Computer Architecture - Lecture 5

- Intel 8080/8086
  - Jump (branch) instructions are either unconditional or conditional.
    - <u>Unconditional jump:</u>

      Jmp <address>
    - The address can be near or far. Near address is a relative address (measured relative the current value of ip). Far address can be a complete address (segment:offset).
    - Technically, the jump instruction modifies the program counter(ip).
    - In assembly we write:

      Jump <lable>
    - The label is interpreted as an address.
    - Example:

      ```
      jmp L1
      mov ax, 7
      L1: mov ax, 10
      ```
    - <u>Conditional branch (jump):</u>
      - For conditional jump, a register called 'flags' is added to register files. The flags register is accessed bitwise. Examples of flags bits:
        - Carry flag (1 if the last operation produces a carry)
          - Example:
            - Mov AL,190
            - Add Al, 67 ; AL should be 257 but since max value is 255, a carry occurs and AL=2
        - The carry flag is set (equals 1).
- The sign flag is set when the sign bit is 1 (negative number).

  Example:

  Mov AX,76

  Sub AX-80 ; negative result (-4) and sign flag is set(1).
- The overflow flag: if the content of the last instruction destination overflows (becomes greater than the maximum value)
- The zero flag: if the content of the destination is zero.
- Conditional jump instructions are based on flag values,
  - JC <address/label>; jump if the carry flag is set.
  - JNC <address, label>; jump if the carry flag is clear.
  - JS <address/label> ; jump if the sign flag is set.
  - JNS <address/label> ; jump if the sign flag is clear.
- Also, higher level forms are supported by assemblers:
  - JL <address/label> ; jump if less (destination <source> in last subtraction) (signed)

- JG <address/label> ; jump if greater (signed).
    - JNL <address/label> ; jump if not less.
    - JGE <address/label> ; jump if greater or equal.
    - JA <address/label> ; jump if above (unsigned).
    - JB <address/label> ; jump if below (unsigned).
- Example:
    - Mov AL, 190
    - Add AL, 67
    - JC L1; will jump to L1 since the carry flag is set.
- Example:
    - Mov AL, 10
    - Sub AL, 15 (AL changes to -5)
    - JC L1; will jump since AL < 15.
- CMP and text instruction:
    - The CMP instruction compares the destination with the source (by subtraction destination – source without storing the result, just change the flags)
- Example:
    - Mov AL, 10
    - Cmp AL, (15 AL remains 10)
    - JL L1 ; jumps since AL < 15.
- Example:
    - Mov AL, 130; (binary 10000010) As a signed number (2's complement) = - (01111110)
    - Cmp AL, 132; (binary 10000100) As a signed number (2's complement) = - (01111100)
    - JA and JB will behave different than JG and JL.
    - JB will jump but JL will not jump.
- Example:
    - Mov AL, -3 - (00000011) → 11111101
    - Cmp AL, 5 (00000101)
    - JA will jump but JG will not jump)
- Loop instruction:
    - Loop <address/label>
    - This is a composite instruction equivalent to:
        - Dec CX
        - Cmp CX, 0
        - Jnz <address/label>

  - To implement a loop of 10 iterations:
    - Mov cx, 10
    - L1: xxx (serval lines of code)
    - Loop L1
- Stack instructions:
  - Stack is a memory area accessed in a LIFO order (last in first out).
  - The top of stack is pointed to SS (stack segment) : SP (stack pointer).
  - Basic stack operations
    - Push <src>; stores the value of the src to the top of stack.
  - Example:
    - Mov AX, 50
    - Push AX; stores 50 on the top of the stack
    - Mov AX, 90
    - Push AX; stores 90 on the top of the stack (above 50)
  - Pop <dst>; removes the topmost stack value and store it in destination.
  - Example:
    - Pop BX; removes 90 from the stack and store it in BX.
    - Pop CX; removes 50 from the stack and store it in CX.
  - Technically, push is a composite instruction.
    - Push <src> is equivalent to:
      - Sub SP, <size of the source>
      - Mov ss:[sp], src
  - Example: Let the SP = 500 then if we write:
    - Mov AX, 50
    - Push AX
  - Then, SP will be 498. Memory at the address ss:[498] contains 50.
    - Mov AX, 50
    - Push AX
  - Then, SP will be 496. Memory at the address ss:[496] contains 90.
- The pop instruction (POP <dst>) is a composite instruction equivalent to:
  - Mov dst, ss:[sp]
  - Add ss:[sp], size of <dist>
- Example:
  - Pop BX
  - BX = ss:[496] = 90
  - SP will be 498
  - Pop CX
  - CX = ss:[498] = 50
  - Sp will be 500

- In programming languages, stack is used for temporary storage [local variables and formal parameters of functions. It also supports the function calls.
- Call and return instruction:
    - Call <address/label>, calls a procedure (jump and wait to return)
    - Ret; returns to address after the last call.
- Example:
    - …
    - Call L1; Pushes L2 into the stack and jumps to L1.
    - L2: xxx (serval lines of code)
    - L1: xxx (serval lines of code)
    - Ret; pops the top of stack into instruction pointer (ip) [i.e., jump to L2]
- X86 32-bit processors:
    - Processors have the same names, but a prefix e added:
    - Registers: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, EIP, EFlags, etc
    - Each is 32-bit registers.
    - AX is the lower 16-bits of EAX, BX is the lower 16-bits of EBX, etc.
    - AL is the lower 8 bits of EAX, AH is the higher 8 bits of (AX). i.e., the second lower 8 bits of EAX.
- X86 64-bit processors:
    - Registers have the same names with a prefix r.
    - Examples: RAX, RBX, RCX, RDX, etc.
    - Also, some registers added like: R8, R9, … R15.
    - Each register is 64 bits.
    - EAX is the lower 32-bits of RAX, etc.