1. What is stack? Give the organization of register stack with all necessary elements and explain the working of push and pop 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 stack in digital computers is essentially a memory unit with an address register that can count only. 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 physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted or deleted.

Register stack:

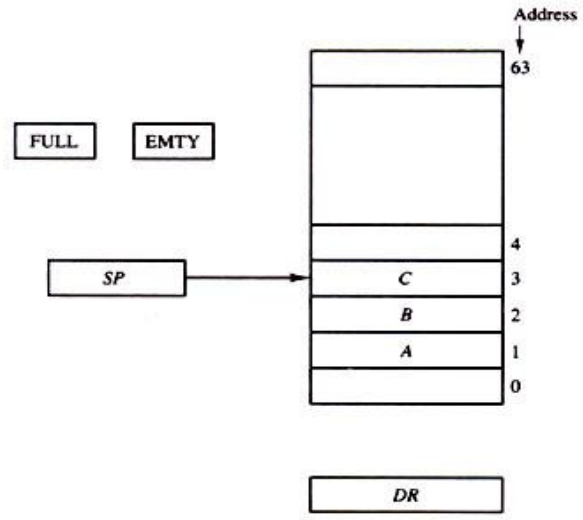


Figure 5.1: Block diagram of a 64-word stack

- 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 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.
- In a 64-word stack, the stack pointer contains 6 bits because $2^6 = 64$.

- Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 are incremented by 1, the result is 0 since 111111 + 1 = 1000000 in binary, but SP can accommodate only the six least significant bits.
- Similarly, when 000000 is decremented by 1, the result is 111111. 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.

PUSH:

- If the stack is not full (FULL =0), a new item is inserted with a push operation. The push operation consists of the following sequences of microoperations:

SP ← SP + 1	Increment stack pointer
M [SP] ← DR	WRITE ITEM ON TOP OF THE STACK
IF (SP = 0) then (FULL ← 1)	Check is stack is full
EMTY ← 0	Mark the stack not empty

- The stack pointer is incremented so that it points to the address of next-higher word. A memory write operation inserts the word from DR into the top of the stack.
- SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available in SP.
- The first item stored in the stack is at address 1. The last item is stored at address 0. If SP reaches 0, the stack is full of items, so FULL is set to 1. This condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location 0.
- Once an item is stored in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0.

POP:

- A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequences of microoperations:

DR ← M [SP]	Read item on top of the 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

- 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 1.
- Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP. If a pop operation reads the item from location 0 and then SP is decremented, SP is changes to 111111, which is equivalent to decimal 63.
- In this configuration, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL = 1 or popped when EMTY = 1.

2. Explain Memory Stack.

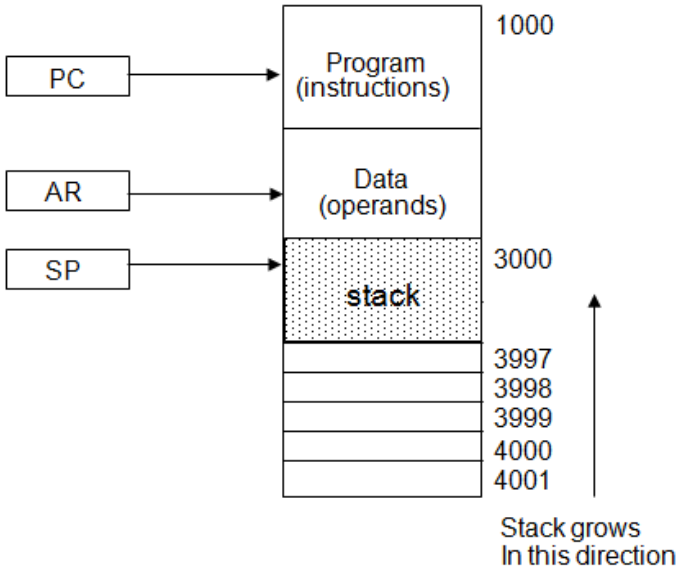


Figure 5.2: Computer memory with program, data, and stack segments

- 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.
- Figure 5.2 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 which is used during the fetch phase to read an instruction.
- The address registers AR points at an array of data which is used during the execute phase to read an operand.
- The stack pointer SP points at the top of the stack which is used to push or pop items into or from the stack.
- The three registers are connected to a common address bus, and either one can provide an address for memory.
- As shown in Figure 5.2, 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.
- We assume that the items in the stack communicate with a data register DR.

