Ch 6 : Program Control Instructions

Introduction

- PCIs direct the flow of a program and allow the flow to change
- the change occurs after a decision made with CMP or TEST is followed by a condition jump
- A label is a symbolic name for a memory address. must be followed by a colon.
- The .STARTUP directive only loads DS with the address of the data segment.
- The .LISTALL directive causes all assembler-generated statements to be listed, including the label @Startup generated by the .STARTUP directive.
- The .EXIT directive is also expanded by .LISTALL to show the use of the DOS INT 21H function 4CH, which returns control to DOS.

6.1 The JUMP Group

- JMP allows programmer to skip sections of a program and brunch to any part of the memory for the next instruction.
- *Conditional jumps* allows programmer to make decisions based on numeric tests, results for which are stored in flag bits then tested by conditional jump instructions
- Labels tell where to jump.
- Assemblers choose the best form of jump themselves.
- *Unconditional Jump (JMP)*
  - Short and near are called intrasegment jumps
  - Short jump: 2 byte instruction that allows jumps to memory locations within +127 to -128 bytes
    - Called relative jumpsbecause they can be moved, along with their related software, to any location in the current code segment without a change
    - displacement is sign extended
    - E.g. JMP SHORT NEXT
    - JMP $+2 jumps over the next 2 memory locations.
  - Near jump: 3 bytes, allows jump within +- 32K bytes i.e. within the current segment. +-2G if machine is in protected mode because code segment is 4G
    - Relocatable like short jump
    - The letter R denotes a relocatable jump address.
  - Far jump: 5 bytes, allows jump to any memory location within the real memory system. often called intersegment jumps
    - obtains a new segment and offset address to accomplish the jump.
    - Bytes 2 and 3 of this 5-byte instruction contain the new OA.
    - Bytes 4 and 5 contain the SA
    - in the protected mode, SA accesses a descriptor.
    - e.g. JMP FAR NEXT often written as JMP FAR PTR NEXT
    - a label can be defined as far label to obtain this jump.
      - EXTRN UP:FAR
    - External labels appear in programs that contain more than one program file.
    - Another way of defining a label as global is to use a double colon(LABEL::)
      - This is required inside procedure blocks that are defined as near if the label is accessed from outside the procedure block.
  - Jumps with Register Operands:
    - Jumps can use a register as an operand
      - register stores the address of the jump
    - considered indirect jump
    - does not add to IP but copies contents of the register to IP
    - This allows jumping to any location within current code segment
    - in protected mode the code segment can be 4G bytes long, so a 32-bit offset address is needed.
  - Indirect jumps using an Index:
    - May use [ ] form of addressing to directly access the jump table.
    - The jump table can contain offset addresses for near indirect jumps, or segment and offset addresses for far indirect jumps.
    - This type of jump is also known as a double-indirect jump if the register jump is called an indirect jump.
    - The assembler assumes that the jump is near unless the FAR PTR directive indicates a far jump instruction.
    - The mechanism used to access the jump table is identical with a normal memory reference.
    - E.g. JMP TABLE, [SI] jumps to address stored at CS with OA stored in SI
    - 16 bit offset i.e. near jumps
- *Conditional Jumps and Conditional Sets*
  - Conditional jumps:

        - short jumps or near (80386) or 2G (pentium 4)
        - Test the S, Z, C, P and O bit.
        - jump if condition is true else not
        - E.g. JC will jump if carry is set
        - Relative magnitude comparisons are complicated
        - FFH is 255H for unsigned numbers but is -1 for signed numbers
        - For signed numbers (greater than/ less than): JG, JL, JGE, JLE, JE, JNE are used.
        - For unsigned numbers(above or below) : JA, JB, JAE, JBE, JE, JNE are used.
        - JE is same as JZ, JA is same as JNBE (many alternates for all)
        - JCXZ jumps if CX is 0, and JECXZ jumps if ECX is 0
    - The Conditional set instructions:
        - They put to use the conditional jumps
        - set or clear a byte
        - useful where a condition must be tested at a point much later in the program.
- *LOOP*
    - A combination of decrement CX/ECX and JNZ
    - E.g. MOV CX, 10
      LOOP *label*
        - This loops till CX becomes zero
    - Conditional loops:
        - LOOPE (if equal) and LOOPNE (if not equal)
        - LOOPE loops as long as the condition is equal and CX is not 0

