Table 1

Addressing mode		Implicit	Accumulator	Immediate	Zero Page Absolute	Zero Page,X	Zero Page,Y	Relative	Absolute	Absolute,X	Absolute,Y	Indirect	Indexed Indirect	Indirect Indexed
Parameters		**-**	**A**	**#$nn**	**$nn**	**$nn,x**	**$nn,y**	**$nn (signed)**	**$nnnn**	**$nnnn,x**	**$nnnn,y**	**($nnnn)**	**($nn,x)**	**($nn),y**
		Implied by operator	**Acts on accumulator**	**literal value**	**value at $nn**	**value at ($nn+x)**	**value at ($nn+y)**	**PC = PC+$nn**	**value at $nnnn**	**value at ($nnnn+x)**	**value at ($nnnnn+y)**	**value of (value at $nnnn)**	**value of (value at ($nnnn+x))**	**value of ((value at $nnnn) + y)**
ADC	Add with Carry			x	x	x			x	x	x		x	x
AND	Logical AND			x	x	x			x	x	x		x	x
ASL	Arithmetic Shift Left		x		x	x			x	x				
BCC	Branch if Carry Clear (C=0)							x						
BCS	Branch if Carry Set (C=1)							x						
BEQ	Branch if Equal (Z = 1)							x						
BIT	Bit test				x				x					
BMI	Branch if Minus (N=1)							x						
BNE	Branch if Not Equal (Z=0)							x						
BPL	Branch if Positive (N=0)							x						
BRK	Break	x												
BVC	Branch if oVerflow is Clear (O=0)							x						
BVS	Branch if oVerflow is Set (O=1)							x						
CLC	Clear Carry	x												
CLD	Clear Decimal	x												
CLI	Clear Interrupt mask	x												
CLV	Clear Overflow flag	x												
CMP	Compare A to value			x	x	x			x	x	x		x	x
CPX	Compare X to value			x	x				x					
CPY	Compare Y to value			x	x				x					
DEC	Decrement value				x	x			x	x				
DEX	Decrement X	x												
DEY	Decrement Y	x												
EOR	Logical XOR			x	x	x			x	x	x		x	x
INC	Increment				x	x			x	x	x		x	x
INX	Increment X	x												
INY	Increment Y	x												
JMP	Jump to location								x			x		
JSR	Jump to subroutine								x					
LDA	Load into A			x	x	x			x	x	x		x	x
LDX	Load into X			x	x		x		x		x			
LDY	Load into Y			x	x	x			x	x				
LSR	Logical Shift Right		x		x	x			x	x				
NOP	No Operation	x												
ORA	Logical OR			x	x	x			x	x	x		x	x
PHA	Push A	x												
PHP	Push Processor (Flags)	x												
PLA	Pull A	x												
PLP	Pull Processor (Flags)	x												
ROL	Rotate Left		x		x	x			x	x				
ROR	Rotate Right		x		x	x			x	x				
RTI	Return from Interrupt	x												
RTS	Return from Subroutine	x												
SBC	Subtract (with carry)			x	x				x	x	x		x	x
SEC	Set Carry	x												
SED	Set Decimal	x												
SEI	Set Interrupt Mask (disable interrupts)	x												
STA	Store A				x	x			x	x	x		x	x
STX	Store X				x		x		x					
STY	Store Y				x	x			x					
TAX	Transfer A to X	x												
TAY	Transfer A to Y	x												
TSX	Transfer SP to X	x												
TXA	Transfer X to A	x												
TXS	Transfer X to SP	x												
TYA	Transfer Y to A	x												

Notes

On numerical values

nn = decimal
$nn = hexadecimal

The assembler converts for you.

Signed vs unsigned
Usually values are interpreted as unsigned. We mostly use signed values for relative jumps, and we let the assembler worry about the values.

Addressing modes

Implicit
No parameters; the instruction implies on what value to act (if any).
Examples: INX (operates on X), RTS (operates on PC)

Accumulator
Use A as parameter to act on A.
Examples: LSR A (left shifts A), LSR $00 (left shift value at $00)

Zero Page
Single byte parameter of zero page memory address.
Example: LDA $00 (load memory value at $00 into A)

Zero Page,X
Single byte parameter of zero page memory address, X will be added to it.
Example: LDX #$a0, LDA $10,X (load memory value at $b0 into A)
Note: Wraps around, so if X = $f0, LDA $10,X will not load $0100 but $00.

Zero Page,Y
Same as Zero Page,X but with Y.

Relative
Single byte parameter, interpreted as signed. Only used by branch instructions, which add the value to the PC (if the condition is matched).
You'll generally use labels, and let the assembler worry about the actual value.
Note: this means you cannot branch farther than -128 - +127 bytes. If you need this, branch to a closer position after which you'll JMP to where you need to go.

Absolute
Two byte parameter of a memory address in the full 65k range.
Examples: JMP $abcd (set PC to $abcd), LDA $1234 (load value at $1234 into A)

Absolute,X
Two byte parameter of a memory address in the full 65k range, X will be added to it.
Example: LDX #$04, LDA $1230,X (load memory value $1234 into A)
Note: does not wrap around, unlike Zero Page,X/Y (although mabye around $ffff?)

Absolute,Y
Same as Absolute,X but with Y.