PUSH

- A new item is inserted with the push operation as follows:

  SP ← SP - 1

  M[SP] ← DR
- 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.

POP

- A new item is deleted with a pop operation as follows:

  DR ← M[SP]

  SP ← SP + 1
- The top item is read from the stack into DR.
- The stack pointer is then incremented to point at the next item in the stack.
- The two microoperations needed for either the push or pop are (1) an access to memory through SP, and (2) updating SP.
- Which of the two microoperations is done first and whether SP is updated by incrementing or decrementing depends on the organization of the stack.
- In figure. 5.2 the stack grows by decreasing the memory address. The stack may be constructed to grow by increasing the memory also.
- The advantage of a memory stack is that the CPU can refer to it without having to specify an address, since the address is always available and automatically updated in the stack pointer.

3. Explain four types of instruction formats.

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.

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

- The advantage of 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.
- An example of a commercial computer that uses three-address instruction is the Cyber 170.

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 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

- The MOV instruction moves or transfers the operands to 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 these is a need for a second register.
- However, here we will neglect the second register and assume that the AC contains the result of all operations. The program to evaluate X = (A + B) * (C + D) is

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

- All the operations are done between the AC register and a memory operand. T is the address of the temporary memory location required for storing the intermediate result.

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 program to evaluate X = (A + B) * (C + D) will be written for a stack-organized computer.

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

- To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse polish notation.

RISC Instructions:

- All other instructions are executed within the registers of the CPU without referring to memory. A program for a RISC type CPU consists of LOAD and STORE instructions that have one memory and one register address, and computational-type instructions that have three addresses with all three specifying processor registers.
- The following is a program to evaluate X = (A + B) * (C + D).

LOAD	R1, A	R1 ← M [A]
LOAD	R1, B	R1 ← M [B]
LOAD	R1, C	R1 ← M [C]
LOAD	R1, D	R1 ← M [D]
ADD	R1, R1, R2	R1 ← R1 + R2
ADD	R3, R3, R2	R3 ← R3 + R4
MUL	R1, R1, R3	R1 ← R1 * R3
STORE	X, R1	M [X] ← R1

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

4. Write a note on different Addressing Modes.

The general addressing modes supported by the computer processor are as follows:

1) <u>Implied mode:</u>

- In this mode the operands are specified implicitly in the definition of the definition of the instruction. For example, the instruction “complement accumulator” is an implied-mode instruction because the operand in the 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 later register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied mode instructions.
- Instructions using this mode have no operands.
  E.g. CLC; which clears carry flag to zero.
  STC; set the carry flag

2) <u>Immediate Mode:</u>

- 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.
- The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
- Immediate mode of instructions is useful for initializing register to constant value.
  E.g. MOV R1, 05H; instruction copies immediate number 05H to AL register

- Immediate data may be 8-bits or 16-bits in length.

Advantage:

- Operand can be accessed quickly as they are directly available in instruction queue.
- No need of External bus or bus-cycles to obtain data.
- No memory reference to fetch data
- Comparatively Faster execution

Limitation:

- The operand can only be used as a source operand.
- Value of the operand will remain Constant

3) Register Mode:

- In this mode the operands are in registers that within the CPU. The particular register is selected from a register field in the instruction.
- A k-bit field can specify any one of 2^k registers.
- Register may be used as source operand or destination operand or both
  E.g. ADD r1, r2; adding r1 and r2 and store the result in r2
  E.g. MOV AX,BX ; move value from BX to AX register

Advantage:

- This mode is normally preferred, as the execution of instruction is faster and compact, because all the registers reside on the same chip.
- Therefore, data transfer is within the chip and no External bus is required.

Limitation:

- Number of CPU registers are limited

4) Register Indirect Mode:

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- 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.
- The advantage of this mode is that address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.
  E.g. MOV [R1], R2; value of BX is moved to the memory location specified in DI

5) Autoincrement or Autodecrement Mode:

- This is similar to the register indirect mode expect that the register is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction.

6) Direct Address 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.

