Chapter 8

Central processing unit

Central Processing Unit

- Performs the bulk of data-processing operations
- 3 major parts: Register set, ALU and Control Unit

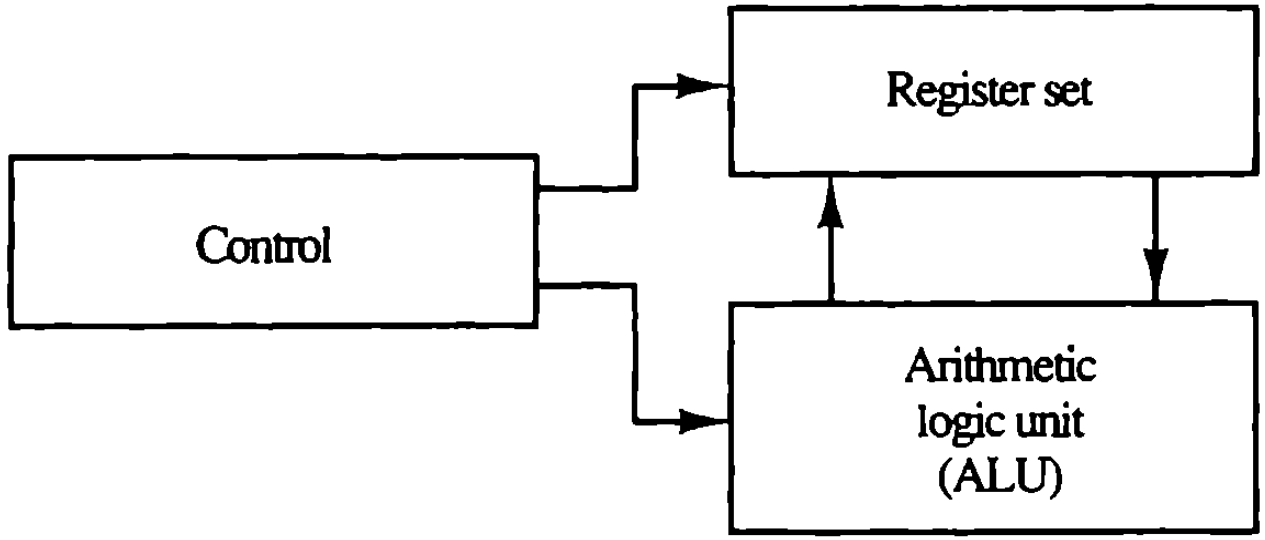
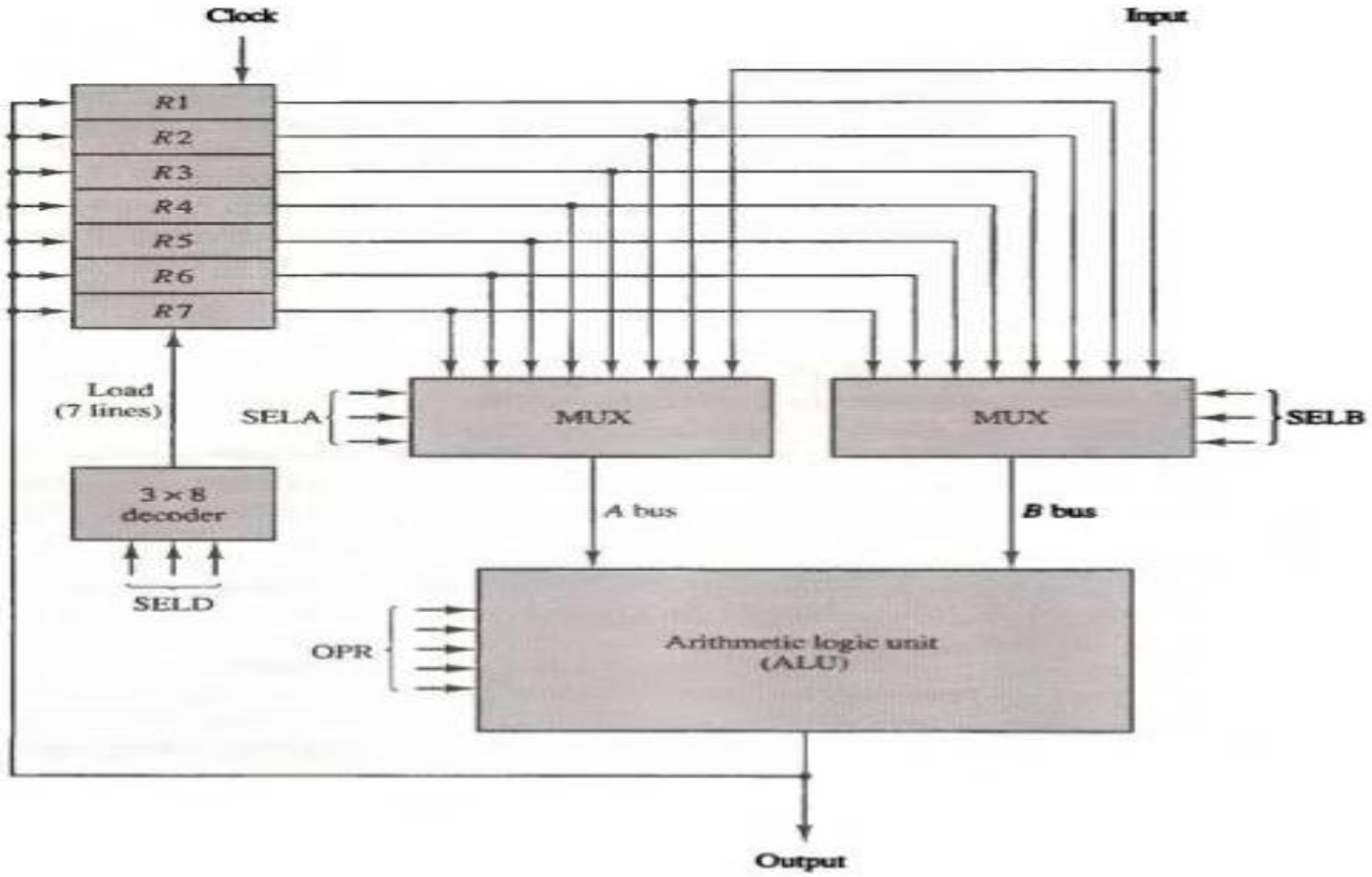
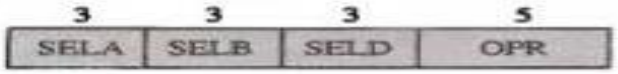


