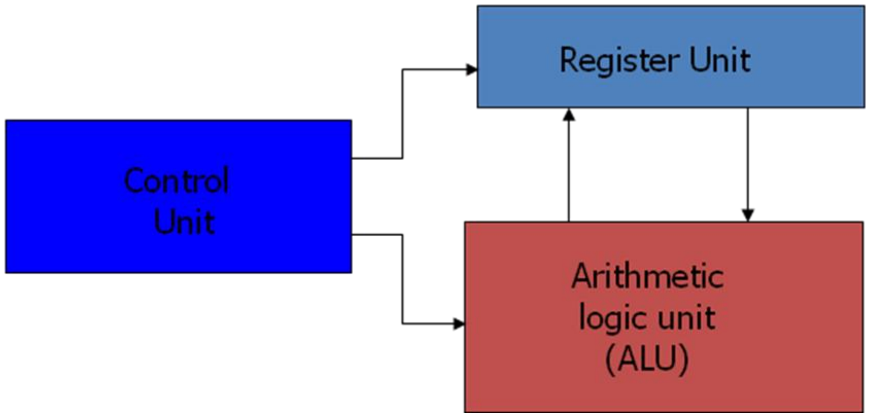


Fig: Components of CPU

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

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

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

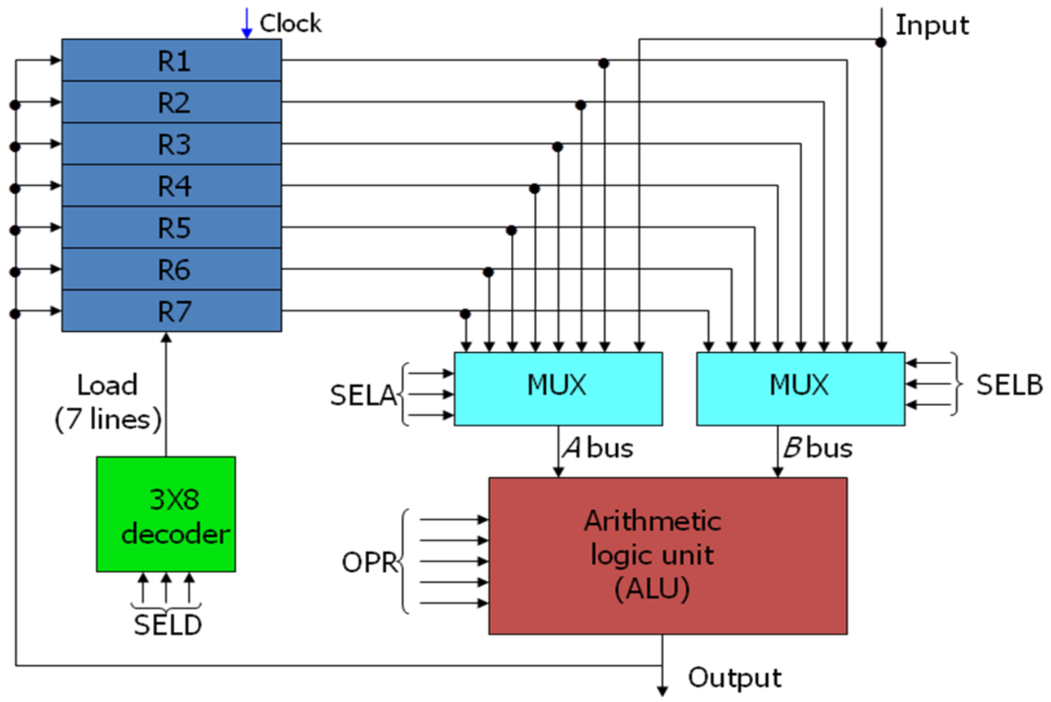


Fig: Processor Unit

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

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

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

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

The input to control unit are:

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

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

The outputs from control unit are:

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

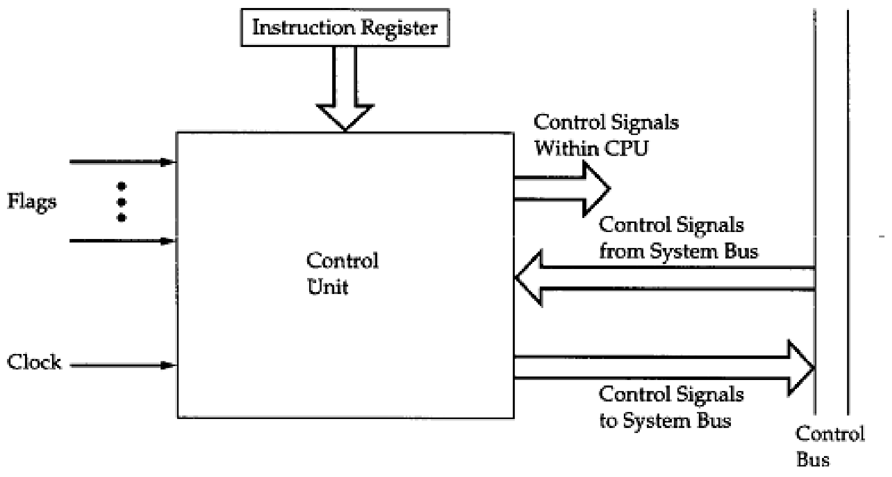

Instruction Register
Control Signals Within CPU
Flags
Control Signals from System Bus
Control Unit
Clock
Control Signals to System Bus
Control Bus