7) Indirect Address Mode:

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

  Effective address = address part of instruction + content of CPU register

8) Relative Address Mode:

- In this mode the content of the program counter is added to the address of the instruction in order to obtain the effective address. The address part of the instruction is usually a signed number which can be either positive or negative.
- When this 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.
- Relative addressing is often used with branch-type instruction when the branch address is in the area surrounding the instruction word itself.
- Relative address means 'relative to IP (Instruction Pointer)'.
  Example : JNC START ; jump to label if no carry is generated
- If CY=O, then PC is loaded with current PC contents plus 8 bit signed value of START, otherwise the next instruction is executed.
- Displacement is calculated on the basis of next location to be executed.

9) Indexed Addressing Mode:

- In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
- The indexed register is a special CPU register that contain an index value. The address field of the instruction defines the beginning address of a data array in memory. Each operand in the array is stored in memory relative to the begging address.
- The distance between the beginning address and the address of the operand is the index value stored in the index register.

10) Base Register Addressing Mode:

- In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
- A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.
- The base register addressing mode is used in computers to facilitate the relocation of

programs in memory.

- With a base register, the displacement values of instruction do not have to change. Only the value of the base register requires updating to reflect the beginning of a new memory segment.

5. Explain Data Transfer Instructions.

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.
- The ***load*** instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator.
- The ***store*** instruction designates a transfer from a processor register into memory.
- The ***move*** instruction has been used in computers with multiple CPU registers to designate a transfer from one register to another. It has also been used for data transfers between CPU registers and memory or between two memory words.
- The ***exchange*** instruction swaps information between two registers or a register and a memory word.
- The ***input and output*** instructions transfer data among processor registers and input or output terminals.
- The ***push and pop*** instructions transfer data between processor registers and a memory stack.

6. Explain 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

7. Explain Logical 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

8. Explain shift instructions.

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

9. What are status register bits? Draw and explain the block diagram showing all status registers.

- It is sometimes convenient to supplement the ALU circuit in the CPU with a status register where status bit conditions be stored for further analysis. Status bits are also called condition-code bits or flag bits.
- Figure 5.3 shows the block diagram of an 8-bit ALU with a 4-bit status register. The four status bits are symbolized by C, S, Z, and V. The bits are set or cleared as a result of an operation performed in the ALU.

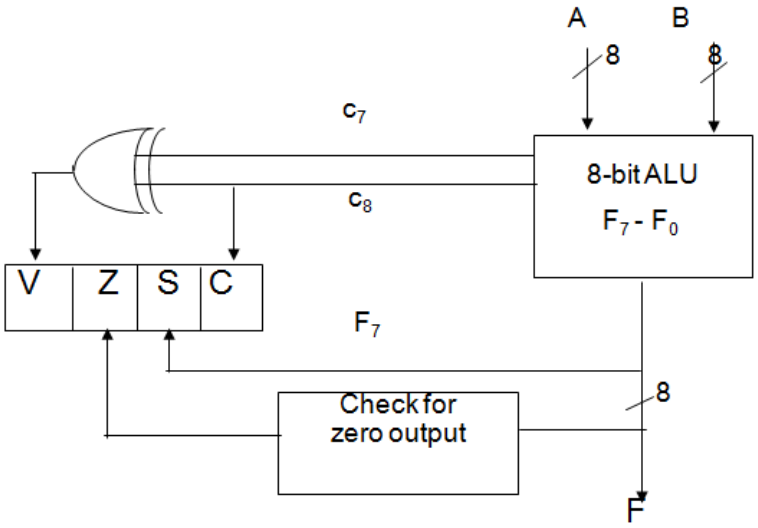


Figure 5.3: Status Register Bits

1. Bit C (carry) is set to 1 if the end carry C_8 is 1. It is cleared to 0 if the carry is 0.
2. Bit S (sign) is set to 1 if the highest-order bit F_7 is 1. It is set to 0 if set to 0 if the bit is 0.
3. Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. it is cleared to 0 otherwise. In other words, Z = 1 if the output is zero and Z = 0 if the output is not zero.
4. Bit V (overflow) is set to 1 if the exclusives-OR of the last two carries is equal to 1, and cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement. For the 8-bit ALU, V = 1 if the output is greater than + 127 or less than -128.

- The status bits can be checked after an ALU operation to determine certain relationships that exist between the vales of A and B.
- If bit V is set after the addition of two signed numbers, it indicates an overflow condition.
- If Z is set after an exclusive-OR operation, it indicates that A = B.
- A single bit in A can be checked to determine if it is 0 or 1 by masking all bits except the bit in question and then checking the Z status bit.