Indirect
Only used by JMP.
Two byte parameter of a memory address which contains the low byte a 16 bit value; the next byte should contain the high byte.
Example: LDA #$cd, STA $f000, LDA #$ab, STA $f001, JMP ($f000) (set PC to $abcd; the contents of $f001 $f000 respectively)

Indexed Indirect
Single byte parameter of a zero page address. The value of X will be added to the address, which then should contain the low byte of a 16 bit value; the next byte should contain the high byte. This is useful for an address table.
Example:
Assume the following bytes at $00: $cd $ab $34 $12
LDX #$00, LDA ($00,X) (loads the value at address $abcd into A)
LDX #$02, LDA ($00,X) (loads the value at address $1234 into A)
Note: This wraps around like Zero Page,X.

Indirect Indexed
Single byte parameter of a zero page address containing the low byte of a 16 bit value; the next byte should contain the high byte. The value of Y will be added to the resulting address. This is useful fo point at a struct in memory.
Example:
Assume the following bytes at $00: $30 $12
`LDY #$00, LDA ($00),Y` (loads the value at address `$1230` into A)
`LDY #$04, LDA ($00),Y` (loads the value at address `$1234` into A)

Processor Status / Flags

Carry - Set if last operation caused overflow from bit 7 or underflow from bit 0.
Zero - Set if the result of the last operation was 0.
Interrupt disable - If set processor will not respond to interrupts
Decimal mode - If set ADC and SBC will use binary coded decimal arithmetic (eg 9+1=10, instead of $9 + $1 = $a).
Break - Set if BRK has been executed and an interrupt has been generated to process it.
Overflow - Set during arithmetic operation if the result has yielded an invalid 2's complement result (e.g. adding to positive numbers and ending up with a negative result: 64 + 64 => -128). It is determined by looking at the carry between bits 6 and 7 and between bit 7 and the carry flag.
Negative - Set if the result of the last operation had bit 7 set to 1.

Memory layout

Zero page

The zero page is is the first page of memory, so `$0000 - $00ff`. There are special addressing modes which use this page. These are generally 1 clock cycle faster than the instructions which access the higher pages in memory so they are a good place for variables you need often.

On our "machine"

- `$fe` contains a new random byte on every instruction
- `$ff` contains the ascii code of the last key pressed

Stack

The stack lives in the second page of memory (`$0100 - $01ff`) and cannot be moved. Some clones differ, or can move the origin.
The SP starts at `$01ff` and decrements when you push values onto the stack. Pulling (popping) values of the stack will not clear them.
If you `JSR`, it will push the high byte of the PC to the address at SP (so `$01ff` at the start), and the low byte to SP-1 (`$01fe`). SP will be decremented to `$01fd`.
When you RTS, it will read back the PC to use from SP+1 and SP+2 (and increment the SP). So you better make sure you pull all values from the stack you pushed onto in in your subroutine!

Display (specific to our "machine")

The next four pages (`$0200-$05ff`) are reserved for the display; every pixel is a byte (although only 4 bits are used). 32 x 32 pixels.
Each page is 256 pixels, so 256/32 = 8 rows.
Fun: we'll need 16 bit arithmetic for this!

Colors

$0 Black	$1 White	$2 Red	$3 Cyan
$4 Purple	$5 Green	$6 Blue	$7 Yellow
$8 Orange	$9 Brown	$a L red	$b D gray
$c Gray	$d L green	$e L blue	$f L grey

Addition

ADC - Add with carry

We can only add with carry (which admittedly is better than only without), so, carry will always be included.

```
CLC      ; clear carry
LDA #$a0 ; load $a0 into A
ADC #$10 ; add $10 to A
         ; A will be $b0
         ; do not clear carry
ADC #$80 ; add $80 to A
         ; A will be $30
         ; Carry will be set
```

We can use this for multiple byte addition
For example 16 byte value at $00-$01

```
LDA $00  ; assumed 0
CLC
ADC #$90 ; A = $90
ADC #$90 ; A = $20, carry is set
STA $00  ; store low byte
LDA $01  ; load high byte
ADC #$00 ; Add carry to A → A = $01
STA $01  ; store high byte
```

Subtraction

SBC - Subtract with Carry

Subtracts including the NOT of the Carry (so in all ways the opposite of ADC). So you need to SET the carry before subtracting.

```
LDA #$10
SEC
SBC #$20 ; A = $f0; carry UNSET
SBC #$00 ; A = $ef
```

Signed values

They're fun, ask me later. We don't need them today.

Other assembler commands

Labels

A label creates a constant with the same name, the value is the address of the next instruction.

```
  LDX #$0f
loop:
  DEX
  BNE loop ; if Z = 0 branch
```

Define

You can use define to define a constant value. You can use this as at any place where you need a numerical value. Same as a label but you get to define the value.

define x $10
LDA x ; load value from $10 into A
LDA #x ; load $10 into A

EQU/ORG

Tell the assembler where to continue to put assembled bytes into memory.
By default we start at $0600 (just after the display memory), but we can use this to move stuff around on our terms.

```
  JMP $06a0
*=$06a0
  LDA #$10
```

DCB

Put out raw bytes

```
*=$0000
dcb $0, $1, $2, $3, $4, $5, $6, $7
dcb $8, $9, $a, $b, $c, $d, $e, $f
```

Note: this will be reset when you hit "reset" :(.