Fig: Control Unit

2.1 CPU Structure and Function

Processor Organization

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

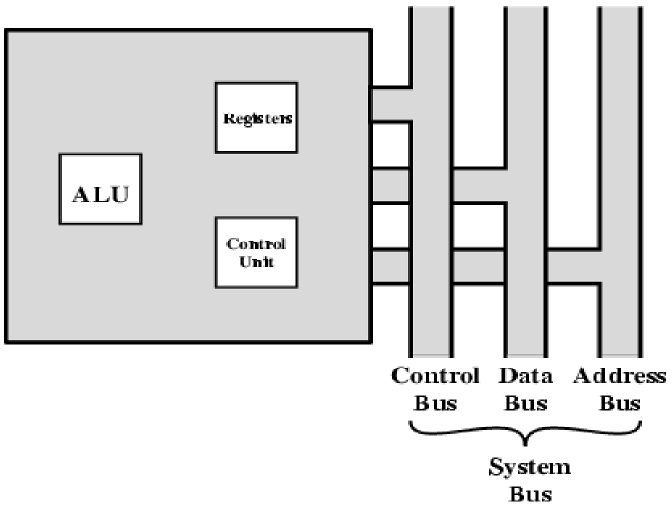


Fig: The CPU with the System Bus

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

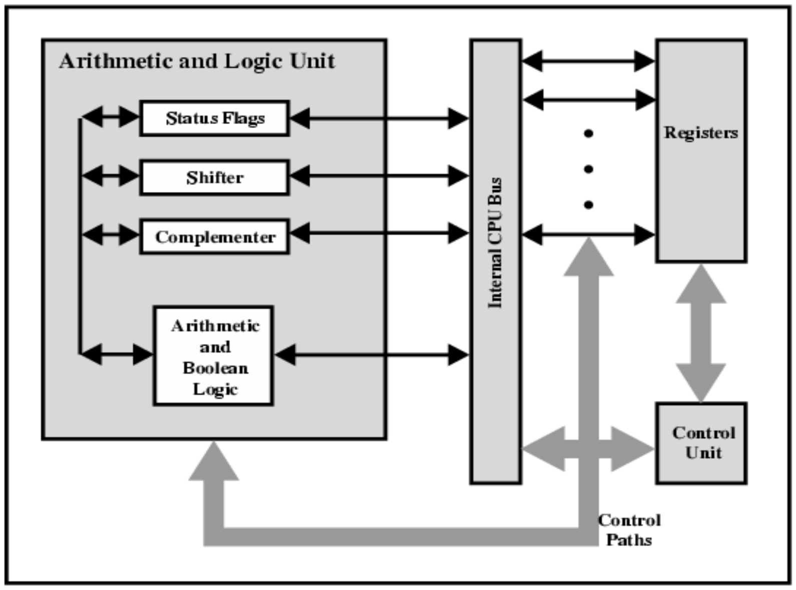


Fig: Internal Structure of the CPU

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

Register Organization

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

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

User-Visible Registers

Categories of Use

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

Design Issues

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

Control and Status Registers

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

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

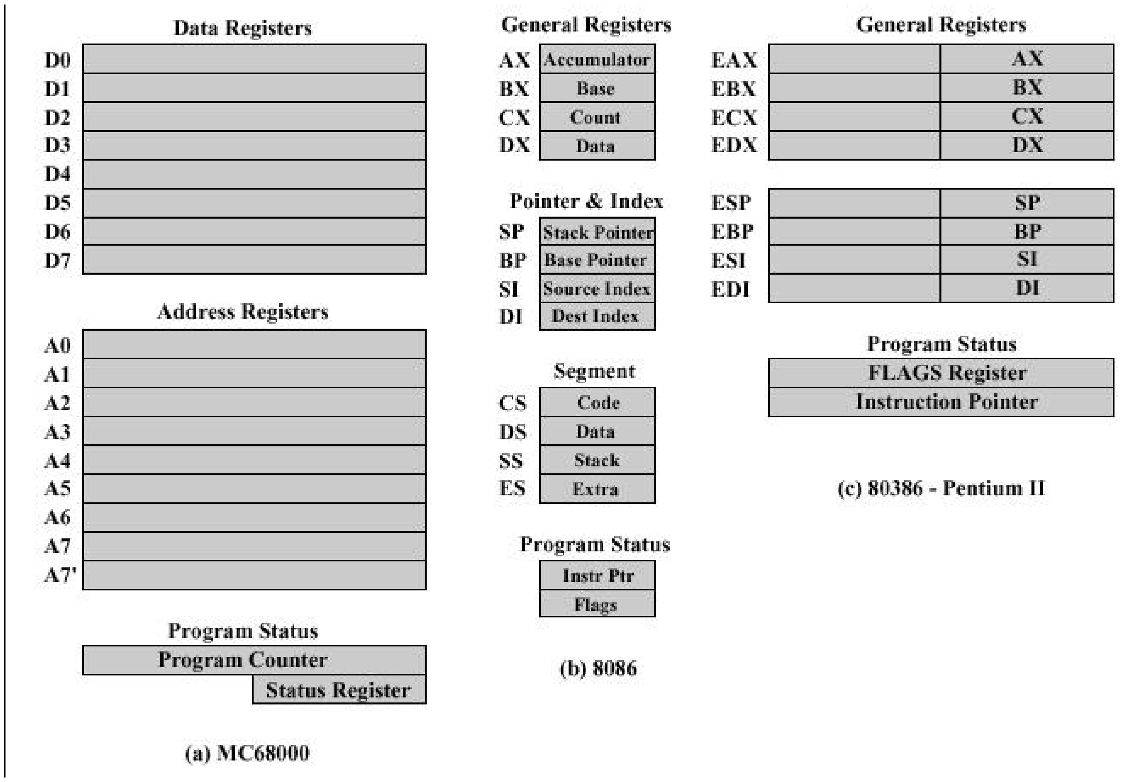


Fig: Example Microprocessor Register Organization

The Instruction Cycle

Basic instruction cycle contains the following sub-cycles.

