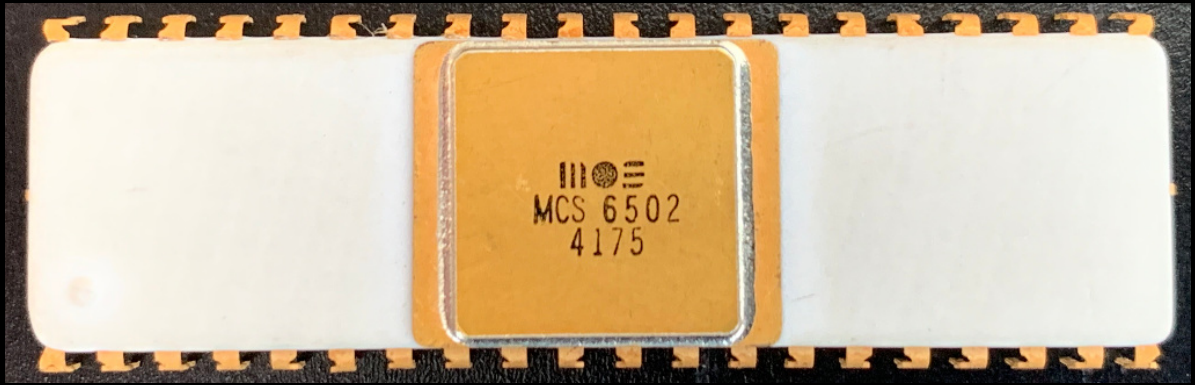
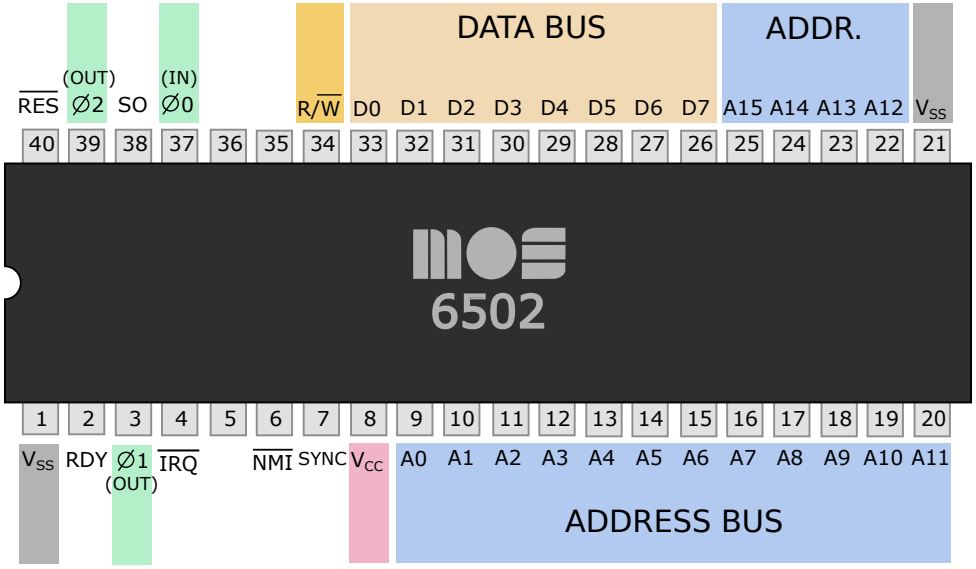
Advanced Assembly Programming for the Apple II

Stephen A. Edwards

6502 image from https://www.pagetable.com/?p=1295



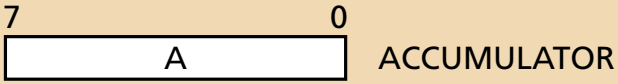
6502 image from https://www.pagetable.com/?p=1295



