**Stacks**

* **Stack** – a list of data elements of a particular size with LIFO accessing
  + The stack resides in main memory
  + Top of the stack (most recent element) = lowest memory address
  + Bottom of stack (least recent element) = highest memory address
  + SP – stack pointer register points to top of the stack
    - Points to the last (topmost) occupied location
  + Push – SP is decremented and data is copied to new location pointed to by SP
  + Pop– data is copied from the location pointed to by SP and incrementing SP
* **Subroutine** – block of instructions that is executed repeatedly
  + Used to:
    - Avoid duplication of code
    - Permit code reuse
    - Enable modularization
  + Program invokes subroutine by executing the call instruction
  + Subroutine executes and return to main program by executing the return instruction
  + A call is a special branch instruction
    - Store contents of PC (return address) in link register – so that program knows where to go after subroutine returns
    - Branch to target address specified by instruction
  + Return – put address in link register back into PC
  + **Nesting subroutines** – e.g. sub1 calls sub2
    - Sub1 needs to save the address in link register on the stack, then call sub2
    - After returning from sub2, sub1 needs to retrieve link register address from the stack
    - Problems can occur if:
      * Stack is full; calling subroutine causes stack overflow
      * Stack is not properly maintained
  + **Parameter passing** – information exchange to/from a subroutine
    - Using registers – only if small # of parameters; subroutine is not recursive; and will not call other subroutines that use the same registers
    - Using memory locations – not usually used
    - Using the stack – store values in stack before call, load values from stack within subroutine
* **Stack frame** – location at the top of the stack used as a private workspace for subroutines
  + Frame pointer register (FP) – enables access to private workspace for the current subroutine
    - Any register other than the SP can be used
  + Items pushed onto the stack frame (in order):
    - Prior to calling a subroutine – parameters, return address
    - During subroutine execution – FP, local variables, register values for registers used by the subroutine
* **Additional instructions**
* Logic operations – AND, OR, NOT
* Shift & rotate operations
  + Arithmetic shift preserves the sign in the MSB
  + Arithmetic shift = logical shift left – add 0s at LSB, push everything left (multiply by 2)
  + Logical shift right – add 0s at MSB, push everything right (divide by 2)
  + Arithmetic shift right – add 0s or 1s at MSB, depending on the sign
  + Rotate – copy bits from one end to another
* Instructions encoding
  + RISC – instructions are limited to single words
  + 32-bit immediate must be assigned by:
    - Load 16-bit, shift left 16 bits, load 16-bit
    - Or save immediate in memory, then load from memory into register
  + Register-operand format
    - MSB – Rsrc1 – Rsrc2 – Rdst – opcode – LSB
  + Immediate-operand format
    - MSB – Rsrc – Rdst – immediate – opcode – LSB
  + Call format
    - MSB – immediate – opcode - LSB