Ch 4 : Data Movement Instructions (upto 4.5)

Introduction

- Data movement instructions include MOV, MOVSX, MOVZX, PUSH, POP, BSWAP, XCHG, XLAT, IN, OUT, LEA, LDS,LES, LFS, LGS, LSS, LAHF, SAHF.
- string instructions MOVS, LODS, STOS, INS, and OUTS.
- The latest data transfer instruction implemented on the Pentium Pro and above is the CMOV (conditional move) instruction
- These are commonly used and easy to understand

4-1 MOV Revisited

- the MOV instruction introduces the machine language instructions available with various addressing modes and instructions.
- Machine code is introduced because it may occasionally be necessary to interpret machine language programs generated by an assembler or inline assembler of Visual C++.
- Interpretation of the machine's native language allows debugging or modification at the machine language level.
- ***Machine Language***
  - the native binary code that the microprocessor understands and uses as instructions to control its operation.
  - 1-13 bytes, 16 bit mode for 8086-80286
    - 80386 and above, 16 bit for real mode
    - In the protected mode(Windows), the upper byte of the descriptor contains the D-bit that selects either the 16- or 32-bit instruction mode.
  - Due to many variations, only some bits are given and remaining determined
  - The first 2 bytes of the 32-bit instruction mode format are called override prefixes because they are not always present.
    - The first modifies the size of the operand address used by the instruction
    - the second modifies the register size.
  - if 32 bit register is used for 16 bit instruction, register size prefix(66H) is appended to the front. This selects a 16 bit register
  - Same usage for address size prefix(67H)
    - The prefixes toggle the size of the register and operand address from l6-bit to 32-bit or from 32-bit to l6-bit for the prefixed instruction.
    - Default is 8 and 16 bit or 8 and 32 bit mode which the prefixes override
  - Fig 4-1 page 132
  - *Opcode*
    - Selects the operation
    - 1-2 bytes long
      - first 6 are binary opcode
      - next is direction (D)...not the flag bit
        - if D=1, flow of data is to register from R/M field
        - if D=0, from REG to R/M
        - in mostly MOV
      - next is W that indicates whether word, or byte
        - W=1 or word or double word
        - determined by instruction mode and register size prefix
        - in all instructions
  - *MOD field*
    - specifies addressing mode
      - 11- R/M is register
      - 00- no displacement
      - 01 - 8 bit sign extended displacement
        - extended to 16 bit
        - positive : 00H-7FH extended like 007FH
        - negative: 80H - FFH extended like FFFFH
      - 10 - 16 bit signed displacement
        - becomes 32 bit to allow for protected mode in 80386
        - 8 bit will be extended to 32
  - *Register Assignments*
    - Table 4-3 page 135 for REG codes
  - *R/M memory addressing*
    - Table 4-4 on page136 for 16 bit R/M codes