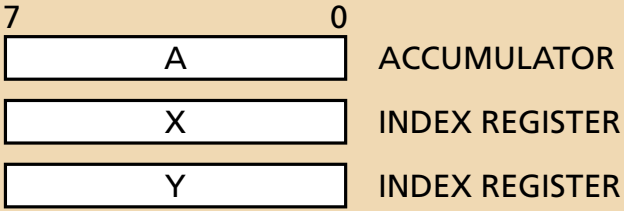
After Bill Bertram, Wikipedia

The 6502 Programmer's Model

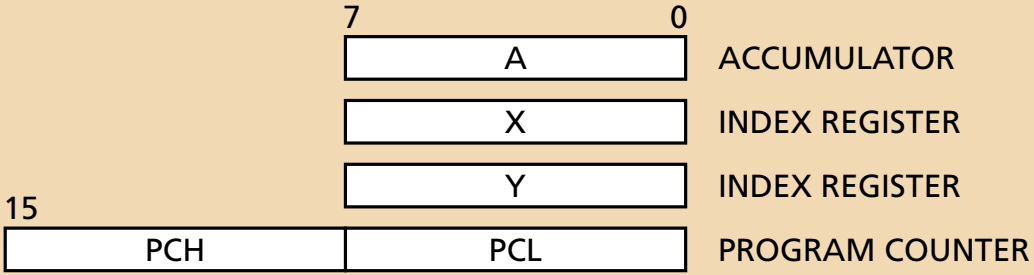
The 6502 Programmer's Model



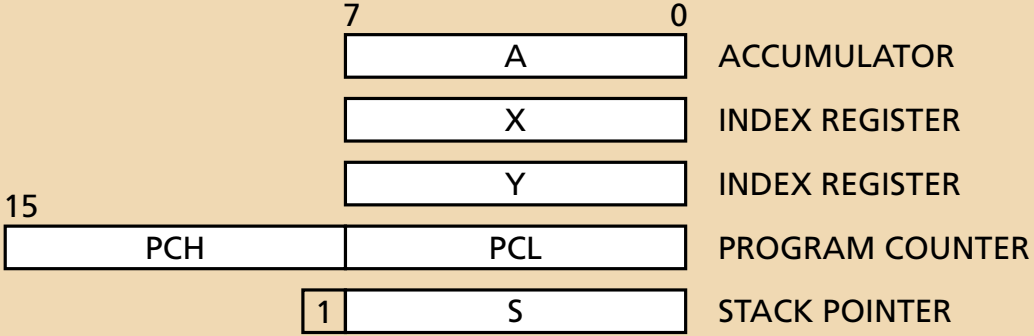
The 6502 Programmer's Model



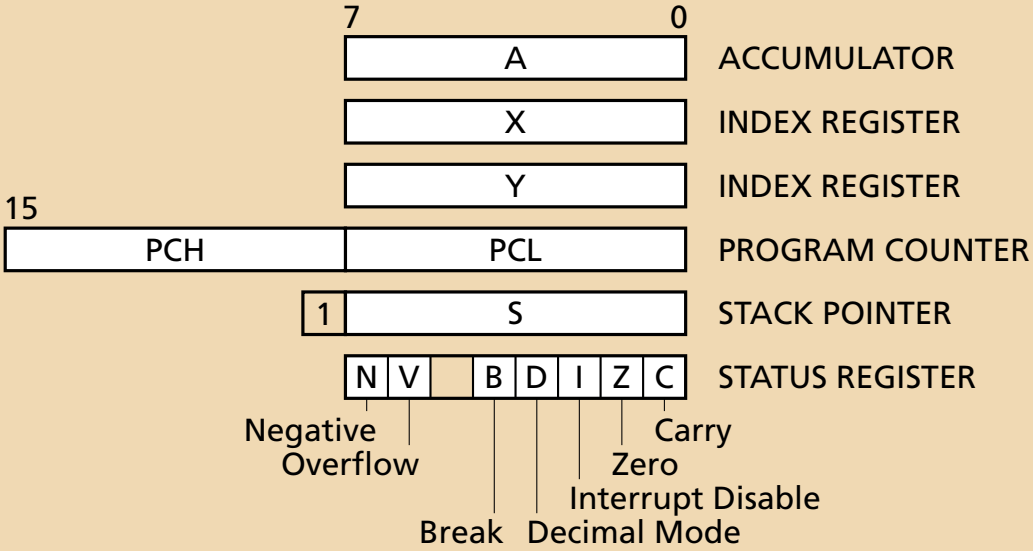
The 6502 Programmer's Model



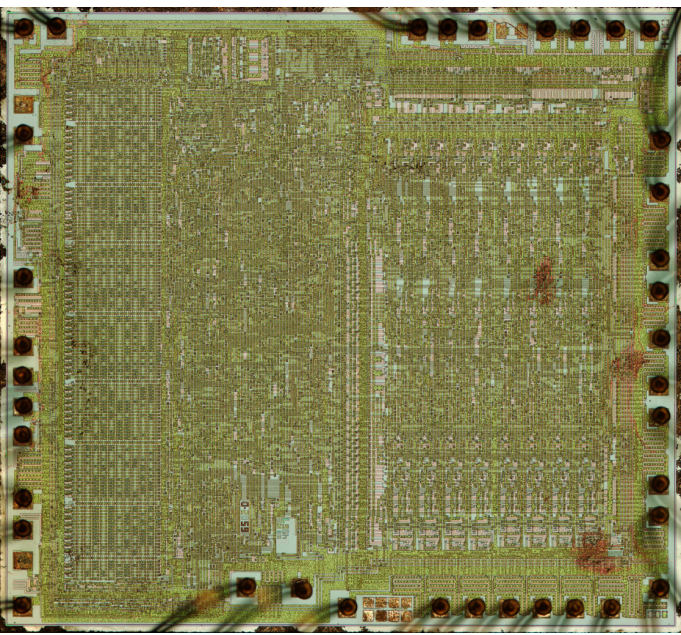
The 6502 Programmer's Model

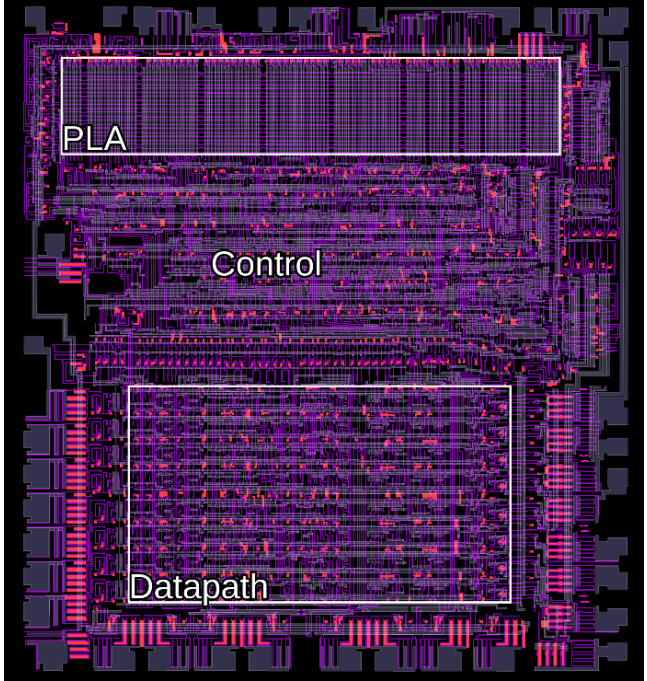


The 6502 Programmer's Model

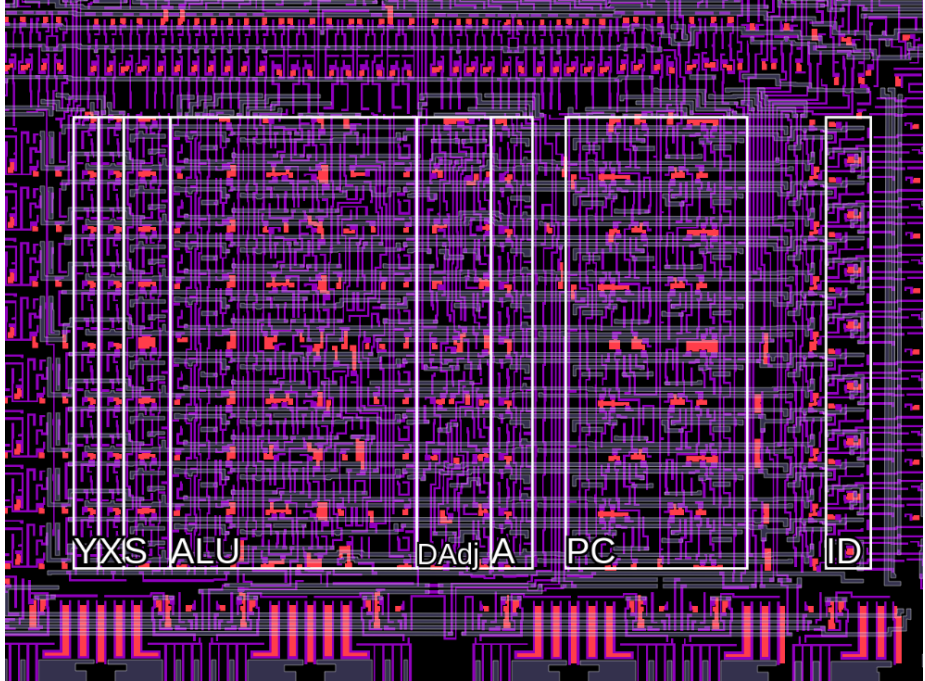


Source: Visual6502.org

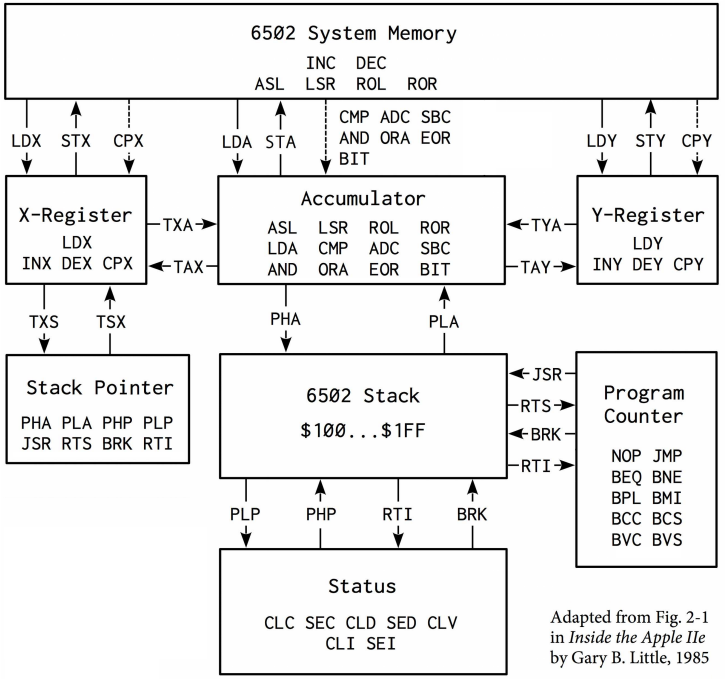




Source: Visual6502.org

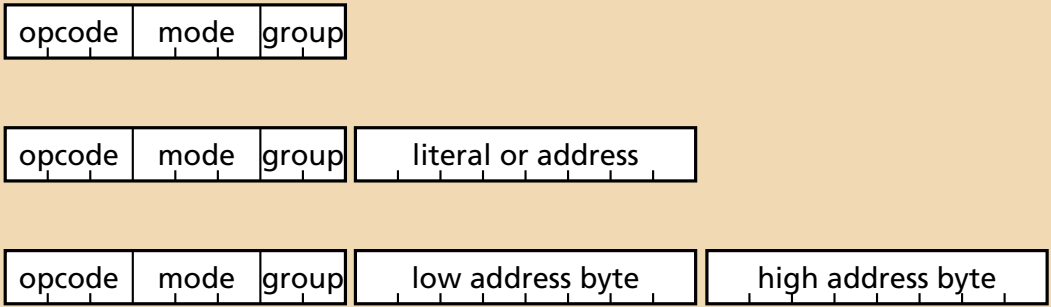


Source: Visual6502.org

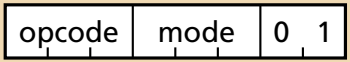


Adapted from Fig. 2-1 in *Inside the Apple IIe* by Gary B. Little, 1985

6502 Instruction Encoding

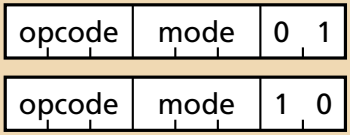


6502 Instruction Encoding

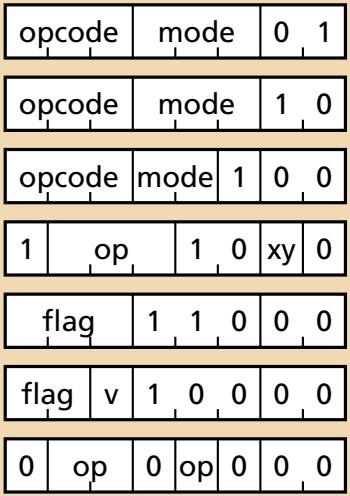


“Group one” add, compare; most addressing modes

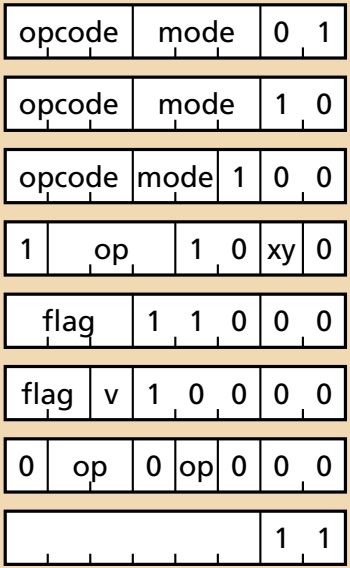
6502 Instruction Encoding



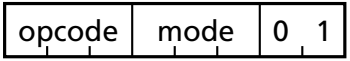
6502 Instruction Encoding



6502 Instruction Encoding

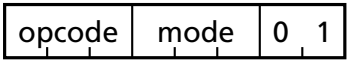


Group One Instructions

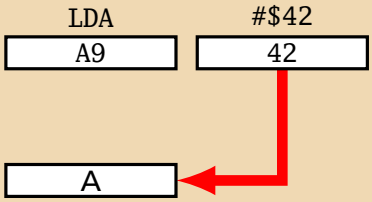



```
A9 42     LDA #$42     ;Load Accumulator     Immediate
```

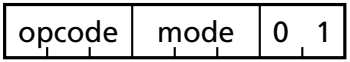
Group One Instructions



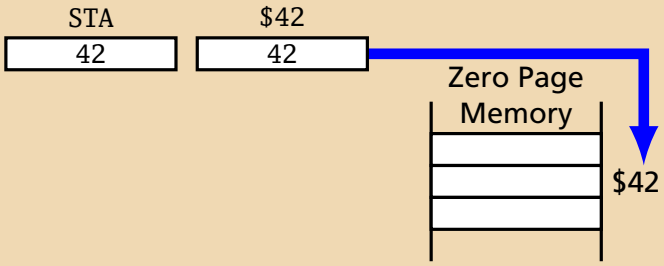

```
A9 42      LDA #$42      ;Load Accumulator      Immediate
```



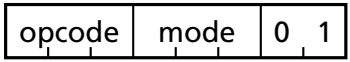
Group One Instructions



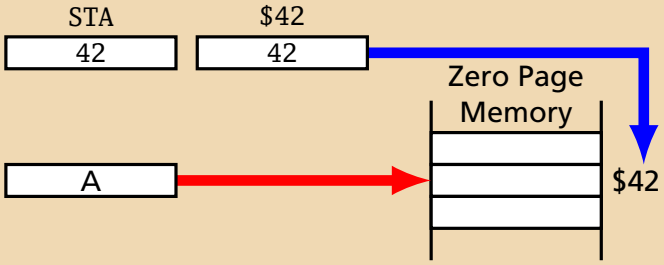

```
A9 42     LDA #$42     ;Load Accumulator     Immediate
85 42     STA $42      ;Store Accumulator    Zero Page
```



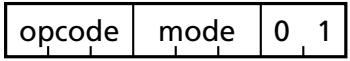
Group One Instructions



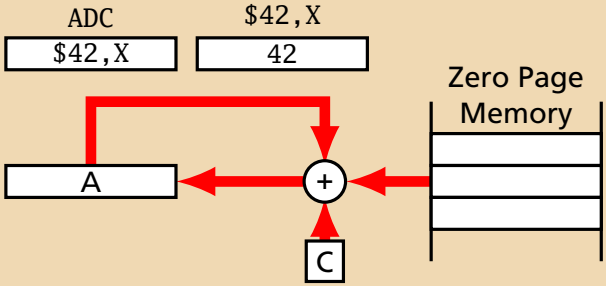

```
A9 42     LDA #$42     ;Load Accumulator     Immediate
85 42     STA $42      ;Store Accumulator    Zero Page
```



Group One Instructions



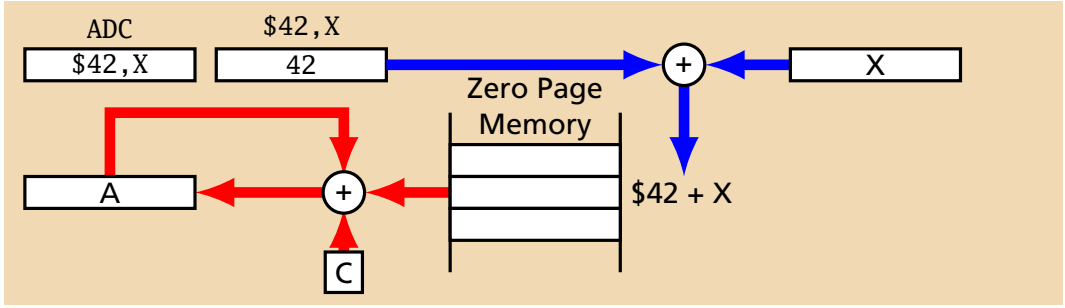

```
A9 42     LDA #$42     ;Load Accumulator      Immediate
85 42     STA $42      ;Store Accumulator     Zero Page
75 42     ADC $42,X    ;Add with Carry        Zero Page Indexed by X
```



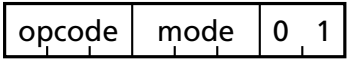
Group One Instructions

opcode	mode	0	1

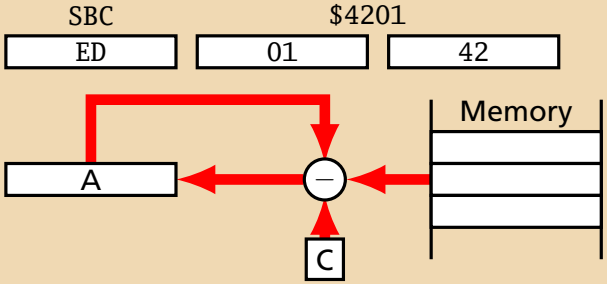
```
A9 42     LDA #$42     ;Load Accumulator      Immediate
85 42     STA $42      ;Store Accumulator     Zero Page
75 42     ADC $42,X    ;Add with Carry        Zero Page Indexed by X
```



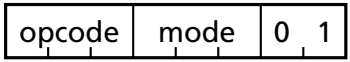
Group One Instructions



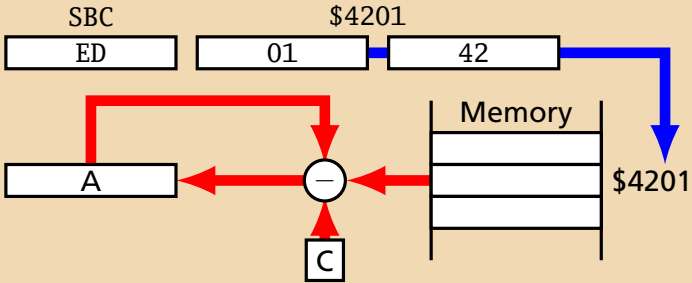

```
A9 42      LDA #$42      ;Load Accumulator      Immediate
85 42      STA $42       ;Store Accumulator     Zero Page
75 42      ADC $42,X     ;Add with Carry        Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry      Absolute
```