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

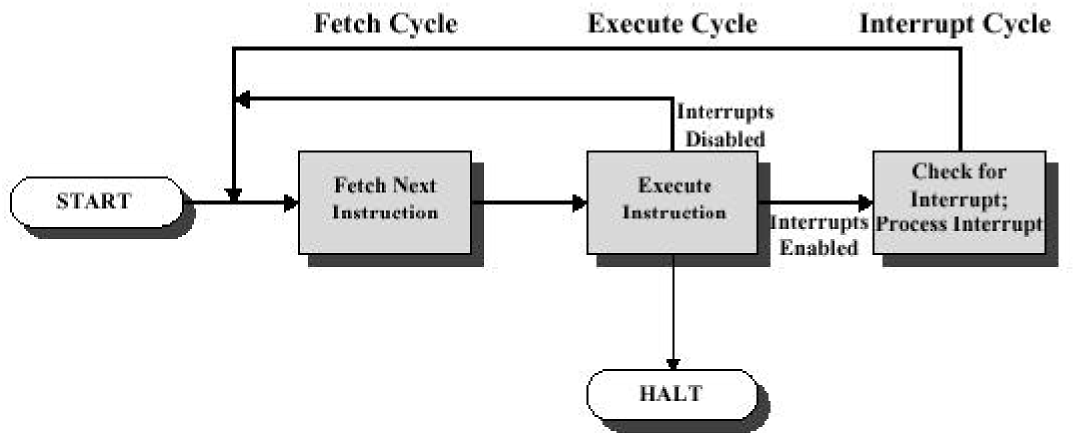


Fig: Instruction Cycles

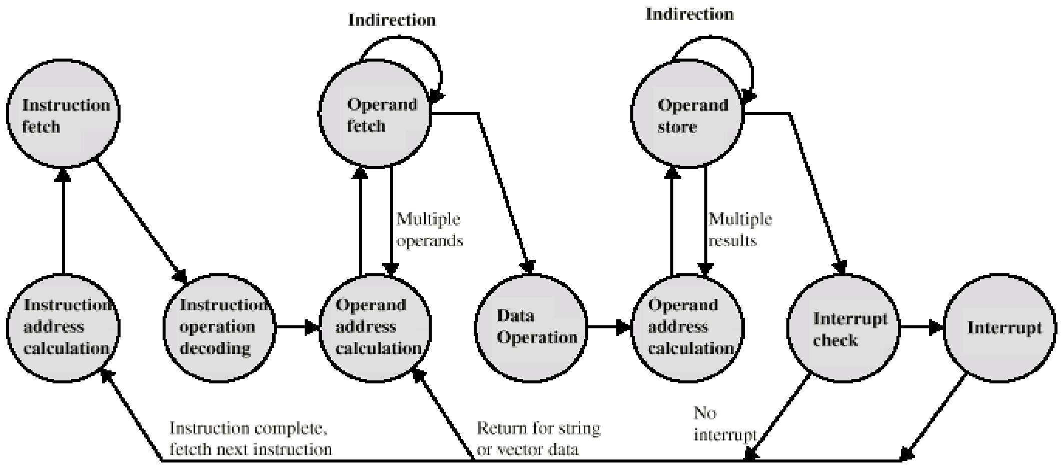

Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt

Fig: Instruction Cycle State Diagram

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

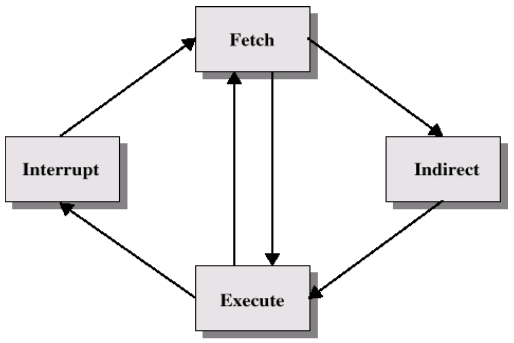


Fig: Instruction Cycle with Indirect

Data Flow

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

Fetch cycle data flow

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

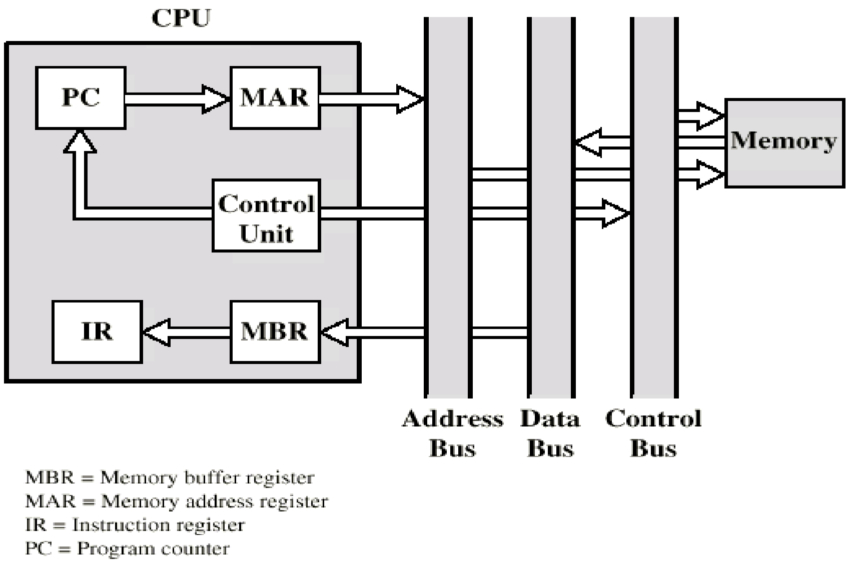

CPU
PC
MAR
Control Unit
IR
MBR
Memory
Address Bus
Data Bus
Control Bus
MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Fig: Data flow, Fetch Cycle