        - for displacement the instruction becomes 3 byte instead of 2
            - MOV DL, [DI] is 8A15
            - MOV DL, [DI+1] is 8A5501
            - MOV DL, [DI+1000H] is 8A750010H
    - *Special Addressing Mode*
        - occurs whenever memory data are referenced by only the displacement mode of addressing for 16-bit instructions.
        - E.g. MOV [1000H],DL and MOV NUMB,DL
        - BP can only be used with a displacement
    - *32-bit Addressing Modes*
        - Either in 32 bit instruction mode or in 16 bit by using address size prefix
        - Table 4-5 page 138 for 32-bit R/M
        - Scaled index byte appears in the instruction when R/M=100
            - indicates additional forms of scaled index addressing
            - Used when 2 registers are added to specify memory address
            - 7 bits opcode, 8 bits scaled index byte
            - 2^{15} =32K possible combinations
            - 00 =× 1, 01 =× 2, 10 =× 4, 11 =× 8
            - MOV EAX, [EBX+ 4*ECX] is coded as 67668B048BH in 16 bit mode
                - 8B048BH in 32 bit mode
    - *An immediate Instruction*
        - E.g. MOV WORD PTR [BX+1000H], 1234H
        - WORD PTR, BYTE PTR etc are necessary only when it is not clear whether the operation is a byte, word, or doubleword.
            - like MOV [BX],9 should be written as MOV DWORD PTR [BX],9
            - MOV [BX],AL is clearly a byte move
    - *Segment MOV instructions*
        - If the contents of a segment register are moved by the MOV,PUSH, or POP instructions, a special set of register bits (REG field) selects the segment register ...Table 4–6 page 140
        - To load a segment register with immediate data, first load another register with the data and then move it to a segment register. Direct is not available
- ***The 64 bit mode for the Pentium4 and Core2***
    - An additional prefix called REX(register extension) is added
        - 40H to 4FH
        - modifies reg and r/m field for 64 bit operation
        - to address registers R8 through R15
    - Table 4-7 page 141 for r/m assignments
    - The reg field can only contain register assignments
    - r/m field contains either a register or memory assignment.
    - the modes allowed by the scaled-index byte are fairly all conclusive allowing pairs of registers to address memory

4-2 PUSH/POP

- important instructions that store and retrieve data from the LIFO (last-in, first-out) stack memory.
- six forms of the PUSH and POP instructions: register, memory, immediate, segment register, flags, and all registers.
- immediate and all registers are not available in 8086/8088
- Data can't be popped into ES
- Immediate data can be popped
- *PUSH:*
    - 2/4 bytes of data transferred to stack
    - PUSHA transfers contents of all internal registers except SRs to the stack
        - In the order: AX,CX, DX, BX, SP, BP, SI, and DI
        - For SP, the value before PUSHA executed is pushed
        - not for 64 bit mode
        - PUSHAD for 32 bit registers in 80386 through Pentium4
    - PUSHF for flags
    - on pushing, first data byte moves to SP-1 SS memory location
        - second data byte moves to SP-2 SS memory location
    - after push, SP decrements by 2
        - 4 bytes for a doubleword
    - PUSHD for immediate datum

        - 6AH opcode if data is 00H-FFH
        - 68H opcode for 0100H-FFFH
        - E.g. PUSH 8 is 6A08H
        - PUSH 1000H is 680010H
        - PUSH 'A' pushes 0041H
- *POP*
    - removes from stack and places into target register, SR or location
    - no immediate POP
    - POPF(16 bit), POPFD(32 bit) for flags
    - POPA removes all data from stack and places them into register
        - in the order: DI, SI, BP, SP, BX, DX, CX, and AX
        - POPAD for 32 bit
    - first data byte picked from SP SS memory location
    - second byte data picked from SP+1 SS memory location
    - SP increments by 2
    - POP CS not allowed
- *Initialising the Stack*
    - Both SS and SP are initialised
    - SS is loaded with bottom location of the area of memory designated as SS
    - all segments are cyclic in nature— the top location of a segment is contiguous with the bottom location of the segment.
    - The assembler and linker programs place the correct stack segment address in SS and the length of the segment (top of the stack) into SP
    - .STACK defines the stack area and initialises SS and SP
    - a warning will appear when the program is linked if stack is not defined
        - it may be ignored if stack size is less than or equal to 128 bytes(this is automatically assigned)
    - the 128 byte memory section is located in the program segment prefix(PSP), which is appended to the beginning of each program file.
        - PSP will erase as more than 128 Bytes is used
        - If the TINY memory model is used, the stack is automatically located at the very end of the segment, which allows for a larger stack area.