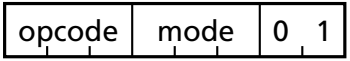
Group One Instructions



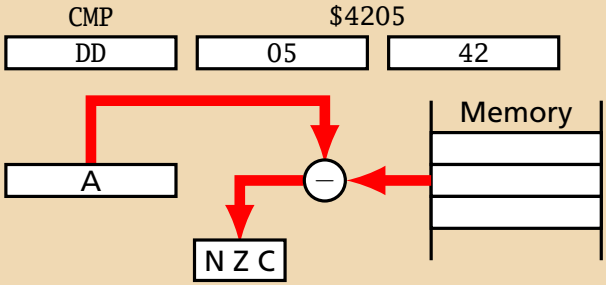

```
A9 42      LDA #$42      ;Load Accumulator      Immediate
85 42      STA $42       ;Store Accumulator     Zero Page
75 42      ADC $42,X     ;Add with Carry        Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry      Absolute
```



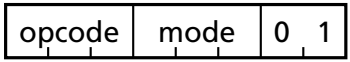
Group One Instructions



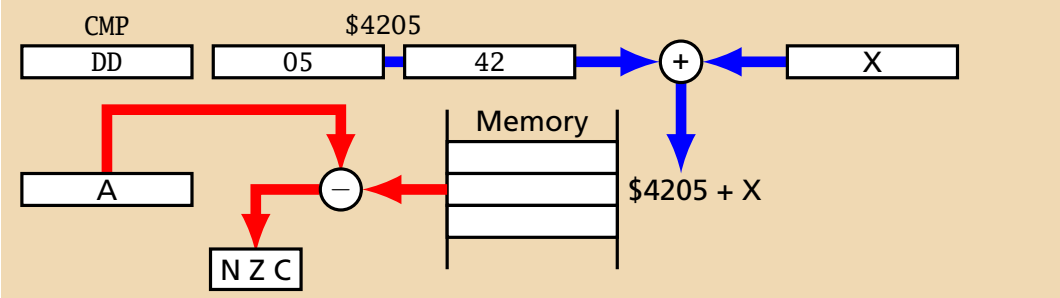

```
A9 42      LDA #$42      ;Load Accumulator      Immediate
85 42      STA $42       ;Store Accumulator     Zero Page
75 42      ADC $42,X     ;Add with Carry        Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry      Absolute
DD 05 42   CMP $4205,X   ;Compare               Absolute Indexed by X
```



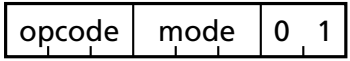
Group One Instructions



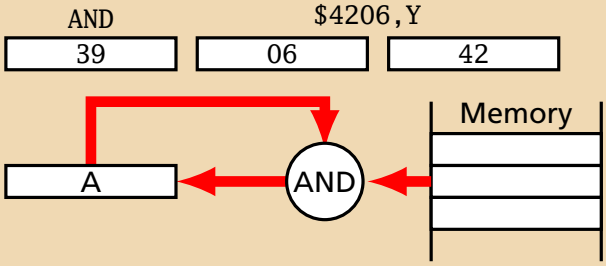

```
A9 42      LDA #$42      ;Load Accumulator      Immediate
85 42      STA $42       ;Store Accumulator     Zero Page
75 42      ADC $42,X     ;Add with Carry        Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry      Absolute
DD 05 42   CMP $4205,X   ;Compare               Absolute Indexed by X
```



Group One Instructions



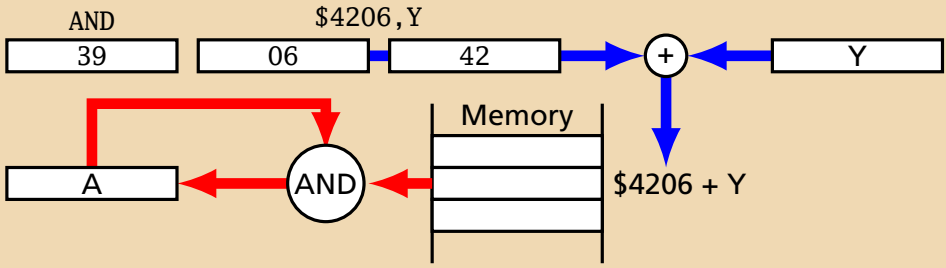

```
A9 42      LDA #$42      ;Load Accumulator      Immediate
85 42      STA $42       ;Store Accumulator     Zero Page
75 42      ADC $42,X     ;Add with Carry        Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry      Absolute
DD 05 42   CMP $4205,X   ;Compare               Absolute Indexed by X
39 06 42   AND $4206,Y   ;Logical AND           Absolute Indexed by Y
```



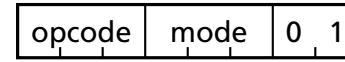
Group One Instructions

opcode	mode	0	1

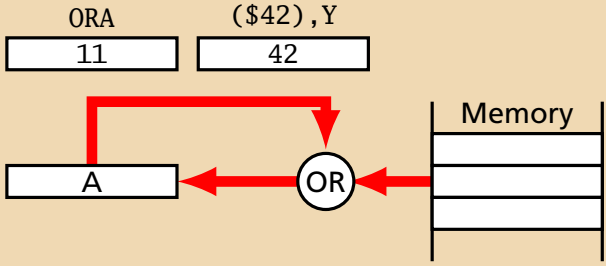
```
A9 42      LDA #$42      ;Load Accumulator      Immediate
85 42      STA $42       ;Store Accumulator     Zero Page
75 42      ADC $42,X     ;Add with Carry        Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry      Absolute
DD 05 42   CMP $4205,X   ;Compare               Absolute Indexed by X
39 06 42   AND $4206,Y   ;Logical AND           Absolute Indexed by Y
```



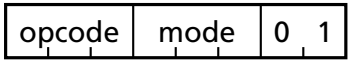
Group One Instructions



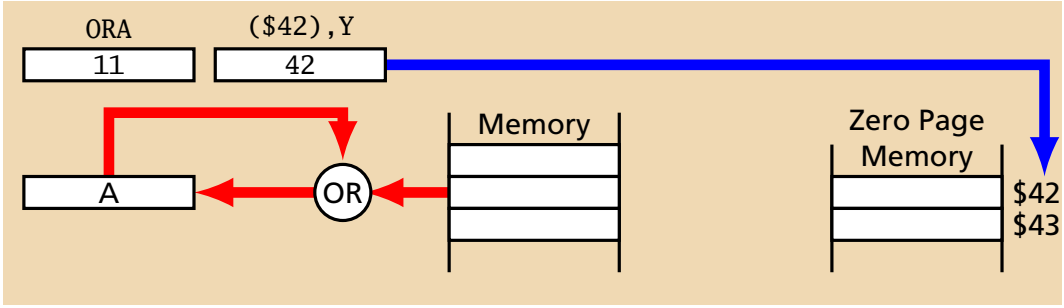

```
A9 42      LDA #$42      ;Load Accumulator      Immediate
85 42      STA $42       ;Store Accumulator     Zero Page
75 42      ADC $42,X     ;Add with Carry        Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry      Absolute
DD 05 42   CMP $4205,X   ;Compare               Absolute Indexed by X
39 06 42   AND $4206,Y   ;Logical AND           Absolute Indexed by Y
11 42      ORA ($42),Y   ;Logical OR            Indirect Indexed
```



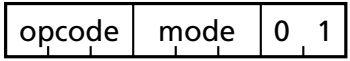
Group One Instructions



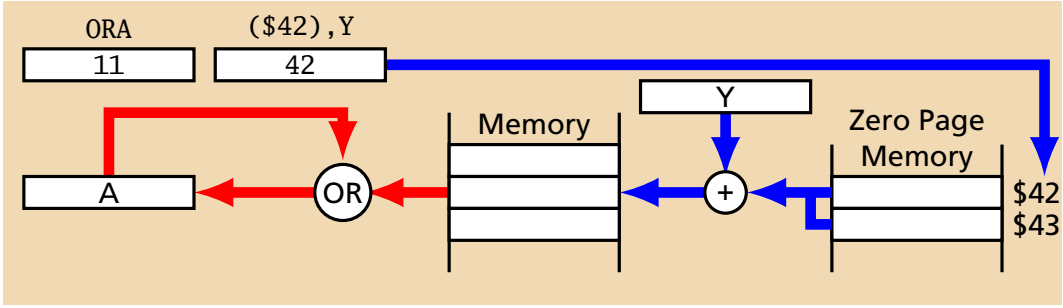

```
A9 42      LDA #$42      ;Load Accumulator       Immediate
85 42      STA $42       ;Store Accumulator      Zero Page
75 42      ADC $42,X     ;Add with Carry         Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry       Absolute
DD 05 42   CMP $4205,X   ;Compare                Absolute Indexed by X
39 06 42   AND $4206,Y   ;Logical AND            Absolute Indexed by Y
11 42      ORA ($42),Y   ;Logical OR             Indirect Indexed
```



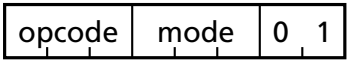
Group One Instructions



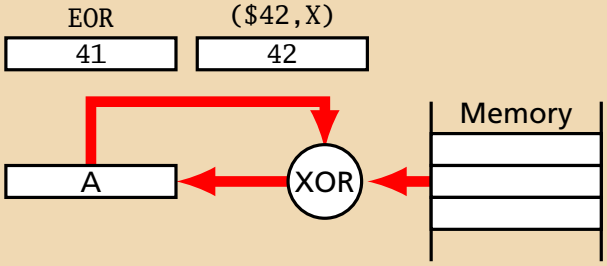

```
A9 42      LDA #$42      ;Load Accumulator      Immediate
85 42      STA $42       ;Store Accumulator     Zero Page
75 42      ADC $42,X     ;Add with Carry        Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry      Absolute
DD 05 42   CMP $4205,X   ;Compare               Absolute Indexed by X
39 06 42   AND $4206,Y   ;Logical AND           Absolute Indexed by Y
11 42      ORA ($42),Y   ;Logical OR            Indirect Indexed
```



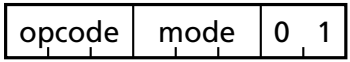
Group One Instructions



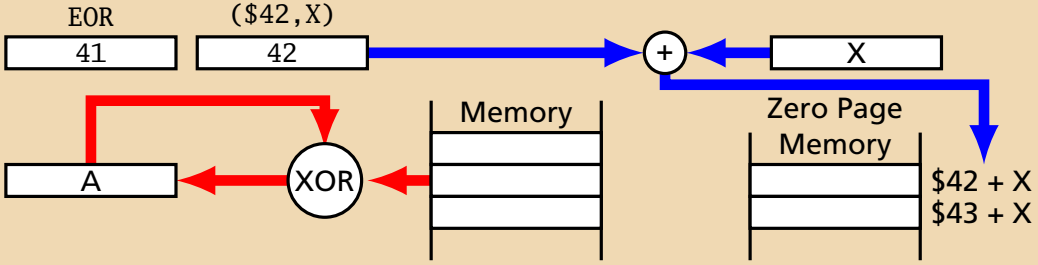

```
A9 42      LDA #$42      ;Load Accumulator      Immediate
85 42      STA $42       ;Store Accumulator     Zero Page
75 42      ADC $42,X     ;Add with Carry        Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry      Absolute
DD 05 42   CMP $4205,X   ;Compare               Absolute Indexed by X
39 06 42   AND $4206,Y   ;Logical AND           Absolute Indexed by Y
11 42      ORA ($42),Y   ;Logical OR            Indirect Indexed
41 42      EOR ($42,X)   ;Exclusive OR          Indexed Indirect
```



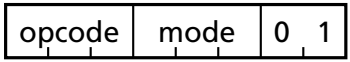
Group One Instructions



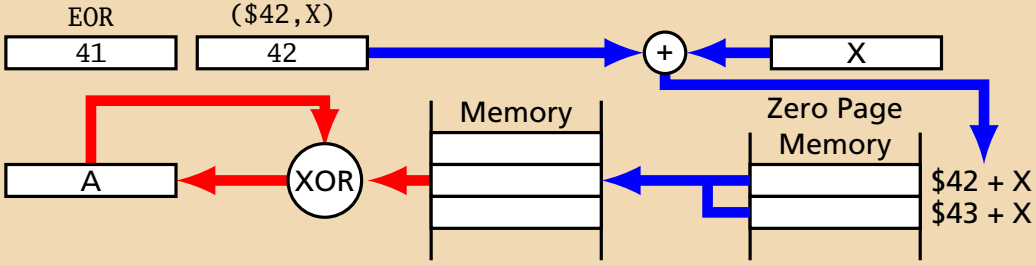

```
A9 42      LDA #$42      ;Load Accumulator       Immediate
85 42      STA $42       ;Store Accumulator      Zero Page
75 42      ADC $42,X     ;Add with Carry         Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry       Absolute
DD 05 42   CMP $4205,X   ;Compare                Absolute Indexed by X
39 06 42   AND $4206,Y   ;Logical AND            Absolute Indexed by Y
11 42      ORA ($42),Y   ;Logical OR             Indirect Indexed
41 42      EOR ($42,X)   ;Exclusive OR           Indexed Indirect
```