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

Indirect cycle data flow

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

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

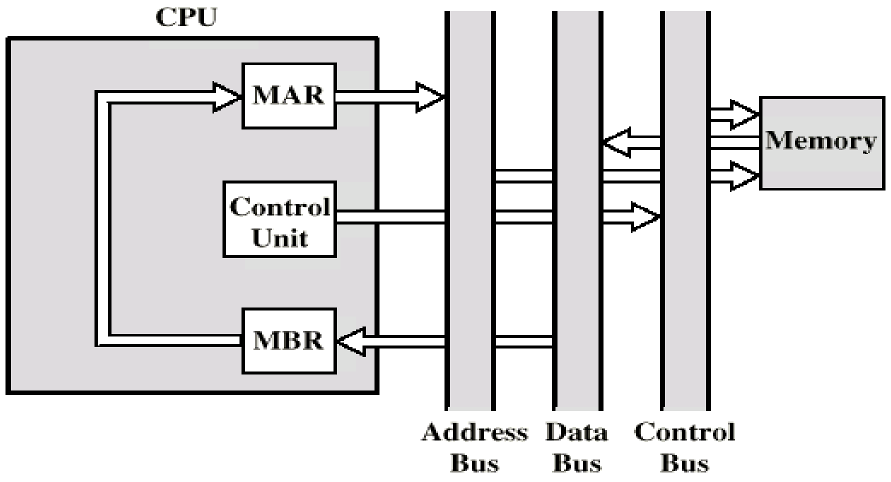


Fig: Data Flow, Indirect Cycle

Execute cycle data flow

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

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

Interrupt cycle data flow

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

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

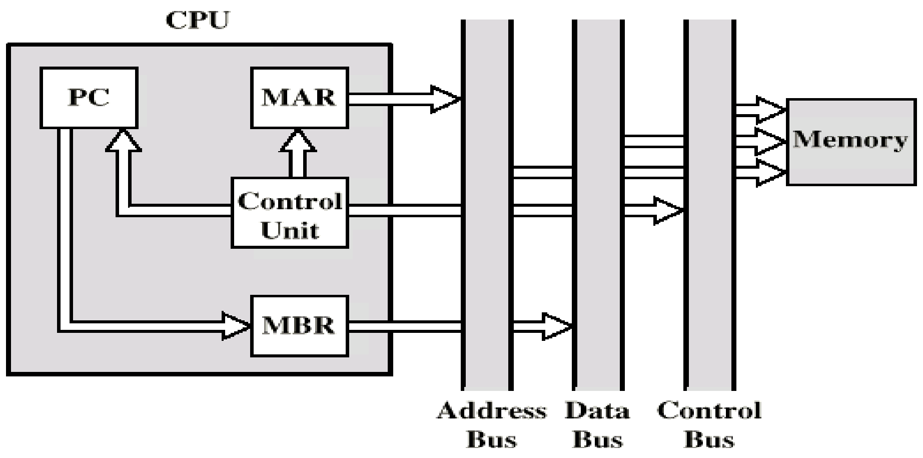
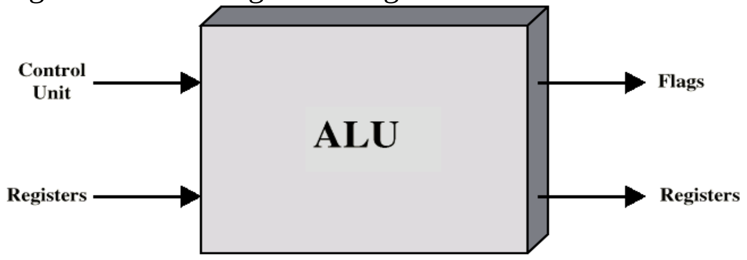


Fig: Data Flow, Interrupt Cycle

2.2 Arithmetic and Logic Unit

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



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

The design of ALU has three stages.

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

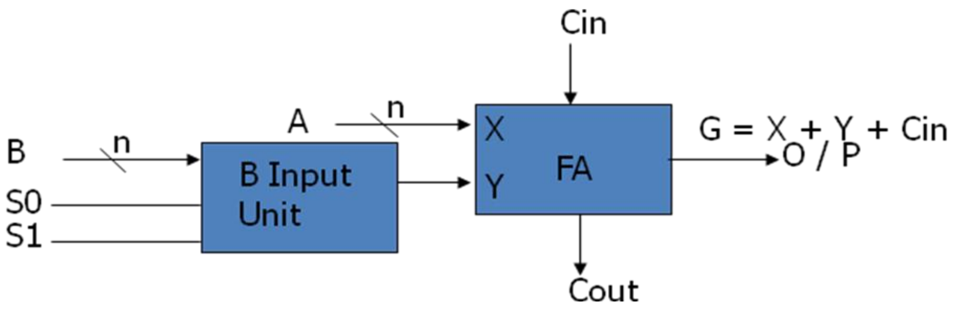


Fig: Block diagram of Arithmetic Unit

Functional table for arithmetic unit:

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

2. **Design the logical section**

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

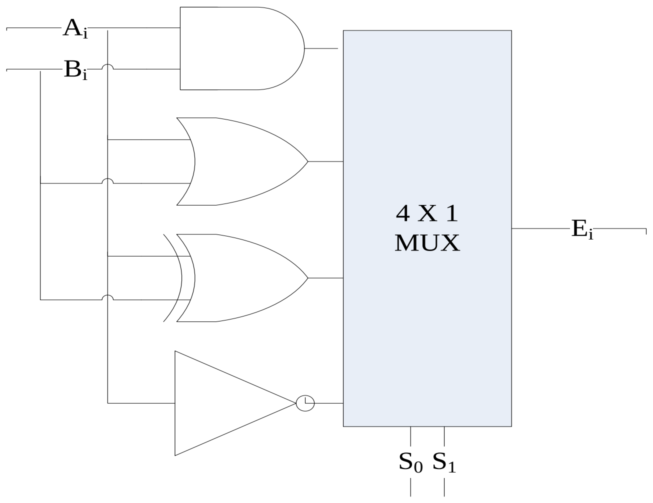


Fig: Block diagram of Logic Unit

Functional table for logic unit:

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

3. **Combine these 2 sections to form the ALU**

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

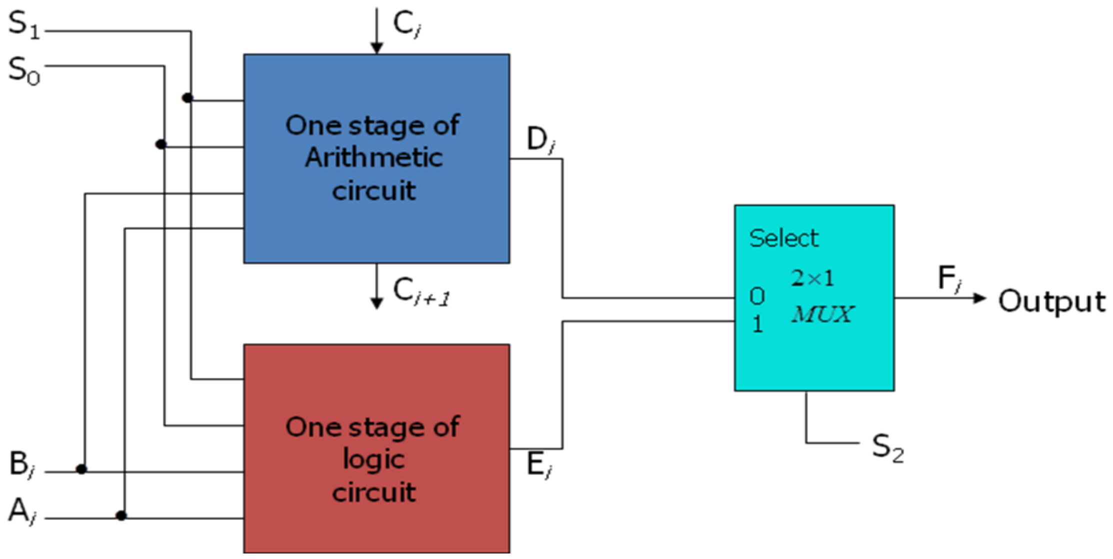
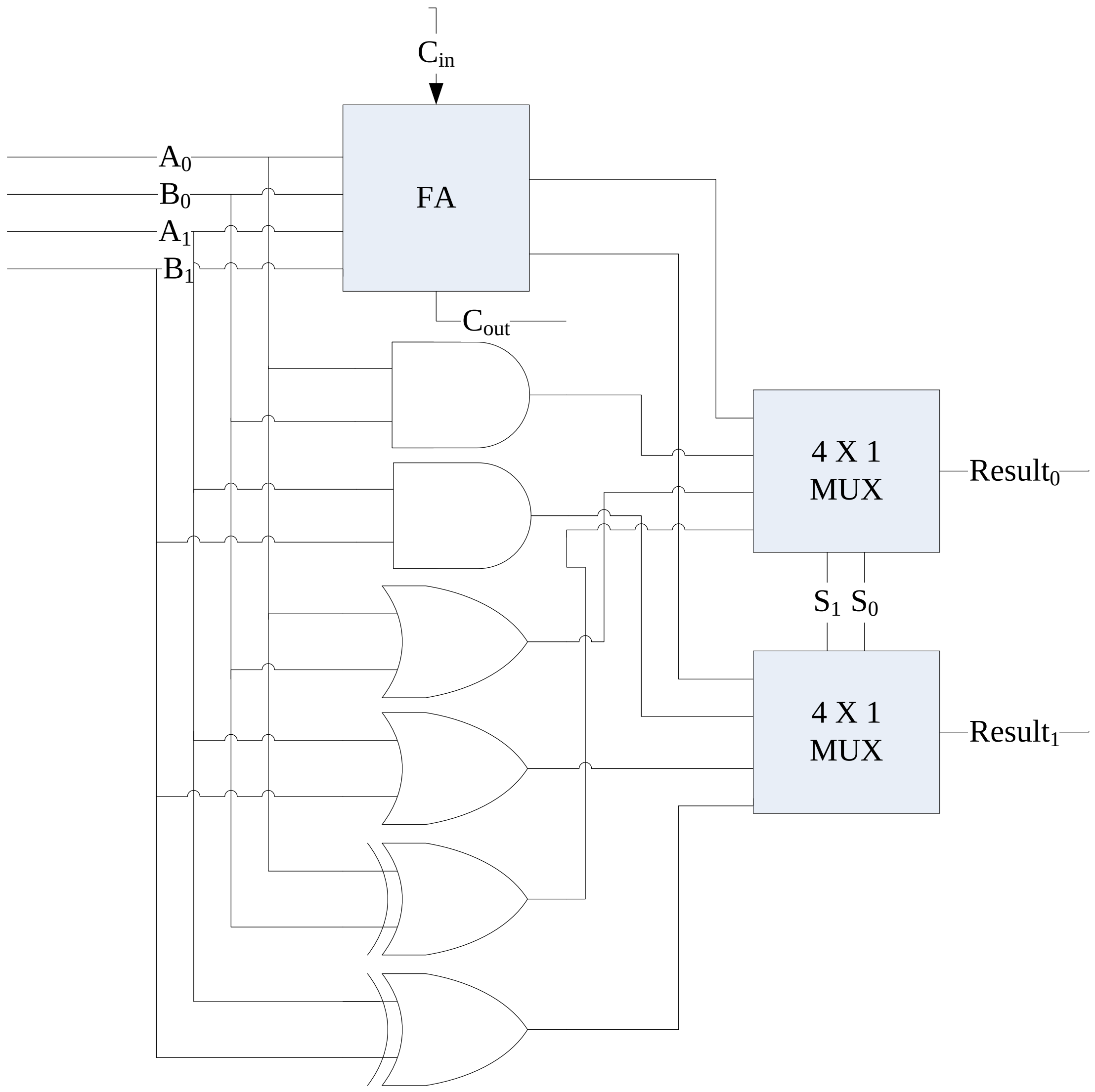


Fig: Block diagram of ALU

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



2.3 Instruction Formats

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

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

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

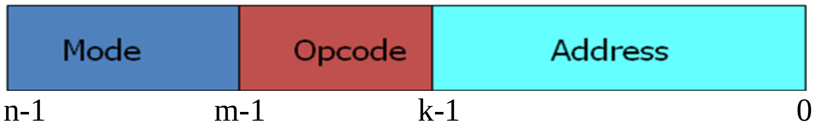


Fig: Instruction format with mode field

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

Types of Instruction

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

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

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

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

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

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

Example: ADD X, Y, Z; X ← Y + Z

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

2. **Two address instruction**

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

3. **One address Instruction**

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

4. **Zero address Instruction**

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

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

1. Three-Address Instructions:

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

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

2. Two-Address Instructions:

MOV R1, A;	R1 ← M[A]
ADD R1, B;	R1 ← R1 + M[B]