Figure 8-1 Major components of CPU.

General Register Organization



(a) Block diagram



(b) Control word

Figure 8-2 Register set with common ALU.

$$R1 \leftarrow R2 + R3$$

the control must provide binary selection variables to the following selector inputs:

1. MUX A selector (SELA): to place the content of $R2$ into bus A.
2. MUX B selector (SELB): to place the content of $R3$ 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 $R1$.

Control Word:

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

Ex: $R1 \leftarrow R2 - R3$

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	010 000 000 00000
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

Stack Organisation

- One of the important feature of CPU
- LIFO:- It is a storage device that store information like a stack of tray. Item that stored last is the memory is retrieved first.
- It is a memory unit with an address register.
- Register that holds the address for the stack is known as stack pointer.
- Stack Pointer:- It always point the top item of the stack.
- Note:- Stack is a physical register that is available for reading and writing the content. Only content of word is inserted or deleted.
- Operation of stack:-

1. Insertion- Push(push down) pushing or inserting a new item on top
2. Deletion- pop(pop up) removing one item from the stack

These operation is simulated by incrementing or decrementing the stack pointer register.

Register Stack

- Collection of Finite number of memory words or register.
- Fig. shows 64 words register stack
- SP- store the address of the word(in binary) that is currently on top of the stack.
- Fig shows 3 item in stacks A,B,C . Item C is in top

of the stack so SP stores the address of item C.

Content of SP = 3.

To remove the item , stack is popped up by reading the memory word at address 3 and SP= SP - 1.(decrement)

Now SP =2.

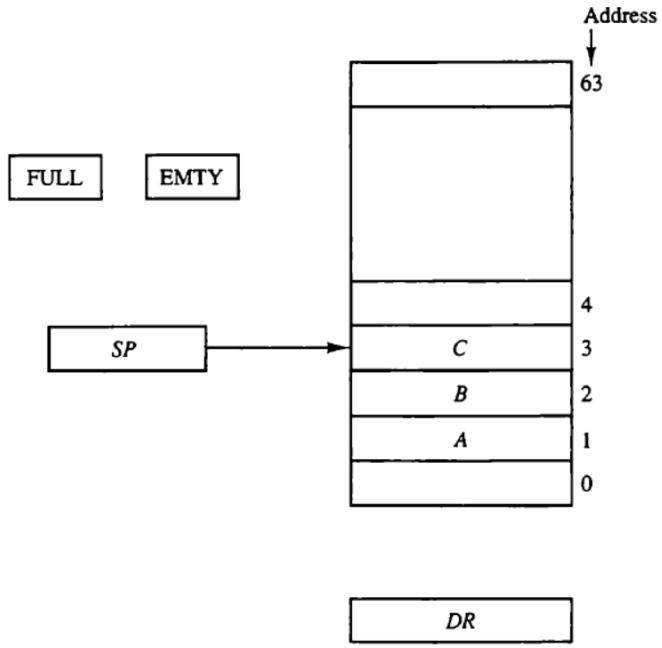


Figure 8-3 Block diagram of a 64-word stack.

- To insert a new item, stack is push up by incrementing SP = SP +1.
- Writing a new word in the next higher location.
- In a 64 word stack, SP = 6 bits each (2^6 =64)
- So, it cannot exceed 63 words(111111).
- When SP is incremented beyond 63 words, i.e.

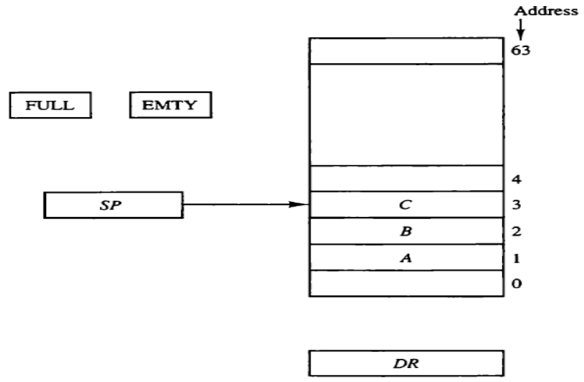


Figure 8-3 Block diagram of a 64-word stack.

63(111111) +1 (000001) = 0 (1000000) -> because SP can accommodates only 6 least significant bits (000000)

Similarly, 0(000000) – 1(000001) = 111111.

- 1-bit register FULL -> set to 1 (if Stack is full)
- 1-bit register EMPTY -> set to 1(if Stack is empty).
- DR -> data register hold the binary data written into or read out of stack.

Push Operation

- Initially, SP = 0, (Clear to Zero), EMTY -> 1, FULL -> 0
- SP points to word at address 0.
- If FULL= 0, new item can be inserted into the stack with push operation.
- Implementation of push operation by Following sequence of micro-operation:-

SP <- SP +1; INCREMENT STACK POINTER

M[SP] <- DR ; WRITE ITEM ON TOP OF STACK

If (SP = 0) then (FULL <- 1) ; CHECK IF STACK IS FULL

EMTY <- 0; MARK THE STACK IS NOT EMPTY

- 1st item stores at address 1.
- Last item stores at address 0.