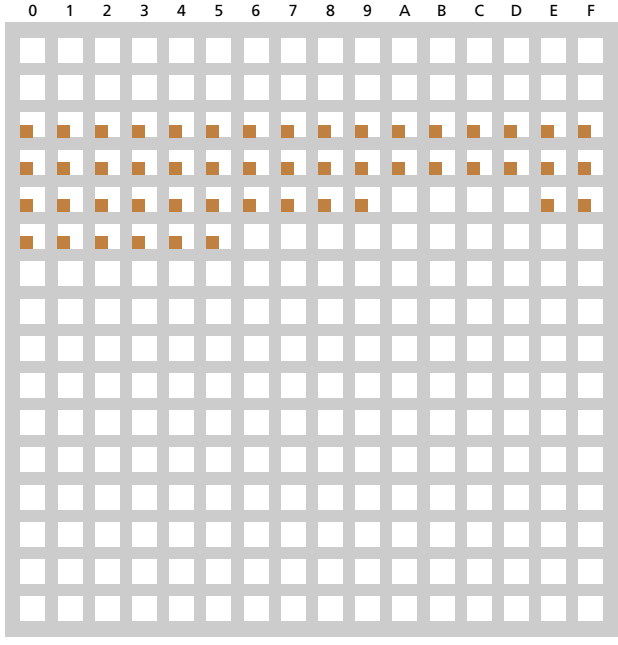
Group One Instructions




```
A9 42      LDA #$42      ;Load Accumulator       Immediate
85 42      STA $42       ;Store Accumulator      Zero Page
75 42      ADC $42,X     ;Add with Carry         Zero Page Indexed by X
ED 01 42   SBC $4201     ;Subtract w/Carry       Absolute
DD 05 42   CMP $4205,X   ;Compare                Absolute Indexed by X
39 06 42   AND $4206,Y   ;Logical AND            Absolute Indexed by Y
11 42      ORA ($42),Y   ;Logical OR             Indirect Indexed
41 42      EOR ($42,X)   ;Exclusive OR           Indexed Indirect
```



Apple II Zero Page Usage

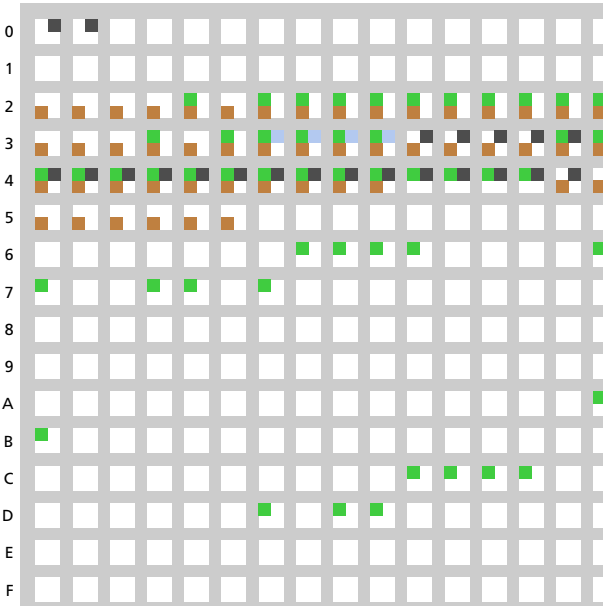


Monitor

Source: comp.sys.apple2 FAQ, Wagner, Wirth and Lechner

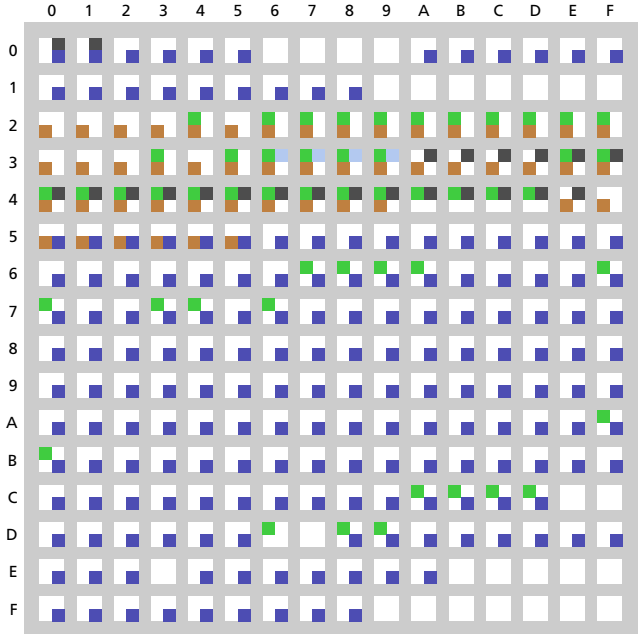
Apple II Zero Page Usage

DOS 3.3 ProDOS Monitor



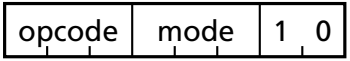
Source: comp.sys.apple2 FAQ, Wagner, Wirth and Lechner

Apple II Zero Page Usage

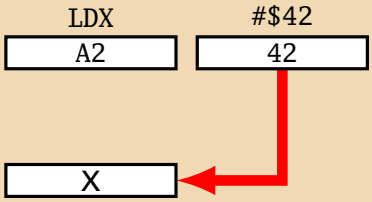


Source: comp.sys.apple2 FAQ, Wagner, Wirth and Lechner

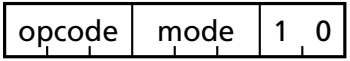
Group Two Instructions



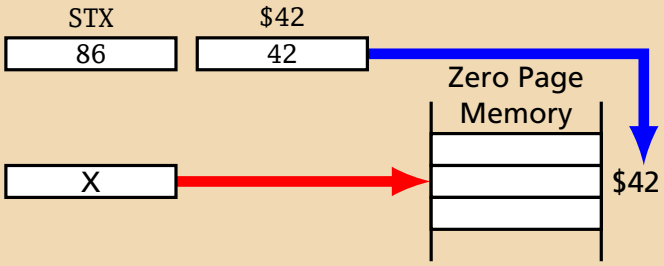

```
A2 42      LDX #$42      ;Load X                    Immediate
```



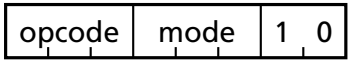
Group Two Instructions



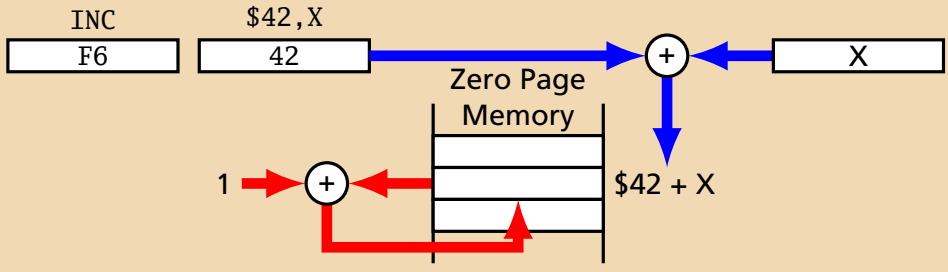

```
A2 42     LDX #$42     ;Load X          Immediate
86 42     STX $42      ;Store X         Zero Page
```



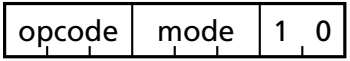
Group Two Instructions



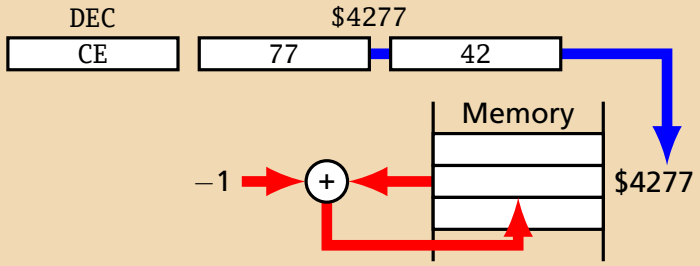

```
A2 42     LDX #$42     ;Load X          Immediate
86 42     STX $42      ;Store X         Zero Page
F6 42     INC $42,X    ;Increment       Zero Page Indexed by X
```



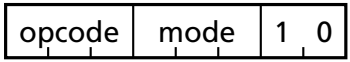
Group Two Instructions



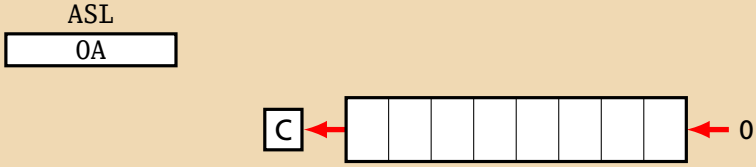

```
A2 42      LDX #$42     ;Load X          Immediate
86 42      STX $42      ;Store X         Zero Page
F6 42      INC $42,X    ;Increment       Zero Page Indexed by X
CE 77 42   DEC $4277    ;Decrement       Absolute
```



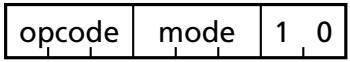
Group Two Instructions



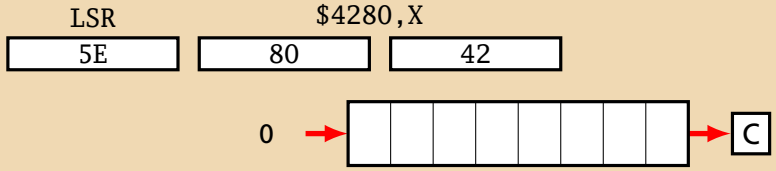

```
A2 42      LDX #$42      ;Load X                    Immediate
86 42      STX $42       ;Store X                   Zero Page
F6 42      INC $42,X     ;Increment                 Zero Page Indexed by X
CE 77 42   DEC $4277     ;Decrement                 Absolute
0A         ASL           ;Arithmetic Shift Left     Accumulator
```



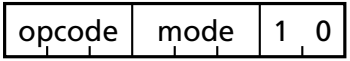
Group Two Instructions



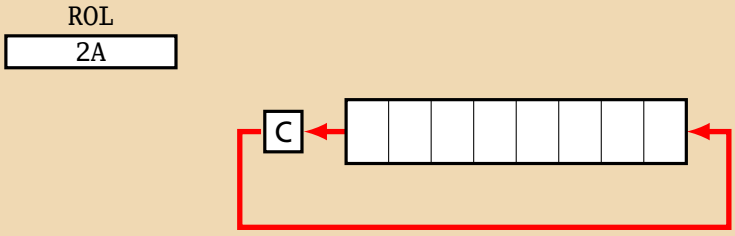

```
A2 42      LDX #$42      ;Load X                  Immediate
86 42      STX $42       ;Store X                 Zero Page
F6 42      INC $42,X     ;Increment               Zero Page Indexed by X
CE 77 42   DEC $4277     ;Decrement               Absolute
0A         ASL           ;Arithmetic Shift Left   Accumulator
5E 80 42   LSR $4280,X   ;Logical Shift Right     Absolute Indexed by X
```



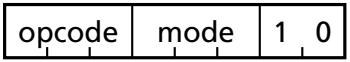
Group Two Instructions



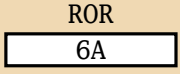

```
A2 42      LDX #$42      ;Load X                Immediate
86 42      STX $42       ;Store X               Zero Page
F6 42      INC $42,X     ;Increment             Zero Page Indexed by X
CE 77 42   DEC $4277     ;Decrement             Absolute
0A         ASL           ;Arithmetic Shift Left Accumulator
5E 80 42   LSR $4280,X   ;Logical Shift Right   Absolute Indexed by X
2A         ROL           ;Rotate Left           Accumulator
```