10. What is program interrupt? What happens when it comes? What are the tasks to be performed by service routine?

OR

Explain Program Interrupts. Explain clearly, discussing the role of stack, PSW and return from interrupt instruction, how interrupts are implemented on computers.

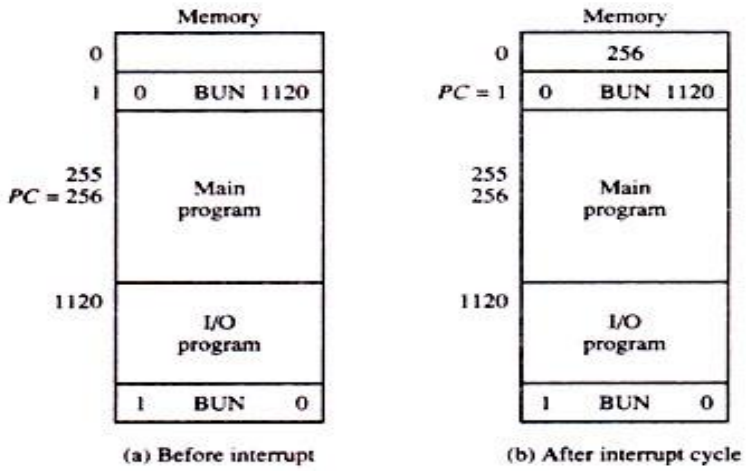
- The concept of program interrupt is used to handle a variety of problems that arise out of normal program sequence.

- Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program is executed.
- After a program has been interrupted and the service routine been executed, the CPU must return to exactly the same state that it was when the interrupt occurred.
- Only if this happens will the interrupted program be able to resume exactly as if nothing had happened.
- The state of the CPU at the end of the execute cycle (when the interrupt is recognized) is determined from:

    1. The content of the program counter
    2. The content of all processor registers
    3. The content of certain status conditions

- The interrupt facility allows the running program to proceed until the input or output device sets its ready flag. Whenever a flag is set to 1, the computer completes the execution of the instruction in progress and then acknowledges the interrupt.
- The result of this action is that the retune address is stared in location 0. The instruction in location 1 is then performed; this initiates a service routine for the input or output transfer. The service routine can be stored in location 1.
- The service routine must have instructions to perform the following tasks:

    1. Save contents of processor registers.
    2. Check which flag is set.
    3. Service the device whose flag is set.
    4. Restore contents of processor registers.
    5. Turn the interrupt facility on.
    6. Return to the running program.



11. Explain various types of interrupts.

There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

1. External interrupts
2. Internal interrupts
3. Software interrupts

1) External interrupts:

- External interrupts come from input-output (I/0) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source.
- Examples that cause external interrupts are I/0 device requesting transfer of data, I/o device finished transfer of data, elapsed time of an event, or power failure. Timeout interrupt may result from a program that is in an endless loop and thus exceeded its time allocation.
- Power failure interrupt may have as its service routine a program that transfers the complete state of the CPU into a nondestructive memory in the few milliseconds before power ceases.
- External interrupts are asynchronous. External interrupts depend on external conditions that are independent of the program being executed at the time.

2) Internal interrupts:

- Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps.
- Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation. These error conditions usually occur as a result of a premature termination of the instruction execution. The service program that processes the internal interrupt determines the corrective measure to be taken.
- Internal interrupts are synchronous with the program. . If the program is rerun, the internal interrupts will occur in the same place each time.

3) Software interrupts:

- A software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.
- The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode. Certain operations in the computer may be assigned to the supervisor mode only, as for example, a complex input or output transfer procedure. A program written by a user must run in the user mode.
- When an input or output transfer is required, the supervisor mode is requested by means of a supervisor call instruction. This instruction causes a software interrupt that stores the old CPU state and brings in a new PSW that belongs to the supervisor mode.

- The calling program must pass information to the operating system in order to specify the particular task requested.

12. What do you understand by Reduced Instruction Set Computers? What are Complex Instruction Set Computers? List important characteristics of CISC and RISC computers. Also in a tabular form compare their relative advantages / disadvantages.