Condition when SP = 63 and push operation perform then, SP = SP + 1 -> SP = 0. Now, Stack is FULL. So EMTY is cleared to 0.

POP Operation

- If stack != EMPTY , new item is deleted from the stack.
- Pop operation consist of following sequence of operation:-

DR <- M[SP]; READ ITEM FROM 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

Memory Stack

- A stack can exist as a stand alone memory or implemented in RAM attached to CPU.
- Implementation of stack in CPU is done by

a) Assigning portion of memory to a stack operation and

b) Use PR (processor register) = SP.

Fig shows portion of computer memory partition into

3 segments:-

1. Program,
2. Data and
3. Stack

PC -> points to address of next instruction in the program.

AR -> points to array of data

SP-> points top of the stack

3 register are connected to common address bus and either one can provide an address for memory.

PC-> used during fetch phase to read the instruction.

AR-> used during execution phase to read an operand.

SP-> used to perform pop or push item from or into stack.

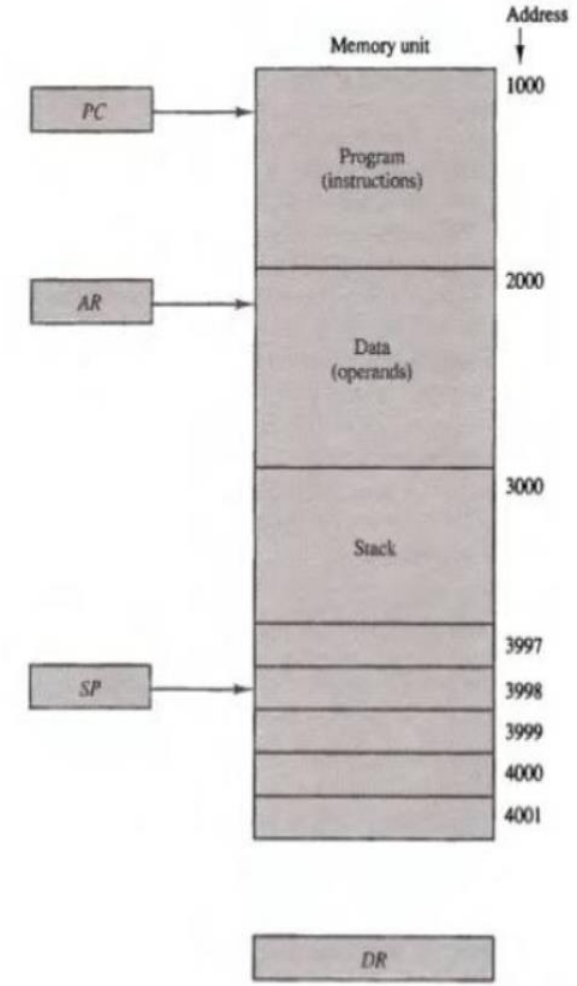


Figure 8-4 Computer memory with program, data, and stack segments.

- In fig. Initial value of SP is 4001.
- As shown stack grows with decreasing order.
- 1st item store at address 4000, 2nd at 3999 so on.
- No provision of stack limit check is available here.
- We assume stack communicate with DR.
- To insert new item push operation is perform as:-

  SP <- SP -1

  M[SP] <- DR; memory write operation insert word from DR to top of stack.

- To delete new item pop operation is perform as:-

  DR <- M[SP]

  SP <- SP +1; item read from top of stack to DR and increment SP.