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

3. One-Address Instruction:

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

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

4. Zero-Address Instructions:

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

2.4 Addressing Modes

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

Effective address (EA):

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

The most well known addressing mode are:

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

Implied Addressing Mode:

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

Instruction

Opcode

Advantage: no memory reference. Disadvantage: limited operand

Immediate Addressing mode:

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

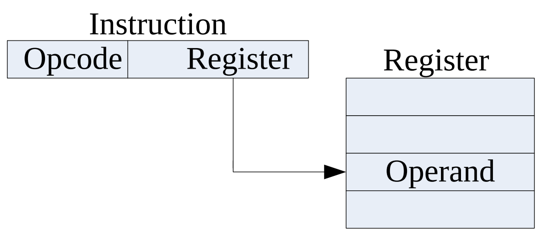
Instruction

Opcode	Operand

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

Register direct addressing mode:

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



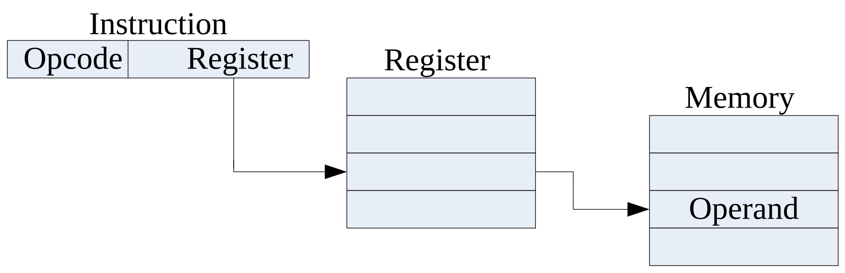
Effective Address (EA) = R

Advantage: no memory reference. Disadvantage: limited address space

Register indirect addressing mode:

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

  For example LDAX B



Effective Address (EA) = (R)

<u>Advantage:</u> Large address space.

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

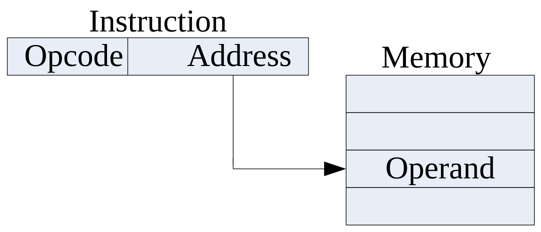
<u>Disadvantage:</u> Extra memory reference

Auto increment or Auto decrement Addressing Mode:

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

Direct Addressing Mode

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

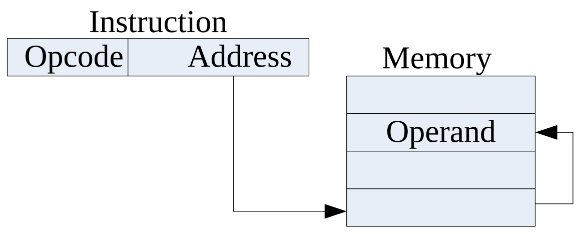


Effective Address (EA) = A

Advantage: Simple. Disadvantage: limited address field

Indirect Addressing Mode

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

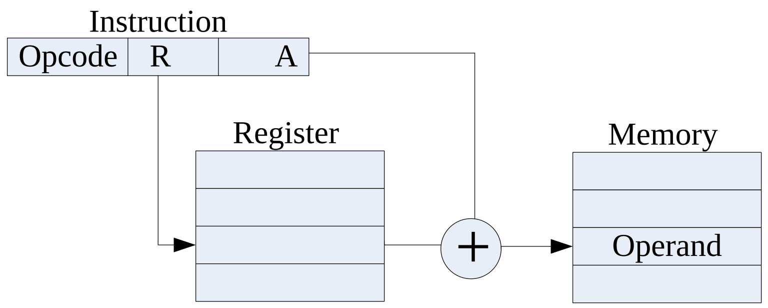


Effective Address (EA) = (A)

Advantage: Flexibility. Disadvantage: Complexity

Displacement Addressing Mode

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



Effective Address (EA) = A + (R)

Advantage: Flexibility. Disadvantage: Complexity

Relative Addressing Mode

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

Indexed Addressing Mode

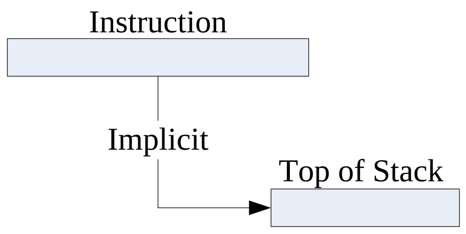
- In this mode the content of an index register (XR) is added to the address part of the instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value.
- Note: If an index-type instruction does not include an address field in its format, the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

Base Register Addressing Mode

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

Stack Addressing Mode

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



Effective Address (EA) = TOS

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

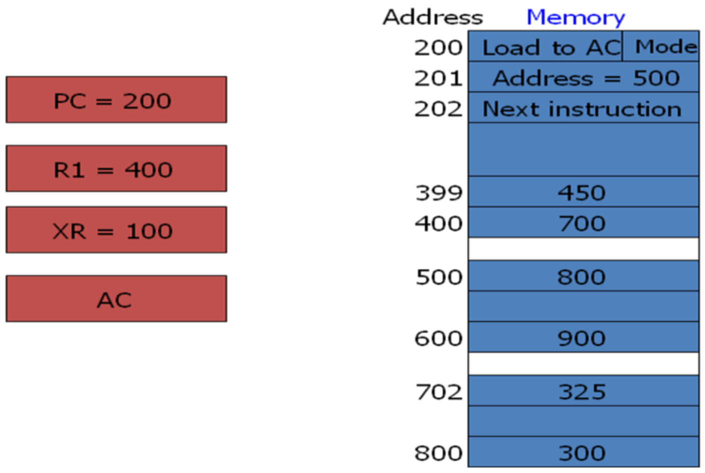


Fig: Numerical Example for Addressing Modes

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

Fig: Tabular list of Numerical Example

2.5 Data Transfer and Manipulation

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

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

Typical Data Transfer Instructions

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

Data manipulation Instructions

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

Arithmetic Instructions

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

Logical and Bit Manipulation Instructions

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

Shift Instructions

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

Program Control Instructions

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

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

Subroutine call and Return

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

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

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

Interrupt

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

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

2.6 RISC and CISC

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

Complex Instruction Set Computers (CISC)