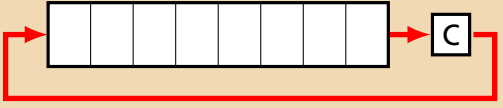


Group Two Instructions

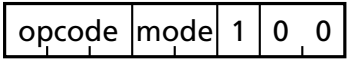



```
A2 42      LDX #$42      ;Load X                Immediate
86 42      STX $42       ;Store X               Zero Page
F6 42      INC $42,X     ;Increment             Zero Page Indexed by X
CE 77 42   DEC $4277     ;Decrement             Absolute
0A         ASL           ;Arithmetic Shift Left Accumulator
5E 80 42   LSR $4280,X   ;Logical Shift Right   Absolute Indexed by X
2A         ROL           ;Rotate Left           Accumulator
6A         ROR           ;Rotate Right          Accumulator
```



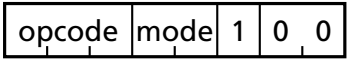


Other Instructions

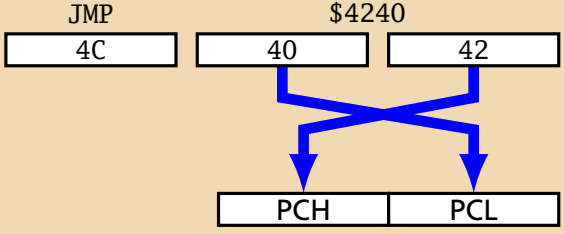



```
A0 42      LDY #$42      ;Load Y          Immediate
94 42      STY $42,X     ;Store Y         Zero Page Indexed by X
C4 42      CPY $42       ;Compare Y       Zero Page
EC 50 42   CPX $4250     ;Compare X       Absolute
```

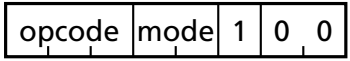
Other Instructions



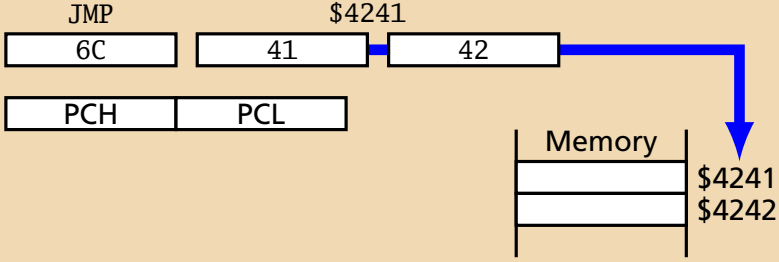

```
A0 42      LDY #$42     ;Load Y          Immediate
94 42      STY $42,X    ;Store Y         Zero Page Indexed by X
C4 42      CPY $42      ;Compare Y       Zero Page
EC 50 42   CPX $4250    ;Compare X       Absolute
4C 40 42   JMP $4240    ;Jump            Absolute
```



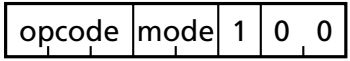
Other Instructions



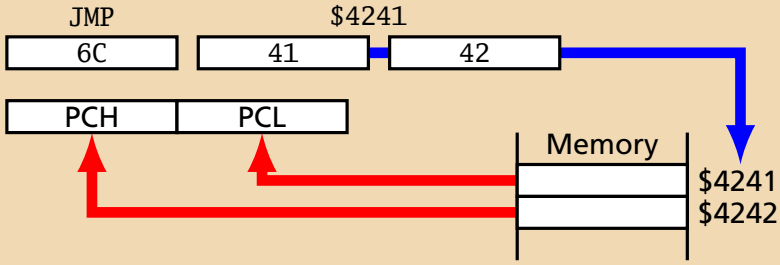

```
A0 42      LDY #$42      ;Load Y          Immediate
94 42      STY $42,X     ;Store Y         Zero Page Indexed by X
C4 42      CPY $42       ;Compare Y       Zero Page
EC 50 42   CPX $4250     ;Compare X       Absolute
4C 40 42   JMP $4240     ;Jump            Absolute
6C 41 42   JMP ($4241)   ;Jump            Indirect
```



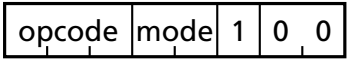
Other Instructions



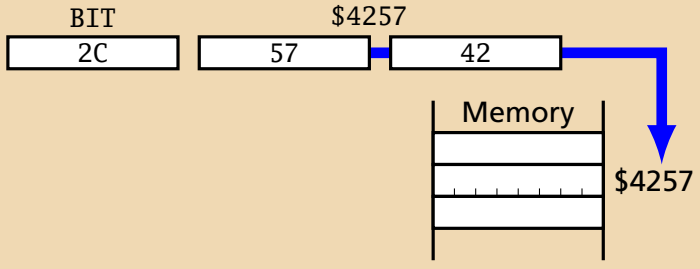

```
A0 42      LDY #$42      ;Load Y          Immediate
94 42      STY $42,X     ;Store Y         Zero Page Indexed by X
C4 42      CPY $42       ;Compare Y       Zero Page
EC 50 42   CPX $4250     ;Compare X       Absolute
4C 40 42   JMP $4240     ;Jump            Absolute
6C 41 42   JMP ($4241)   ;Jump            Indirect
```



Other Instructions



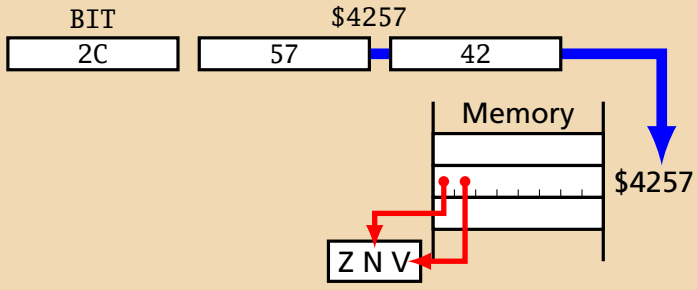

```
A0 42      LDY #$42      ;Load Y              Immediate
94 42      STY $42,X     ;Store Y             Zero Page Indexed by X
C4 42      CPY $42       ;Compare Y           Zero Page
EC 50 42   CPX $4250     ;Compare X           Absolute
4C 40 42   JMP $4240     ;Jump                Absolute
6C 41 42   JMP ($4241)   ;Jump                Indirect
2C 57 42   BIT $4257     ;Test Bits w/ Accum. Absolute
```



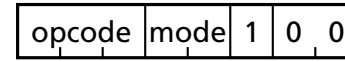
Other Instructions

opcode	mode	1	0	0

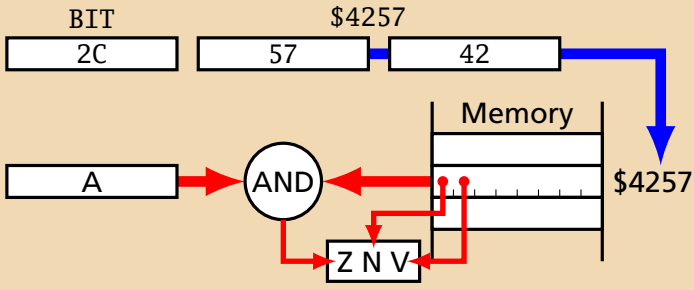
```
A0 42      LDY #$42      ;Load Y              Immediate
94 42      STY $42,X     ;Store Y             Zero Page Indexed by X
C4 42      CPY $42       ;Compare Y           Zero Page
EC 50 42   CPX $4250     ;Compare X           Absolute
4C 40 42   JMP $4240     ;Jump                Absolute
6C 41 42   JMP ($4241)   ;Jump                Indirect
2C 57 42   BIT $4257     ;Test Bits w/ Accum. Absolute
```



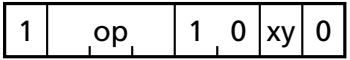
Other Instructions




```
A0 42      LDY #$42      ;Load Y              Immediate
94 42      STY $42,X     ;Store Y             Zero Page Indexed by X
C4 42      CPY $42       ;Compare Y           Zero Page
EC 50 42   CPX $4250     ;Compare X           Absolute
4C 40 42   JMP $4240     ;Jump                Absolute
6C 41 42   JMP ($4241)   ;Jump                Indirect
2C 57 42   BIT $4257     ;Test Bits w/ Accum. Absolute
```

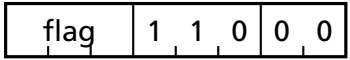


Single-Byte Data Instructions




```
E8   INX  ;Increment X
CA   DEX  ;Decrement X
C8   INY  ;Increment Y
88   DEY  ;Decrement Y
AA   TAX  ;Transfer Accumulator to X
A8   TAY  ;Transfer Accumulator to Y
8A   TXA  ;Transfer X to Accumulator
98   TYA  ;Transfer Y to Accumulator
BA   TSX  ;Transfer Stack Pointer to X
9A   TXS  ;Transfer X to Stack Pointer
```

Single-Byte Flag Instructions



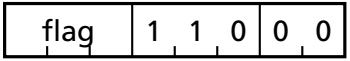

```
18    CLC ;Clear Carry Flag
38    SEC ;Set Carry Flag
58    CLI ;Clear Interrupt Disable
78    SEI ;Set Interrupt Disable
B8    CLV ;Clear Overflow Flag
```

Set and clear the carry flag; useful with **ADC** and **SBC**

Enable and disable interrupts. Useful for critical regions

Clear the overflow flag

Single-Byte Flag Instructions




```
18   CLC ;Clear Carry Flag
38   SEC ;Set Carry Flag
58   CLI ;Clear Interrupt Disable
78   SEI ;Set Interrupt Disable
B8   CLV ;Clear Overflow Flag
D8   CLD ;Clear Decimal Mode     (ADC, SBC perform binary arithmetic)
F8   SED ;Set Decimal Mode       (ADC, SBC perform BCD arithemtic)
```

In decimal mode, **ADC** and **SBC** perform BCD arithmetic.

In decimal mode, $19 + $1 = $20

In binary mode, $19 + $1 = $1A

US Patent 3,991,307 (1976) describes the 6502's decimal adjust logic

The 6502-based Ricoh 2A03 used in the Famicom/Nintendo Entertainment System does not support decimal mode

Branch Instructions

flag	v	1	0	0	0	0

```
10 FE   START   BPL START   ;Branch on Plus            N=0
30 0A           BMI END     ;Branch on Minus           N=1
50 FA           BVC START   ;Branch on Overflow Clear  V=0
70 06           BVS END     ;Branch on Overflow Set    V=1
90 F6           BCC START   ;Branch on Carry Clear     C=0
B0 02           BCS END     ;Branch on Carry Set       C=1
D0 F2           BNE START   ;Branch on Not Equal       Z=0
F0 FE   END     BEQ END     ;Branch on Equal           Z=1
```

Conditionally branch depending on the state of one of the four flags

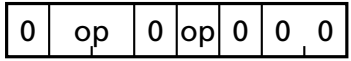
Branch destination is program-counter-relative: between −128 and 127 bytes

Use **JMP** to branch farther

BNE and **BEQ** could be called “branch on non-zero” and “branch on zero”

The 65C02 added the unconditional **BRA** instruction

Stack Instructions



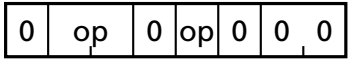

```
20 ED FD   JSR $FDED   ;Jump to Subroutine
60         RTS         ;Return from Subroutine
```

For subroutine linkage:

JSR $FDED pushes the program counter onto the stack then jumps to $FDED

RTS pops the program counter value off the stack, returning to just after the **JSR** that sent it there

Stack Instructions

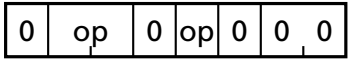



```
20 ED FD   JSR $FDED   ;Jump to Subroutine
60         RTS         ;Return from Subroutine
40         RTI         ;Return from Interrupt
```

Return from an interrupt by popping the status register and the program counter from the stack. (Like **RTS**, but also restores the status register)

Useful in interrupt routines and returning from **BRK**

Stack Instructions



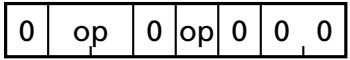

```
20 ED FD   JSR $FDED   ;Jump to Subroutine
60         RTS         ;Return from Subroutine
40         RTI         ;Return from Interrupt
08         PHP         ;Push Processor Status
28         PLP         ;Pull Processor Status
48         PHA         ;Push Accumulator
68         PLA         ;Pull Accumulator
```

Save and restore the processor status on the stack

Save and restore the accumulator on the stack

Typical processors encourage functions to save registers on the stack, but the 6502's cramped (256 byte) stack demands parsimony

Stack Instructions




```
20 ED FD   JSR $FDED   ;Jump to Subroutine
60         RTS         ;Return from Subroutine
40         RTI         ;Return from Interrupt
08         PHP         ;Push Processor Status
28         PLP         ;Pull Processor Status
48         PHA         ;Push Accumulator
68         PLA         ;Pull Accumulator
00         BRK         ;Break
EA         NOP         ;No operation
```

BRK is a software interrupt that pushes the current program counter and status register on the stack before jumping to the break vector stored (in ROM) at $FFFE.