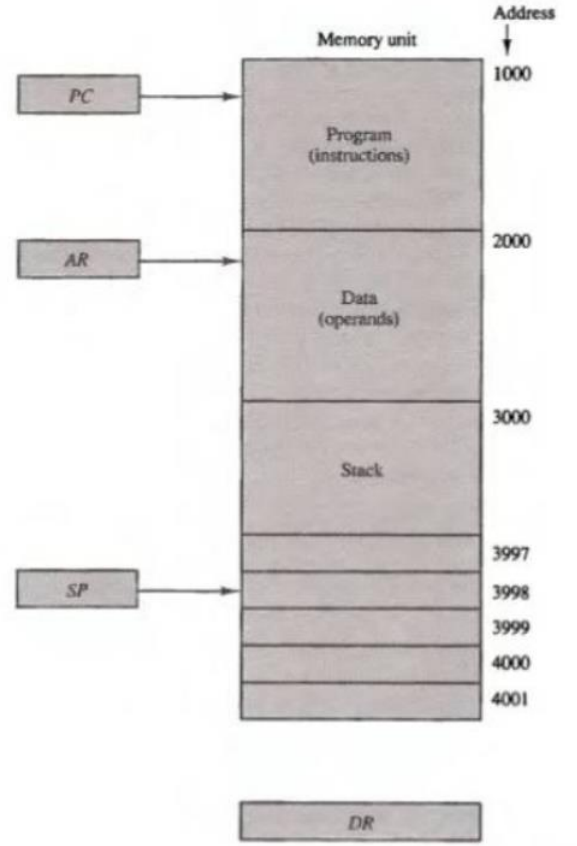


Figure 8-4 Computer memory with program, data, and stack segments.

Stack Limit

- Most computer not include H/W to check for stack overflow (Full stack) or underflow (empty stack).
- Stack limit can be check by using two processor register

1. to hold Upper limit(3000 in this case)

   After push operation SP compare with UL register

2. to hold Lower limit (4001 in this case)

   After pop operation SP compare with LL register.

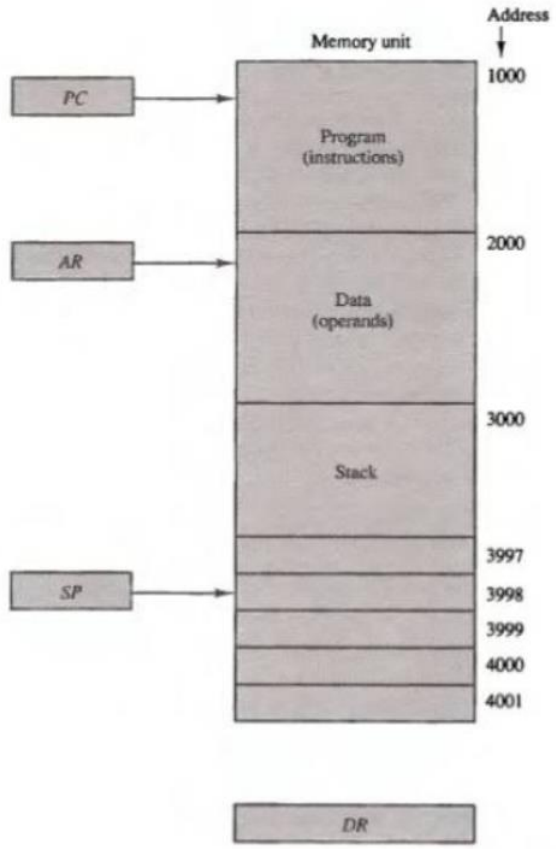


Figure 8-4 Computer memory with program, data, and stack segments.

Advantage of Stack Limit :-

CPU can easily refer to memory stack without having to specify an address, since the address is always available and updated automatically in the SP.

Reverse Polish Notation

- A Stack organisation is very effective for evaluating arithmetic expression.
- Common mathematical method of writing arithmetic expression imposes difficulties when evaluated by a computer.
- Common Arithmetic expressions are written in infix notation, with each operator written between the operands.

EX: A * B + C * D

Arithmetic expression can also be written in Postfix and prefix notation.