4-3 Load Effective Address

- LEA loads any register with the OA determined by the addressing mode
- LDS and LES loads register with OA retrieved from memory location and load either DS or ES with SA retrieved from memory
- LFS, LGS and LSS are added in 80386 and above to select a 32 bit register to receive a 32 bit OA
- LDS and LES are invalid in 64 bit mode for Pentium4 because of no use of segments in flat memory model
- *LEA*
    - Loads register with OA of data specified by the operand
    - E.g. LEA AX, NUMB. address of NUMB stored in AX
        - LEA BX, [DI] stores the OA given in DI in BX
        - MOV BX, [DI] stores the data at the EA, formed by OA given in DI, into BX
    - Same as MOV BX, OFFSET LIST
        - OFFSET is more effective with operands like LIST
        - Does not work for [DI] or LIST[SI].
    - LEA takes longer to execute
        - because the microprocessor calculates the address for this instruction
        - for MOV with OFFSET, the assembler calculates the address
            - it is assembled as a move immediate instruction thus more effective
    - E.g. LEA SI, [BX+DI] if sum of BX+DI is modulo 64K sum (say 10F00H) then its 16 bit result is stored in SI (i.e. 0F00H)
- *LDS, LES, LFS, LGS, LSS*
    - load registers with OA
    - load SRs with SA
    - Access a 32 or 48 bit section of memory
    - obtain a new far address from memory stored by the assembler
        - basically considers the data stored at the EA as the new address to be stored in SRs.
        - this is how SRs get data into them
    - LDS BX, [DI] transfers data addressed by DI into BX and DS
    - in 48 bit, first 4 bytes are OA and next SA

    - LSS is most useful
        - CLI and STI must be included to disable interrupts

4-4 String Data Transfers

- LODS, STOS, MOVS, INS, and OUTS.
- Allows data transfers of a single byte, word, doubleword or their blocks
- D flag bit, DI and SI are applied
- *The Direction Flag (D)*
    - selects auto increment or decrement for SI or DI registers
    - only for string operations
    - CLD clears D and STD sets it to 1
        - CLD - D=0 means auto increment
        - STD - D=1 means auto decrement
    - DI and/or SI increment or decrement by 1 everytime a string instruction transfers a byte
        - EDI and ESI in 80386 and above
        - by 2 for word
        - by 4 for doubleword
    - For STOSB, DI is used
    - For LODSB, SI is used
- *DI and SI*
    - Contain OA
    - DI accesses data in ES
        - segment assigned cant be changed
    - SI accesses data in DS
        - segment assigned can be changed with segment override prefix
    - The reason that one pointer addresses data in the extra segment and the other in the data segment is so that the MOVS instruction can move 64K bytes of data from one segment of memory to another.
        - 4G bytes in 80386 and above
- *LODS*
    - Uses SI
    - loads AL, AX, or EAX with data stored at the DS OA
        - AL-byte
        - AX-word
        - EAX-doubleword
    - Append B, W, D
    - *Table 4-11 for examples page151*
- *STOS*
    - Uses DI
    - stores AL, AX, or EAX at the extra segment memory location
    - Append B, W, D
    - *STOS with a REP:*
        - repeat prefix(REP) added to string data transfer instructions except LODS
        - CX is caused to decrement.
        - instruction is repeated till CX is 0
        - causes a block of memory to be stored
- *MOVS*
    - transfers data from one memory location to another
    - Uses SI as source operand and DI as destination operand
        - SI can be changes but not DI
- *INS*
    - Input String Instruction
    - transfers byte, word or doubleword
    - transfers from an I/O device into the ES
    - Uses DI
    - I/O address is contained in the DX
    - inputs a block of data from an external I/O device directly into the memory.
        - One application transfers data from a disk drive to memory.
        - Disk drives are often considered and interfaced as I/O devices in a computer system.

    - INSW for word, INSD for doubleword
    - Repeated using REP prefix
- *OUTS*
    - transfers a byte, word, or doubleword
    - from the DS memory location to an I/O device.
    - The I/O device is addressed by the DX register
    - Uses SI
- The DB pseudo-operation defines byte(s), the DW pseudo-operation defines word(s), and the DD pseudo-operations define doubleword(s).