NOP does nothing for two cycles. Useful in delay loops or to patch a program in memory.


MERLIN-8 2.58
By Glen Bredon
C :Catalog
L :Load source
S :Save source
A :Append file
D :Disk command
E :Enter ED/ASM
O :Save object code
@ :Set date
Q :Quit
Source: A$0901,L$0000
Prefix: /MERLIN.8/
%

```
Editor

:ASM
Assembling
                1           ORG   $8000      ; Set code origin
                2
                3   COUT    EQU   $FDED      ; Define COUT label (character ou
t)
                4
8000: A2 00     5   ENTRY   LDX   #0         ; Use X because COUT leaves it un
changed
8002: BD 0E 80  6   LOOP    LDA   MSG,X      ; Get character from string
8005: F0 06     7           BEQ   DONE       ; Are we at the end?
8007: 20 ED FD  8           JSR   COUT       ; No: print the character
800A: E8        9           INX              ; Go to next character in string
800B: D0 F5     10          BNE   LOOP       ; A trick: always taken
800D: 60        11  DONE    RTS              ; Return
                12
800E: C8 E5 EC  13  MSG     ASC   "Hello World!" ; Generate ASCII character co
des
8011: EC EF A0 D7
8015: EF F2 EC E4
8019: A1
801A: 8D
```


MERLIN-8 2.58
By Glen Bredon
C :Catalog
L :Load source
S :Save source
A :Append file
D :Disk command
E :Enter ED/ASM
O :Save object code
@ :Set date
Q :Quit
Source: A$0901,L$01BC
Object: A$8000,L$001C,BIN
Prefix: /MERLIN.8/
Disk command:BRUN HELLO
Hello World!
Disk command:

```
1       ORG   $8000      ; Set code origin
COUT    EQU   $FDED      ; Define COUT label (character out)

ENTRY   LDX   #0         ; Use X because COUT leaves it unchanged
LOOP    LDA   MSG,X      ; Get character from string
        BEQ   DONE       ; Are we at the end?
        JSR   COUT       ; No: print the character
        INX              ; Go to next character in string
        BNE   LOOP       ; A trick: always taken
DONE    RTS              ; Return

MSG     ASC   "Hello World!" ; Generate ASCII character codes
        DB    $8D,00     ; Define byte: 8D is return, 00 is end
```

Merlin Editor Cheatsheet

OA-Q	Quit
Ctrl-D	Delete character
Ctrl-I	Toggle insert mode
Ctrl-B	To beginning of line
Ctrl-N	To end of line
OA-D	Delete current line
OA-I	Insert blank line
OA-B	Go to first line
OA-N	Go to last line
OA-8	Line of asterisks

Hello World in 6502 Assembly

```
          ORG $8000       ; Set code origin

COUT      EQU $FDED       ; Define COUT label (character out)

ENTRY     LDX #0          ; Use X because COUT leaves it unchanged
LOOP      LDA MSG,X       ; Get character from string
          BEQ DONE        ; Are we at the end?
          JSR COUT        ; No: print the character
          INX             ; Go to next character in string
          BNE LOOP        ; A trick: always taken
DONE      RTS             ; Return

MSG       ASC "Hello World!" ; Generate ASCII character codes
          DB $8D,00       ; Define byte: 8D is return, 00 is end
```

Printed Assembler Output

```
                    1             ORG   $8000      ; Set code origin
                    2
                    3    COUT     EQU   $FDED      ; Define COUT label (character out)
                    4
8000: A2 00         5    ENTRY    LDX   #0         ; Use X because COUT leaves it unchanged
8002: BD 0E 80      6    LOOP     LDA   MSG,X      ; Get character from string
8005: F0 06         7             BEQ   DONE       ; Are we at the end?
8007: 20 ED FD      8             JSR   COUT       ; No: print the character
800A: E8            9             INX             ; Go to next character in string
800B: D0 F5         10            BNE   LOOP       ; A trick: always taken
800D: 60            11   DONE     RTS             ; Return
                    12
800E: C8 E5 EC      13   MSG      ASC   "Hello World!" ; Generate ASCII character codes
8011: EC EF A0 D7
8015: EF F2 EC E4
8019: A1
801A: 8D 00         14            DB    $8D,00     ; Define byte: 8D is return, 00 is end

--End assembly, 28 bytes, Errors: 0

Symbol table - alphabetical order:

   COUT    =$FDED      DONE    =$800D   ?  ENTRY   =$8000      LOOP    =$8002
   MSG     =$800E

Symbol table - numerical order:

?  ENTRY   =$8000      LOOP    =$8002      DONE    =$800D      MSG     =$800E
   COUT    =$FDED
```

From the Linapple emulator

```
]CALL -151

*8000L
8000-   A2 00        LDX   #$00
8002-   BD 0E 80     LDA   $800E,X
8005-   F0 06        BEQ   $800D
8007-   20 ED FD     JSR   $FDED
800A-   E8           INX
800B-   D0 F5        BNE   $8002
800D-   60           RTS
800E-   C8           INY
800F-   E5 EC        SBC   $EC
8011-   EC EF A0     CPX   $A0EF
8014-   D7           ???
8015-   EF           ???
8016-   F2           ???
8017-   EC E4 A1     CPX   $A1E4
801A-   8D 00 FF     STA   $FF00
801D-   FF           ???
801E-   00           BRK
801F-   00           BRK
8020-   FF           ???
8021-   FF           ???
*
```

Bouncing Balls

do

Clear and display the Hires 1 screen

Bouncing Balls

do

Clear and display the Hires 1 screen
Draw the frame

Bouncing Balls

do

Clear and display the Hires 1 screen
Draw the frame
Initialize ball locations and velocities

Bouncing Balls

do

Clear and display the Hires 1 screen
Draw the frame
Initialize ball locations and velocities
Draw each ball on the screen

Bouncing Balls

do
    Clear and display the Hires 1 screen
    Draw the frame
    Initialize ball locations and velocities
    Draw each ball on the screen
    do
        for *each ball* **do**

Bouncing Balls

```
do
    Clear and display the Hires 1 screen
    Draw the frame
    Initialize ball locations and velocities
    Draw each ball on the screen
    do
        for each ball do
            Erase the ball
```

Bouncing Balls

do
    Clear and display the Hires 1 screen
    Draw the frame
    Initialize ball locations and velocities
    Draw each ball on the screen
    do
        for *each ball* **do**
            Erase the ball
            Update the ball's horizontal position, possibly bouncing

Bouncing Balls

do
    Clear and display the Hires 1 screen
    Draw the frame
    Initialize ball locations and velocities
    Draw each ball on the screen
    do
        for *each ball* **do**
            Erase the ball
            Update the ball's horizontal position, possibly bouncing
            Apply gravity to the ball's vertical velocity

Bouncing Balls

do
    Clear and display the Hires 1 screen
    Draw the frame
    Initialize ball locations and velocities
    Draw each ball on the screen
    do
        for *each ball* **do**
            Erase the ball
            Update the ball's horizontal position, possibly bouncing
            Apply gravity to the ball's vertical velocity
            Update the ball's vertical position, bouncing at bottom

Bouncing Balls

```
do
    Clear and display the Hires 1 screen
    Draw the frame
    Initialize ball locations and velocities
    Draw each ball on the screen
    do
        for each ball do
            Erase the ball
            Update the ball's horizontal position, possibly bouncing
            Apply gravity to the ball's vertical velocity
            Update the ball's vertical position, bouncing at bottom
            Draw the ball
```

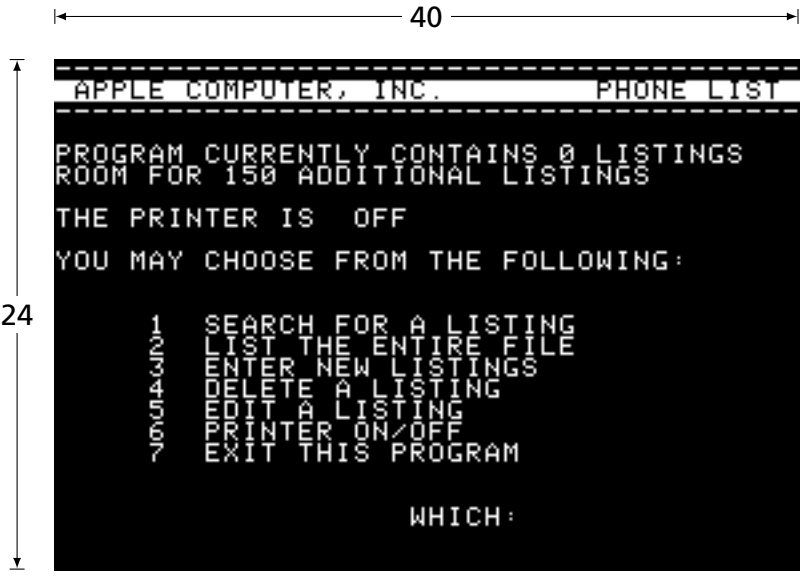
Bouncing Balls

do
    Clear and display the Hires 1 screen
    Draw the frame
    Initialize ball locations and velocities
    Draw each ball on the screen
    do
        for *each ball* **do**
            Erase the ball
            Update the ball's horizontal position, possibly bouncing
            Apply gravity to the ball's vertical velocity
            Update the ball's vertical position, bouncing at bottom
            Draw the ball
    while *no key pressed*

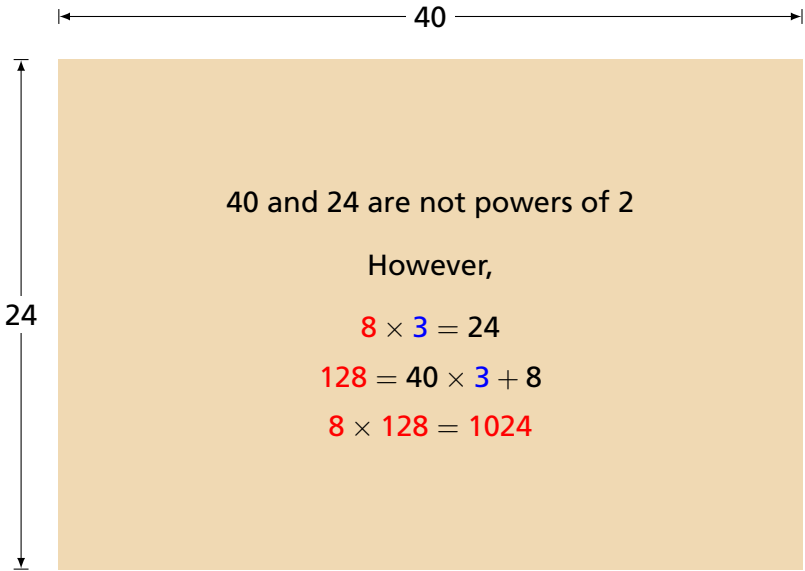
Bouncing Balls

do
    Clear and display the Hires 1 screen
    Draw the frame
    Initialize ball locations and velocities
    Draw each ball on the screen
    do
        for *each ball* **do**
            Erase the ball
            Update the ball's horizontal position, possibly bouncing
            Apply gravity to the ball's vertical velocity
            Update the ball's vertical position, bouncing at bottom
            Draw the ball
    while *no key pressed*
while *the "R" key was pressed*