The following examples demonstrate the three representation:

A + B; Infix Notation

+ AB ; PREFIX or Polish Notation

AB +; Postfix or reverse polish notation

- A * B + C * D
- Step 1: AB * + C * D
- Step 2: AB * + CD*
- Step 3: AB * CD* +
- Evaluation of Arithmetic Expression:

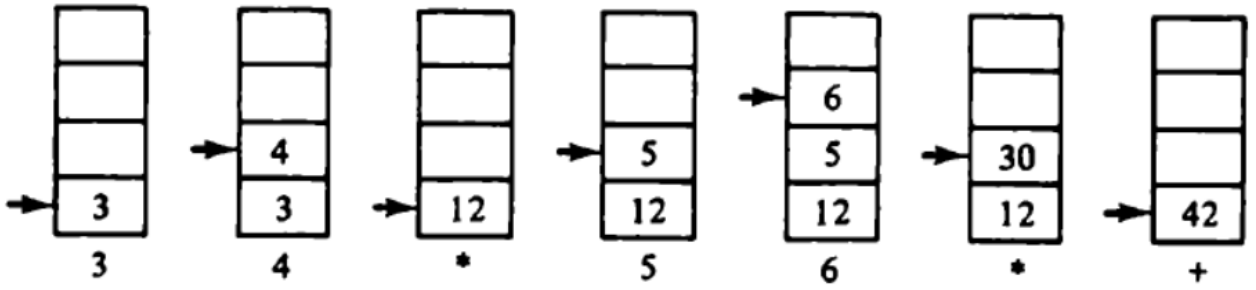
RPN combine with a stack arrangement of register is the most efficient way to evaluate arithmetic expressions.

Example to clarify the procedure:

(3*4) + (5*6)

1. Covert it to RPN
   34* 56 * +
2. Stack operation is performed
3. Each box represent one stack operation
4. -> point to top of stack

Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



Instruction format

- Symbolize the bit of instructions.
- The bits on instruction are divided into groups called fields.
- Most common fields in instruction formats are:

1. An operation code filed that specifies the operation to be performed.
2. An address field that designates a memory address or processor register.
3. A mode field that specifies the way operand or effective address is determined.

- Register Address: Operand residing in processor register are specified with a register address.
- A register address in a binary number of k- bit that define one of the 2^k register in a CPU.
- Example : CPU having 16 processor register R0 to R15 will have register address field of 4 bit each.

2^4 = 16

0101 designate to R5 register.

- Computer may have instruction of different length depends upon the internal organisation of its registers.
- Mainly 3 types of CPU organisation are used:

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

Single accumulator organization

- All operations are performed with implied accumulator register.
- Instruction format use one address field.
- Ex: ADD X

  X: address of the operand

  AC <- AC + M[X]

  AC: accumulator register

  M[X] : memory word located at address X.

General register organization

- Instruction format in this type of organization uses 3 register address fields.
- Ex: ADD R1, R2, R3

  Denotes R1 <- R2 + R3

No. of address field can be reduce from 3 to 2 if designation register is same as source register.

ADD R1, R2

denotes R1 <- R1 + R2

Stack Organization

- In stack organization PUSH and POP instruction has been used.
- Ex: PUSH X : will push the word at address X to top of the stack.
- Operation type of instruction in stack organization do not need an address field. Ex: ADD
- Some computer used the combination of 2 organization structures.

- Let us illustrate the influence of the number of addresses on computer program

  Ex: X = (A + B) * (C + D)

1. Three address instruction

- Computer with 3 address instruction uses each address field to specify processor register or memory operand.
- 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
- R1, R2 two processor register
- M[A] operand at memory location A.
- Advantage: Result in short program when evaluate arithmetic expression.
- Disadvantage: Binary coded instruction require too many bits to specify three address.

Two address instruction

- Most common in commercial computer
- Each instruction is specify either a processor register or memory word.
- Ex:

```
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 Instruction

- It uses a implied accumulator register for all data manipulation.
- Ex: All operations are done between the AC register and memory operands.

```
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

- Stack organisation does not use an address field for the instruction ADD , MUL.
- PUSH and POP is used.
- Ex:
- TOS: top of stack

```
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
```

- The name “Zero Address” is given because of the absence of an address field in computational instruction.