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

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

The major characteristics of CISC architecture

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

Reduced Instruction Set Computers (RISC)

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

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

The major characteristics of RISC architecture

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

Other characteristics attributed to RISC architecture

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

Comparison between RISC and CISC Architectures

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

Overlapped register windows

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

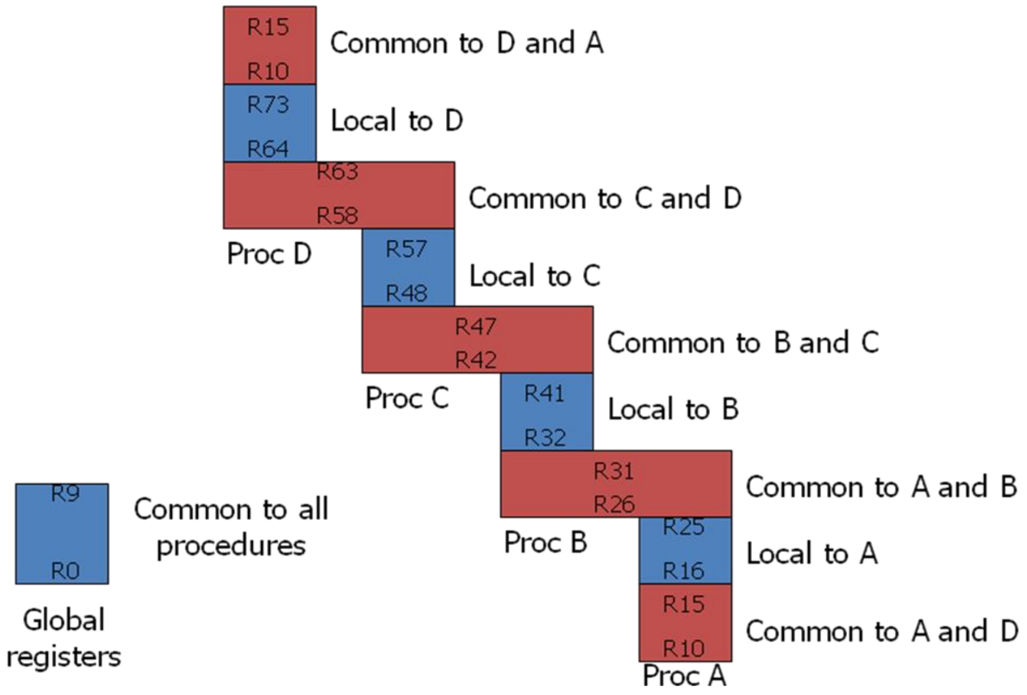


Fig: Overlapped Register Window

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

Berkeley RISC I

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

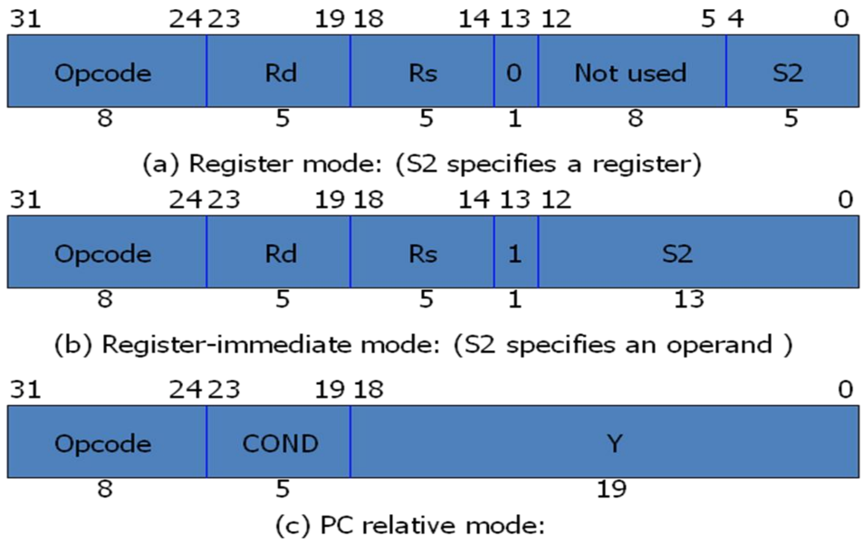
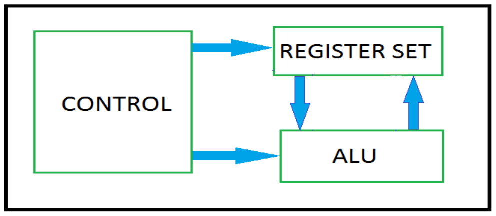