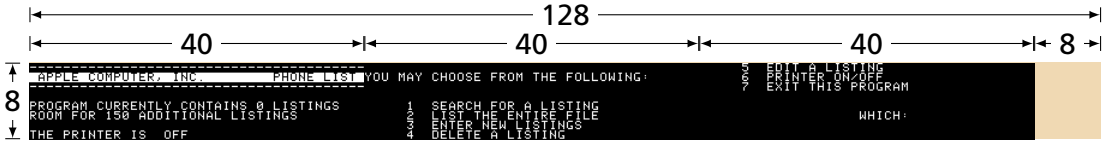
Apple II Text Mode: 40 × 24, 7 × 8 characters, 1 byte/char



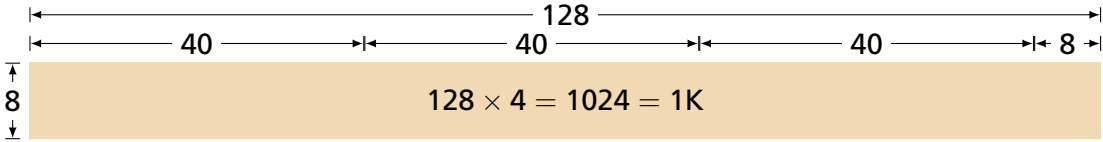
Apple II Text Mode: 40×24, 7×8 characters, 1 byte/char



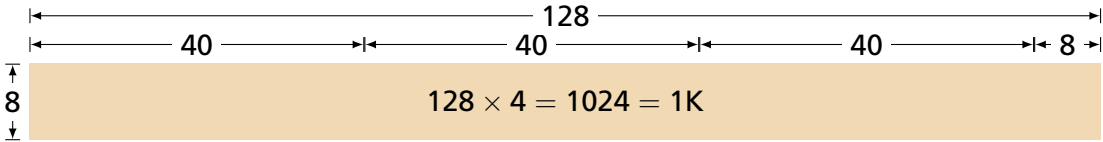
Apple II Text Mode: 40 × 24, 7 × 8 characters, 1 byte/char



Apple II Text Mode: 40×24, 7×8 characters, 1 byte/char

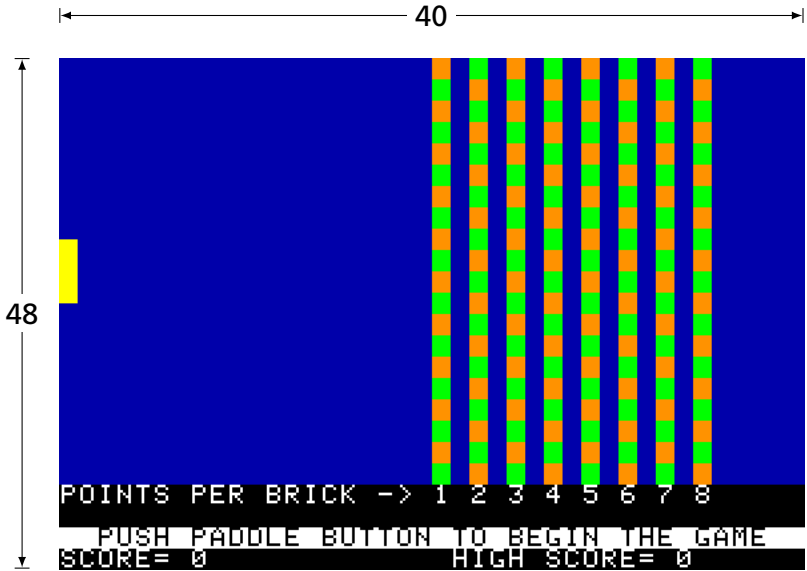


Apple II Text Mode: 40 × 24, 7 × 8 characters, 1 byte/char



$400	$428	$450	$478
$480	$4A8	$4D0	$4F8
$500	$528	$550	$578
$580	$5A8	$5D0	$5F8
$600	$628	$650	$678
$680	$6A8	$6D0	$6F8
$700	$728	$750	$778
$780	$7A8	$7D0	$7F8

Apple II Lores: 40 × 48, 16 colors, 4 bits/pixel, 1K/screen




THE ORIGINAL ROOT BEER
Hires
Hires
ROOT BEER

Apple II Hires: 140 × 192, 6 colors, 8K/screen

Apple II Hires Addresses: Even more bizarre

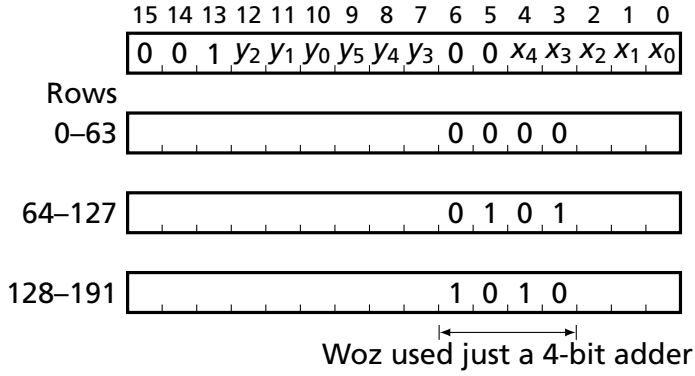
Y	Hex	Binary
0	$2000	1 000 000 0000000
1	$2400	1 001 000 0000000
2	$2800	1 010 000 0000000
3	$2C00	1 011 000 0000000
4	$3000	1 100 000 0000000
5	$3400	1 101 000 0000000
6	$3800	1 110 000 0000000
7	$3C00	1 111 000 0000000

Apple II Hires Addresses: Even more bizarre

Y	Hex	Binary
0	$2000	1 000 000 0000000
1	$2400	1 001 000 0000000
2	$2800	1 010 000 0000000
3	$2C00	1 011 000 0000000
4	$3000	1 100 000 0000000
5	$3400	1 101 000 0000000
6	$3800	1 110 000 0000000
7	$3C00	1 111 000 0000000
8	$2080	1 000 001 0000000
9	$2480	1 001 001 0000000
10	$2880	1 010 001 0000000
11	$2C80	1 011 001 0000000
12	$3080	1 100 001 0000000
13	$3480	1 101 001 0000000
14	$3880	1 110 001 0000000
15	$3C80	1 111 001 0000000

Apple II Hires Addresses: Even more bizarre

Y	Hex	Binary
0	$2000	1 000 000 0000000
1	$2400	1 001 000 0000000
2	$2800	1 010 000 0000000
3	$2C00	1 011 000 0000000
4	$3000	1 100 000 0000000
5	$3400	1 101 000 0000000
6	$3800	1 110 000 0000000
7	$3C00	1 111 000 0000000
8	$2080	1 000 001 0000000
9	$2480	1 001 001 0000000
10	$2880	1 010 001 0000000
11	$2C80	1 011 001 0000000
12	$3080	1 100 001 0000000
13	$3480	1 101 001 0000000
14	$3880	1 110 001 0000000
15	$3C80	1 111 001 0000000
16	$2100	1 000 010 0000000
17	$2500	1 001 010 0000000
18	$2900	1 010 010 0000000
19	$2D00	1 011 010 0000000



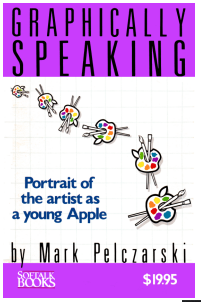
Apple II Hires Addresses: Even more bizarre

Y	Hex	Binary
0	$2000	1 000 000 0000000
1	$2400	1 001 000 0000000
2	$2800	1 010 000 0000000
3	$2C00	1 011 000 0000000
4	$3000	1 100 000 0000000
5	$3400	1 101 000 0000000
6	$3800	1 110 000 0000000
7	$3C00	1 111 000 0000000
8	$2080	1 000 001 0000000
9	$2480	1 001 001 0000000
10	$2880	1 010 001 0000000
11	$2C80	1 011 001 0000000
12	$3080	1 100 001 0000000
13	$3480	1 101 001 0000000
14	$3880	1 110 001 0000000
15	$3C80	1 111 001 0000000
16	$2100	1 000 010 0000000
17	$2500	1 001 010 0000000
18	$2900	1 010 010 0000000
19	$2D00	1 011 010 0000000

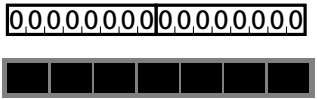
```
HPOSN   STA   HGRY
        STX   HGRX
        STY   HGRX+1
        PHA
        AND   #%11000000
        STA   GBASL
        LSR   A
        LSR   A
        ORA   GBASL
        STA   GBASL
        PLA
        STA   GBASH
        ASL   A
        ASL   A
        ASL   A
        ROL   GBASH
        ASL   A
        ROL   GBASH
        ASL   A
        ROR   GBASL
        LDA   GBASH
        AND   #%00011111
        ORA   HGRPAGE
        STA   GBASH
        TXA
        CPY   #0
        BEQ   HPOSN2
        LDY   #35
        ADC   #4
HPOSN1  INY
HPOSN2  SBC   #7
        BCS   HPOSN1
        STY   HGRHORIZ
        TAX
        LDA   MSKTBL-249,X
        STA   HMASK
        TYA
        LSR   A
        LDA   HGRCOLOR
        STA   HCOLOR1
        BCS   COLORSHIFT
        RTS
```

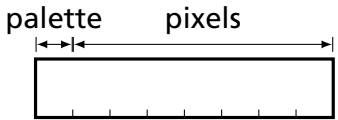
Solution: 384 Byte Address Lookup Tables

```
LKHI hex 2024282c3034383c2024282c3034383c
     hex 2125292d3135393d2125292d3135393d
     hex 22262a2e32363a3e22262a2e32363a3e
     hex 23272b2f33373b3f23272b2f33373b3f
     hex 2024282c3034383c2024282c3034383c
                         ...

LKLO hex 0000000000000000808080808080808080
     hex 0000000000000000808080808080808080
     hex 0000000000000000808080808080808080
     hex 0000000000000000808080808080808080
     hex 2828282828282828a8a8a8a8a8a8a8a8
                         ...
```