Assignment : Addressing mode

Data Transfer and Manipulation

- Computer instructions can be classified into 3 categories :

1. Data transfer instructions.
2. Data manipulation instructions.
3. Program control instructions.

DATA TRANSFER INSTRUCTIONS:

It cause transfer of data from one location to another without changing the binary information content/ data content.

The most common transfers are between

1. memory and processor registers,
2. Between processor registers and input or output,
3. Between the processor register themselves.

- Table 8-5 shows the eight data transfer instruction used in computer.

TABLE 8-5 Typical Data Transfer Instructions

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

TABLE 8-6 Eight Addressing Modes for the Load Instruction

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)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

Data manipulation instruction

- It perform operation on data and provide the computational capabilities for the computer.
- Data manipulation instruction typically divided into 3 basic types:

1. Arithmetic instruction
2. Logical and bit manipulation instruction
3. Shift instruction.

Arithmetic Instruction

TABLE 8-7 Typical 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 instruction

- Logical instruction perform binary operation on strings of bits stored in register.
- They are useful for manipulation of individual bits or group of bits.

TABLE 8-8 Typical 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

TABLE 8-9 Typical 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

- It is used to change the address value in program counter and also used to change the flow of control.

TABLE 8-10 Typical Program Control Instructions

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

Status Bit Register

- Status bits are also called conditional code bits or flag bit.

1. Bit C(carry) =1 if C_8 = 1 else C=0.
2. Bit S(sign) = 1 if F_7 =1 else S =0
3. Bit Z(zero) =1 if output of ALU contains all 0's i.e output is 0 else Z=0.
4. Bit V(overflow) =1 if $C_8 XOR\ C_7$ =1 else V = 0, or
   for 8 bit ALU if -127 <output <+127 V =1 else V=0.

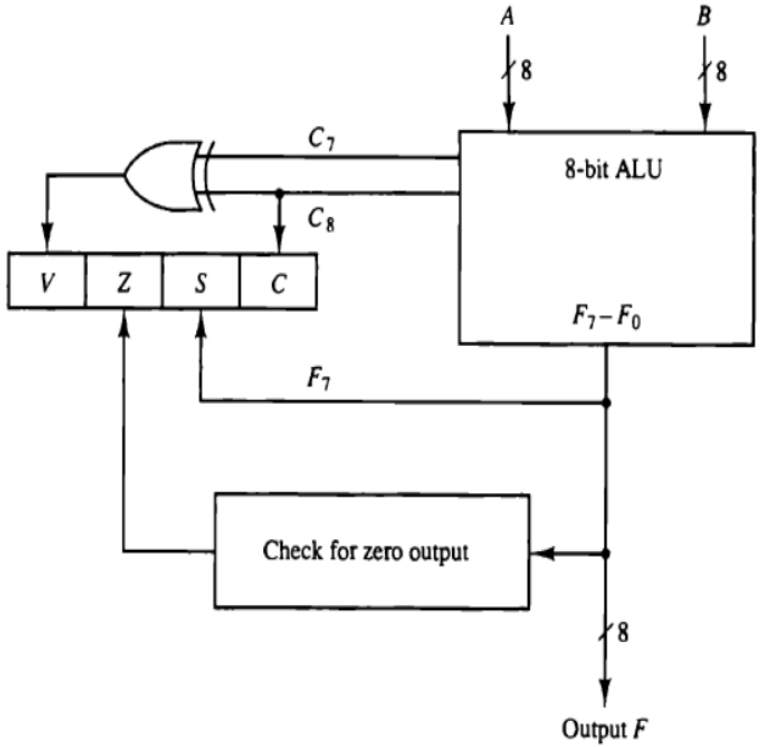


Figure 8-8 Status register bits.