6.2 Controlling the flow of the program

- .IF, .ELSE, .ELSEIF, and .ENDIF are used to control the flow of the program
- .REPEAT–, .UNTIL and .WHILE–.ENDW statements are also used
- these are called dot commands
- *WHILE loops:*
    - .WHILE begins, .ENDW ends
    - .BREAK and .CONTINUE are also available
- *REPEAT-UNTIL loops:*
    - instructions repeated till a condition occurs
    - .REPEAT starts begin, .UNTIL marks end with a condition
    - .UNTILCXZ uses CX register as a counter to repeat a loop a fixed number of times.

6.3 Procedures

- subroutine, method or function
- It is a group of instructions that usually perform one task
- reusable section of code, stored in memory once, uses as often as necessary
- Saves memory space
- Takes time for computer to link to program and return from it
- CALL links to procedure, RET returns
- the stack stores the return instruction pushed by CALL, popped by RET
- Begins with PROC directive followed by name, ends with ENDP
- Can be followed by type : NEAR or FAR (for jumping)
    - can be followed by USES.
    - The USES statement allows any number of registers to be automatically pushed to the stack and popped from the stack within the procedure
- A near return removes a 16-bit number from the stack and places it into the IP to return from the procedure in the current CS.
- A far return removes a 32-bit number from the stack and places it into both IP and CS to return from the procedure to any memory location.
- global procedures should be far
- local procedures should be near
- *CALL*
    - pushes IP contents onto stack
    - Near CALL:
        - similar to Near JMP
        - OA of next instruction is pushed into stack

- Far CALL:
    - like far JMP
    - Byte 1 is opcode, Bytes 2 and 3 contain the new contents of the IP, and bytes 4 and 5 contain the new contents for CS.
    - places the contents of both IP and CS on the stack
  - CALLs with Register Operands:
    - like jumps with register operands
    - CALL BX pushes IP content onto stack and jumps to OA stored at BX
  - CALLs with indirect memory addresses:
    - Used when different subroutines need to be chosen in a program
- *RET*
  - pops return address from stack and places into IP
  - Near RET/ Far RET defined at PROC
  - When IP/EIP or IP/EIP and CS are changed, the address of the next instruction is at a new memory location.
    - This new location is the address of the instruction that immediately follows the most recent CALL to a procedure
  - Another form of the return instruction adds a number to the contents of the stack pointer (SP) after the return address is removed from the stack.
    - uses an immediate operand
    - ideal for use in a system that uses the C/C++ or PASCAL calling conventions.
  - These conventions push parameters on the stack before calling a procedure
  - If the parameters are to be discarded upon return, the return instruction contains a number that represents the number of bytes pushed to the stack as parameters.

6.4 Introduction to Interrupts

- Either a hardware-generated CALL
  - externally derived from a hardware signal
- or a software-generated CALL
  - internally derived from the execution of an instruction or by some other interval event
  - sometimes called an exception
- done by calling Interrupt Service Procedure (ISP) or interrupt handler
- *Interrupt Vectors*
  - 4 byte number stored in the first 1024 bytes of the memory (00000H - 003FFH) in real mode
  - the vector table is replaced by interrupt descriptor table (8 byte descriptor) in protected mode.
  - 256 vectors, each has the address of an ISP
  - For Each vector ,The first 2 bytes contain the IP, and the last 2 bytes contain the CS that forms the address of the interrupt service procedure
  - *Table 6-4 on pdf page 233 for vector type numbers.*
- *Interrupt Instructions*
  - Fetch a vector from the vector table then calls the procedure stored at the location addressed by the vector, in real mode
  - In the protected mode, each of these instructions fetches an interrupt descriptor from the interrupt descriptor table.
    - The descriptor specifies the address of the interrupt service procedure.
  - INTs:
    - 256 software interrupt instructions
    - numeric operands ranging from 00H-FFH
    - 2 bytes long: opcode- one byte, vector type number- one byte
    - The address of the interrupt vector is determined by multiplying the interrupt type number by
      - 4 in real mode. E.g. INT 10H is calling the ISP at 40H
      - 8 in protected mode.
    - INT 3 is an exception.
    - On execution:
      - It pushes the flags onto the stack,
      - clears the T and I flag bits,
      - pushes CS onto the stack,
      - fetches the new value for CS from the interrupt vector,
      - pushes IP/EIP onto the stack,
      - fetches the new value for IP/EIP from the vector,
      - jumps to the new location addressed by CS and IP/EIP.
    - can be considered as PUSHF followed by a far CALL
    - Used to call system procedures since address of the function may not be known

