The Stack and Addressing Modes
The Stack

• The stack is simply an area of memory used for data storage

• The stack is a LIFO buffer (Last In First Out)

• The stack pointer (SP) holds the address of the ‘top’ of the stack
Stack Operation

[Diagram showing stack operation with labels for low and high addresses, and data points]

The Stack and Addressing Modes
Push and Pull

(a) Push data onto stack to store it

(b) Pull data from stack to retrieve it

The Stack and Addressing Modes
What’s it For?

- The most important use of the stack is for subroutines.
- When a program jumps to a subroutine, the return address is pushed onto the stack (JSR).
- When the subroutine returns (RTS) the address is pulled from the stack.
- This is particularly useful when nested subroutines are used.
- The stack is also often used for the temporary storage of data.
Be Careful

• Stack overflow – When the stack grows so large it overwrites user code

• Items must be pulled from the stack in the reverse order that they were pushed to it (match each push with a pull)

• Every subroutine must end with a return from subroutine (RTS)

• Remember the stack actually decrements with a push. i.e. it grows downwards in memory space
Subroutines

• Begin with a label and end with RTS

• E.g.

  Start:
  
  LDAA #$00
  JSR subroutine
  BRA Start

subroutine:

  LDAA #$FF
  RTS
Nested Subroutines

MAIN calls a subroutine.
MAIN calls a subroutine, which in turn calls a nested subroutine.
Subroutines and the Stack

- Parameters may be passed via the stack (important if the subroutine needs to be reentrant)
  - Calling code pushes data onto the stack and saves location in an index register
  - The subroutine uses the data (accessed via the index register)
  - The subroutine puts the results in the stack for the calling code to pull off

- Local variables
  - The stack can be used for storage of ‘temporary’ local variables
  - More efficient use of memory, can avoid memory ‘fragmentation’
Addressing Modes

- Immediate
- Direct
- Extended
- Indexed
- Inherent
- Relative
Immediate Addressing

• Data is part of the instruction

• E.g.  

LDAA #$5A
Direct Addressing

• Data is located somewhere in (on-chip) RAM (i.e. in the $00 - $FF range)

• E.g. STAA $80
Extended Addressing

• Location of data is specified by a 16-bit address

• E.g. STAA $8800
Indexed Addressing

• The location of data is specified by the sum of a 16-bit index register (x or y) and an 8-bit offset value

• E.g.
  
  LDX #$1000
  LDAA 5,X
Inherent Addressing

- Data is ‘inherent’. Nothing is required except the opcode.

- E.g.  
  
  TAB  
  RTS
Relative Addressing

• Location is specified by an 8-bit offset from the address of the instruction currently being executed

• Only used in branch instructions

• E.g. BRA 3