Conditional Branch Instruction

TABLE 8-11 Conditional Branch Instructions

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
Unsigned compare conditions ($A - B$)		
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
Signed compare conditions ($A - B$)		
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

- Ex: A=11110000 and B=00010100 ; perform A – B

A = 11110000

B’ + 1= 11101100

A – B: 11011100

C= 1, S=1, V=0 , Z=0

Subroutine Call and Return:

A subroutine is a self-contained sequence of instruction that performs a given computational task.

During the execution of a program, subroutine may be called to perform its function many times at various points in the main program

- When subroutine called → branch is executed at the beginning of the subroutine→ After subroutine execution , branch made back to the main program.

- Common name of instruction that transfers program control to sub-routine:

1. Call subroutine ,
2. Jump to subroutine,
3. Branch to subroutine,
4. Branch and save address.

A call subroutine instruction consist of operand code + address that specifies the beginning of the subroutine.

The instruction is executed by performing two operation:

1. Return address to be store in a temporary location
2. Control is transferred to the beginning of the subroutine.

The last instruction of subroutine is commonly known as return from subroutine.

Program Interrupt

- The concept of program interrupt is used to handle a variety of problem that arise out of normal program sequence.
- PI refer to transfer of PC from currently running program to another service program that generated from internal or external generated request. Control return to original program after the service program is executed.
- Interrupt procedure same as subroutine:

1. Interrupt initiated by internal or external signal
2. Address of the interrupt service is determine by hardware rather then address field of the instruction.
3. State of the CPU be store along with the PC.

After the execution of interrupt subroutine, state of the CPU resume which is determined by:

1. The content of program counter
2. The content of all processor register
3. The content of certain status condition.

Program status word: (collection of all status bit condition of CPU) is store in a separate h/w register that characterize the state of CPU.

Type of Interrupt

- 3 major type of interrupts:

1. External interrupt
2. Internal interrupt
3. Software interrupt

External interrupt:-

Comes from I/O device ,from timing device, from ckt monitoring the power supply.

Internal interrupt:-

Illegal or erroneous use of instruction or data.

Ex: register overflow, attempt to divide by zero, invalid operation code, stack overflow.

Internal Interrupt

- Initiated by some exceptional condition caused by program itself.
- Synchronous with the program
- It occurs at the same place each time in the program

External Interrupt

- Initiated by external event
- Asynchronous
- Depend upon external condition that are independent of the program being executed at the time.

Software Interrupt

- It is initiated by executing an instruction
- It is a special call instruction that behave like an interrupt rather then a subroutine call.
- It is used by a programmer to initiate an interrupt at any desired point in the program.
- Ex: Supervisor call instruction

  Switching from CPU user mode to supervisor mode

Supervisor mode→ CPU is executing a program that is part of operating system rather then a user program.

COMPUTER ORGANIZATION | RISC AND CISC

Reduced Set Instruction Set Architecture (RISC) –
The main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating and storing operations just like a load command will load data, store command will store the data.

Complex Instruction Set Architecture (CISC) –
The main idea is to make hardware complex as a single instruction will do all loading, evaluating and storing operations just like a multiplication command will do stuff like loading data, evaluating and storing it.

- Both approaches try to increase the CPU performance
- **RISC:** Reduce the cycles per instruction at the cost of the number of instructions per program.
- **CISC:** The CISC approach attempts to minimize the number of instructions per program but at the cost of increase in number of cycles per instruction.

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

- **Characteristic of RISC –**
- Simpler instruction, hence simple instruction decoding.
- Instruction come under size of one word.
- Instruction take single clock cycle to get executed.
- More number of general purpose register.
- Simple Addressing Modes.
- Less Data types.
- Highly pipelined.

- **Characteristic of CISC –**
- Complex instruction, hence complex instruction decoding.
- Instruction are larger than one word size.
- Instruction may take more than single clock cycle to get executed.
- Less number of general purpose register as operation get performed in memory itself.
- Complex Addressing Modes.
- More Data types.
- Normally not pipelined or less pipelined.