Characteristics of RISC:

1. Relatively few instructions
2. Relatively few addressing modes
3. Memory access limited to load and store instructions
4. All operations done within the registers of the CPU
5. Fixed-length, easily decoded instruction format
6. Single-cycle instruction execution
7. Hardwired rather than microprogrammed control
8. A relatively large number of registers in the processor unit
9. Use of overlapped register windows to speed-up procedure call and return
10. Efficient instruction pipeline
11. Compiler support for efficient translation of high-level language programs into machine language programs

Characteristics of CISC:

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

13. What is overlapped register window? How the window size and register file size is calculated?

- 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. Each procedure call results in the allocation of a new window consisting of a set of registers from the register file for use by the new procedure.

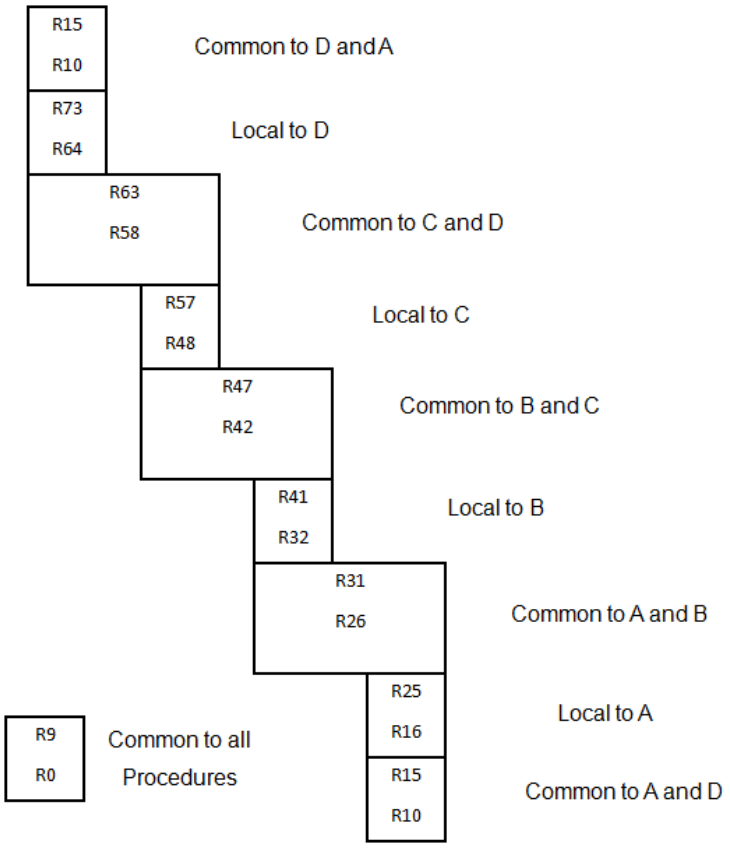


Figure 5.4: Overlapped Register Windows

- Each procedure call activates a new register window by incrementing a pointer, while the return statement decrements the pointer and causes the activation of the previous window. Windows for adjacent procedures have overlapping registers that are shared to provide the passing of parameters and results.
- The concept of overlade register windows is shown in figure. The system had a total of 74 registers. Register R0 through R9 are global registers that hold parameters shared by all procedures
- The other 64 registers are divided into four windows to accommodate procedure A, B, C and D. Each register window consists of 10 local registers and two sets of six registers common to adjacent windows.
- Only one register window is activated at any given time with a pointer indicating the active window.
- The high register of the calling procedure overlap the low registers of the called procedure, and therefore the parameters automatically transfer from calling to called procedure.
- 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 register common to two windows = C
  Number of windows = W

- The number of registers available for each window is calculated as follows:

  Window size = L + 2C + G
- The total number of register needed in the processor is

  Register file = (L + C) W + G

14. Explain Reverse Polish Notation (RPN) with appropriate example.

- The postfix RPN notation, referred to as Reverse Polish Notation (RPN), places the operator after the operands.
- The following examples demonstrate the three representations:

  A + B Infix notation
  + A B Prefix or Polish notation
  A B + Postfix or reverse Polish notation
- The reverse Polish notation is in a form suitable for stack manipulation.
  The expression

  A * B + C * D is written in reverse Polish notation as
  A B * C D * +
- The conversion from infix notation to reverse Polish notation must take into consideration the operational hierarchy adopted for infix notation.
- This hierarchy dictates that we first perform all arithmetic inside inner parentheses, then inside outer parentheses, and do multiplication and division operations before addition and subtraction operations.

Evaluation of Arithmetic Expressions

- Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polish notation

  (3 * 4) + (5 * 6) ⇒ 3 4 * 5 6 * +