Fig: Instruction Format of Berkeley RISC I

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

2.7 64 – bit Processor

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



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

Internal Architecture

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

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

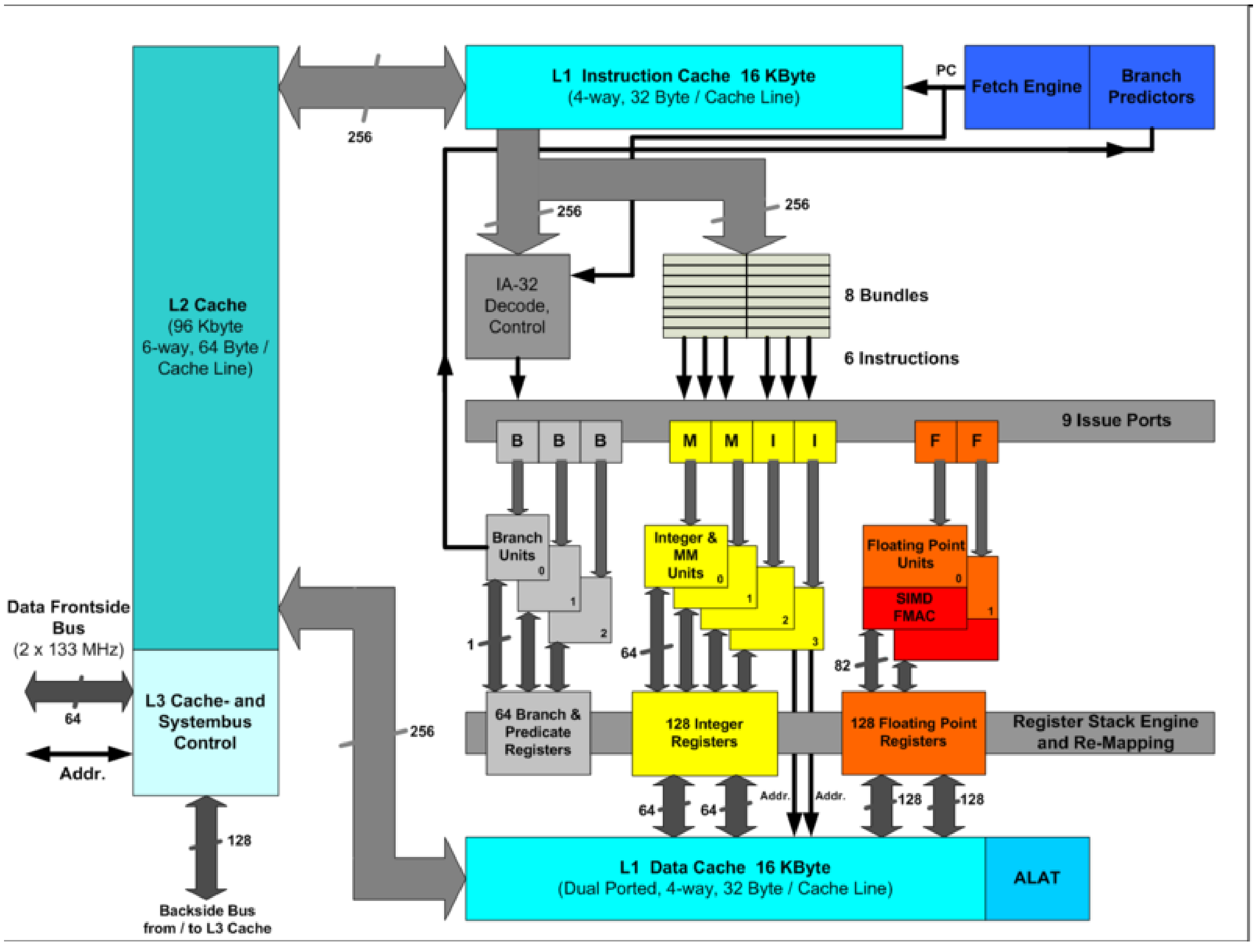


Fig: Block diagram of 64-bit internal architecture

Architecture Elements

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

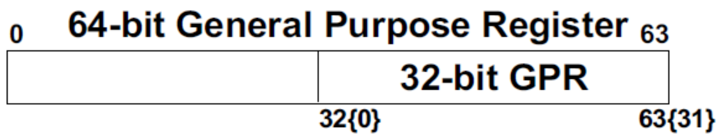
Addressing modes

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

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

General purposes register (GPR)

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



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

New instructions in 64-bit:

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

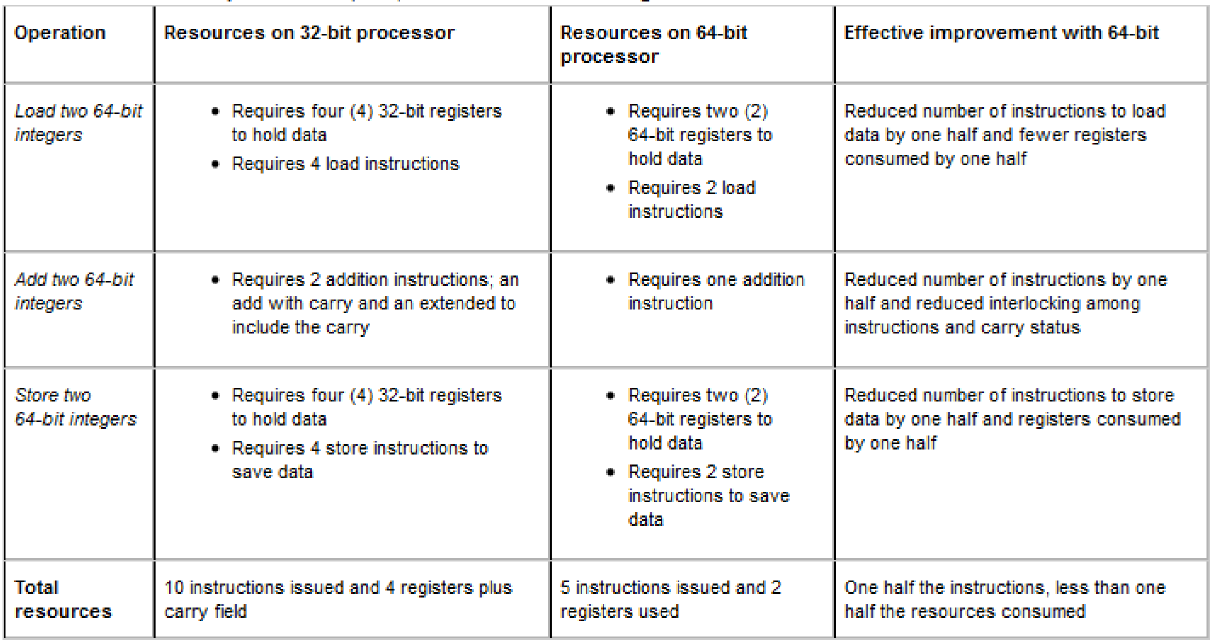
New immediate Instructions

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

Comparison of 64-bit with 32-bit

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

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



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

Advantages and disadvantages:

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