        - IRET/IRETD:
            - on execution:
                - pop stack data back into IP
                - pop stack data back into CS
                - pop stack data back into flag register
            - can be considered POPF followed by far RET
            - removes six bytes from the stack: two for the IP, two for the CS, and two for the flags.
            - it restores contents of I and T to preserve the state of these flags
            - IRETD is for protected mode
        - INT3:
            - It is a 1-byte special software interrupt used for breakpoints.
                - because it is easy to insert a one-byte instruction into a program
            - very commonly used
            - helps in debugging
        - INTO:
            - Interrupt on Overflow
            - conditional software interrupt that tests the overflow flag (O)
            - a vector type number 4 interrupt occurs when O=1
            - Appears in software that add or subtract signed binary numbers with possible overflows
        - An Interrupt Service Provider:
            - common tasks can be developed as software interrupts
- *Interrupt Control*
    - Set interrupt flag (STI) enables interrupt
    - Clear interrupt flag (CLI) disables interrupt
    - hardware interrupts are enabled early because all I/O devices are interrupt processed
- *Interrupts in the Personal Computers*
    - Only contained Intel-specified interrupts (0-4)
    - Protected mode is accessed using kernels
- *64-bit mode interrupts*
    - Uses IRETQ .
        - it receives 8 byte return address
        - also retrieves the 32-bit EFLAG register from the stack and places it into the RFLAG register.

6.5 Machine Control and Miscellaneous Instructions

- *Controlling the Carry flag bit:*
    - C flag propagates the carry or borrow
    - set by STC, cleared by CLC, CMC complements carry
    - also used to indicate error upon return from procedure, C=1 if error occurs
- *WAIT*
    - monitors the BUSY pin on 80286 and 80386 and TEST pin on 8086/8088
    - microprocessor waits if BUSY = 0 till it becomes 1
- *HLT*
    - stops execution of software
    - three ways to halt:
        - call an interrupt
        - reset hardware
        - during a DMA operation
    - occurs to wait for an interrupt
    - syncs external hardware interrupts with the software system
    - not for DOS and Windows
- *NOP*
    - No operation instruction
    - Makes the MP take a short time to execute
    - Added in case you need to add instructions at some future point.
    - can help in time delaying/ wasting
- *LOCK Prefix*
    - Appends an instruction and causes LOCK pin to become 0
    - disables external bus masters or other system components

    - this prefix causes the LOCK pin to activate for only the duration of a locked instruction.
    - E.g. LOCK:MOV AL,[SI]
- *ESC*
    - escape passes instructions to the floating point coprocessor from MP
    - on execution, MP provides the memory address only if required else performs an NOP
    - 6 bits
    - obsolete in use
- *BOUND*
    - a comparison instruction that may cause an interrupt (vector type number 5).
    - compares contents of a register(16/32 bit) against the contents of 2 words or doublewords of memory : upper and lower boundary
    - if not in the boundary, interrupt occurs, else next instruction
    - E.g. BOUND SI,DATA
- *ENTER and LEAVE*
    - used with stack frames
        - these are mechanisms used to pass parameters to a procedure through the stack memory
        - holds local memory variables for the procedure
    - ENTER:
        - creates a stack frame by pushing BP onto the stack then loading BP with upper most address of the stack frame
        - this allows stack frame variables to be accessed through the BP register.
        - contains two operands:
            - The first operand specifies the number of bytes to reserve for variables on the stack frame,
            - the second specifies the level of the procedure.
        - E.g. ENTER 8,0 instruction reserves 8 bytes of memory for the stack frame and the zero specifies level 0.
            - It subtracts 8 from the stack pointer, leaving 8 bytes of memory space for temporary data storage
    - LEAVE:
        - reverses the above process by reloading SP and BP with their previous values