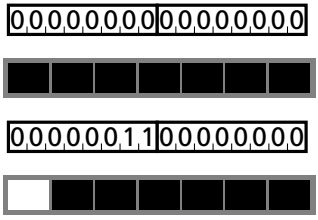


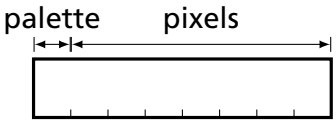
Apple II Hires Colors: 7 pixels/2 bytes



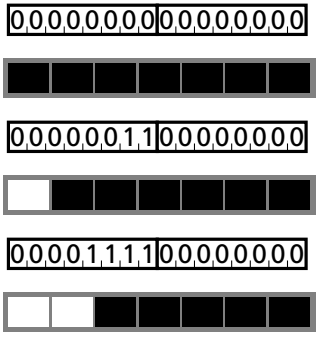


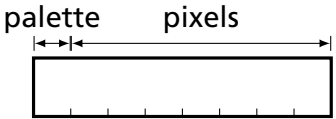
Apple II Hires Colors: 7 pixels/2 bytes



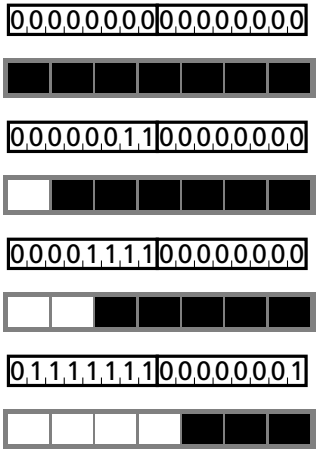


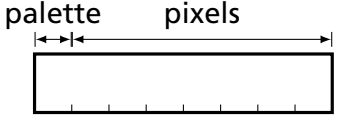
Apple II Hires Colors: 7 pixels/2 bytes



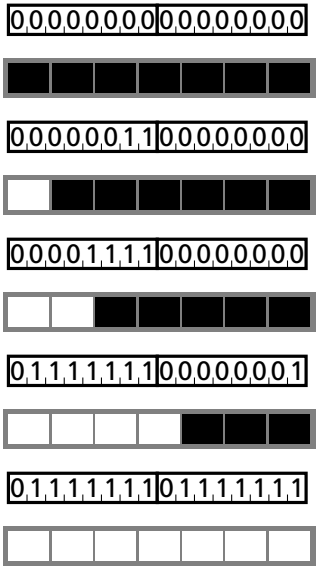


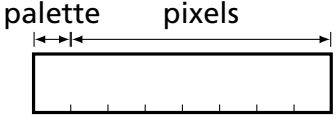
Apple II Hires Colors: 7 pixels/2 bytes



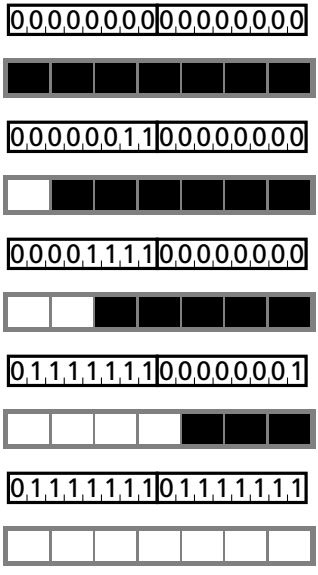


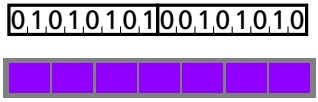
Apple II Hires Colors: 7 pixels/2 bytes

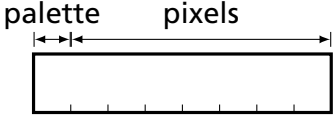




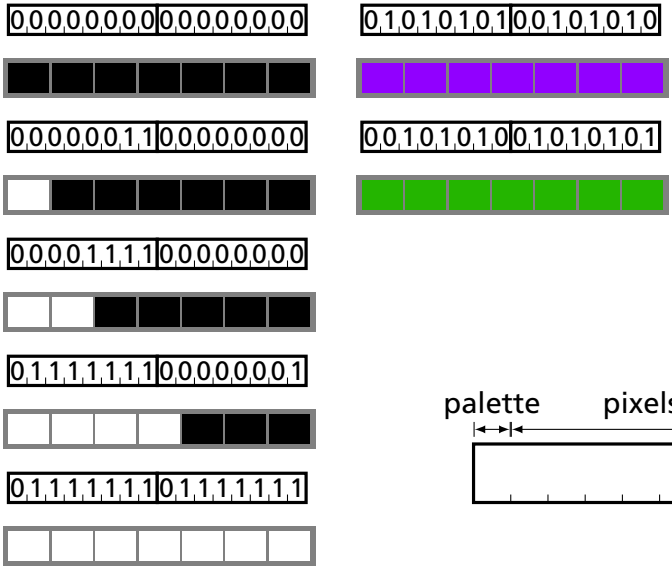
Apple II Hires Colors: 7 pixels/2 bytes



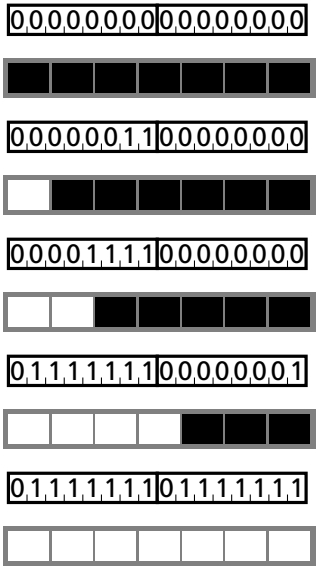


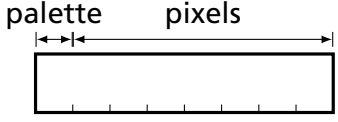


Apple II Hires Colors: 7 pixels/2 bytes

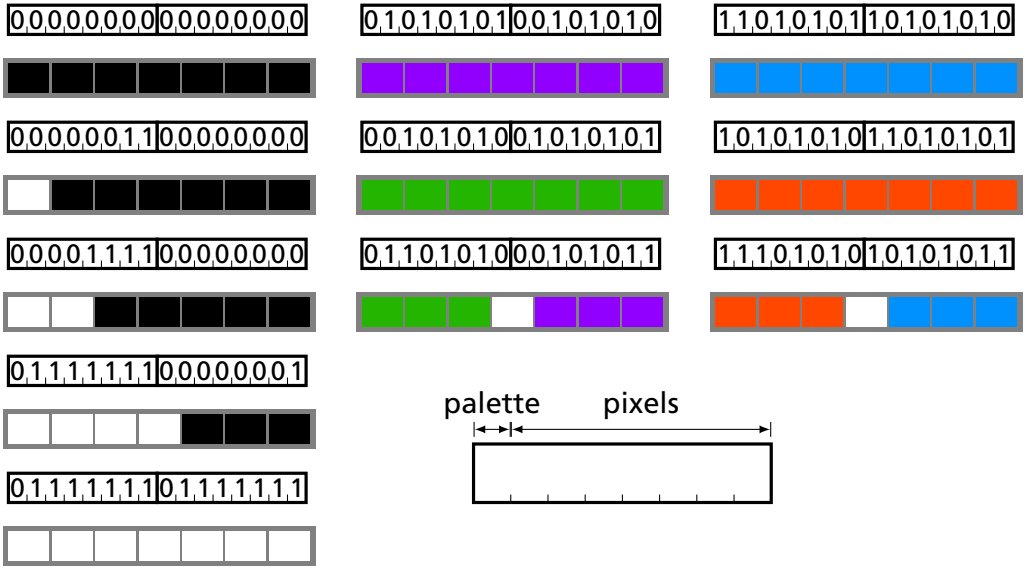


Apple II Hires Colors: 7 pixels/2 bytes





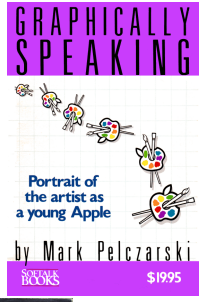
Apple II Hires Colors: 7 pixels/2 bytes



Preshifted Shapes: 7 versions

```
BALL0   db %00111100
        db %01111111
        db %01111111
        db %01111111
        db %01111111
        db %01111111
        db %01111111
        db %00111100

        db %01111000
        db %01111110
        db %01111110
        db %01111110
        db %01111110
        db %01111110
        db %01111110
        db %01111000

        db %01110000
        db %01111100
        db %01111100
        db %01111100
        db %01111100
        db %01111100
            ...
```

```
BALL1   db %00000000
        db %00000001
        db %00000001
        db %00000001
        db %00000001
        db %00000001
        db %00000001
        db %00000000

        db %00000000
        db %00000011
        db %00000011
        db %00000011
        db %00000011
        db %00000011
        db %00000011
        db %00000000

        db %00000001
        db %00000111
        db %00000111
        db %00000111
        db %00000111
        db %00000111
            ...
```



Bouncing Balls: Clear the screen

```
hclear  ldx #>HGR1SCRN  ; $20, also the number of pages to clear
        stx GBASH
        lda #0          ; Clear to black
        sta GBASL
        tay
```

Bouncing Balls: Clear the screen

```
hclear  ldx #>HGR1SCRN  ; $20, also the number of pages to clear
        stx GBASH
        lda #0          ; Clear to black
        sta GBASL
        tay
hclr1   sta (GBASL),y
        iny
        bne hclr1       ; Done with the page?
```

Bouncing Balls: Clear the screen

```
hclear  ldx #>HGR1SCRN  ; $20, also the number of pages to clear
        stx GBASH
        lda #0          ; Clear to black
        sta GBASL
        tay
hclr1   sta (GBASL),y
        iny
        bne hclr1       ; Done with the page?
        inc GBASH
        dex
        bne hclr1       ; Done with all pages?
```

Bouncing Balls: Clear the screen

```
hclear  ldx #>HGR1SCRN  ; $20, also the number of pages to clear
        stx GBASH
        lda #0          ; Clear to black
        sta GBASL
        tay
hclr1   sta (GBASL),y
        iny
        bne hclr1       ; Done with the page?
        inc GBASH
        dex
        bne hclr1       ; Done with all pages?
        bit HIRES       ; Switch to hires mode
        bit TXTCLR
        rts
```

Bouncing Balls: Horizontal line

```
; Draw a horizontal line
; A = color byte to repeat, e.g., $7F
; Y = row (0-191) ($FF on exit)
;
; Uses GBASL, GBASH
```

Bouncing Balls: Horizontal line

```
; Draw a horizontal line
; A = color byte to repeat, e.g., $7F
; Y = row (0-191) ($FF on exit)
;
; Uses GBASL, GBASH
hline   pha
```

Bouncing Balls: Horizontal line

```
; Draw a horizontal line
; A = color byte to repeat, e.g., $7F
; Y = row (0-191) ($FF on exit)
;
; Uses GBASL, GBASH
hline   pha
        lda LKLO,y
        sta GBASL
        lda LKHI,y
        sta GBASH
```

Bouncing Balls: Horizontal line

```
; Draw a horizontal line
; A = color byte to repeat, e.g., $7F
; Y = row (0-191) ($FF on exit)
;
; Uses GBASL, GBASH
hline   pha
        lda LKLO,y
        sta GBASL
        lda LKHI,y
        sta GBASH
        ldy #COLUMNS-1  ; Width of screen in bytes
        pla
```

Bouncing Balls: Horizontal line

```
; Draw a horizontal line
; A = color byte to repeat, e.g., $7F
; Y = row (0-191) ($FF on exit)
;
; Uses GBASL, GBASH
hline   pha
        lda LKLO,y
        sta GBASL
        lda LKHI,y
        sta GBASH
        ldy #COLUMNS-1  ; Width of screen in bytes
        pla
hl1     sta (GBASL),y
        dey
        bpl hl1
        rts
```

Bouncing Balls: Vertical line

```
; Draw a vertical line on all but the topmost and bottommost rows
; A = byte to write in each position
; Y = column
;
; Uses GBASL, GBASH, HCOLOR1
vline   sta HCOLOR1
        ldx #190        ; Start at second-to-last row
vl1     lda LKLO,x      ; Get the row address
        sta GBASL
        lda LKHI,x
        sta GBASH
        lda HCOLOR1
        sta (GBASL),y   ; Write the color byte
        dex             ; Previous row
        bne vl1
        rts
```

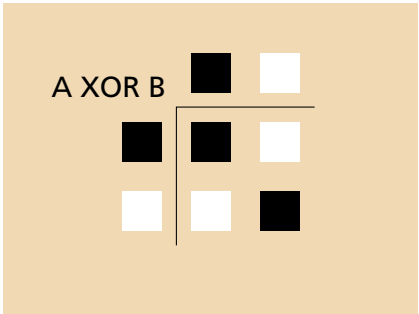
The Exclusive-OR Trick

A XOR B	B=0	B=1
A=0	0	1

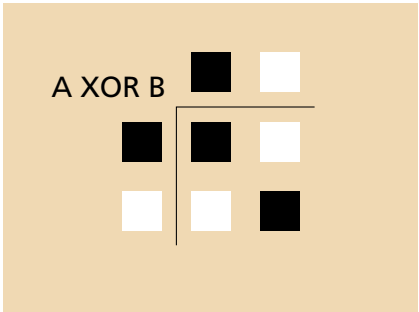
The Exclusive-OR Trick

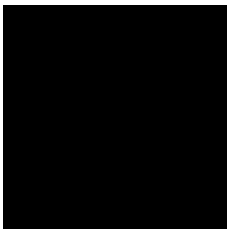
A XOR B	B=0	B=1
A=0	0	1
A=1	1	0

The Exclusive-OR Trick

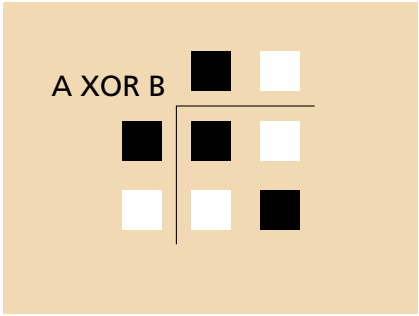


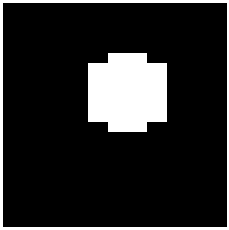
The Exclusive-OR Trick



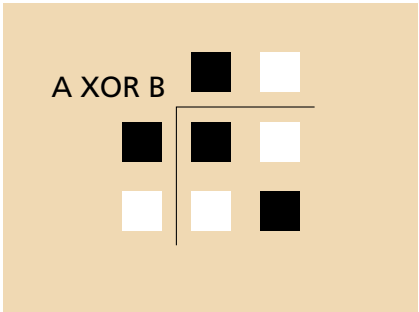


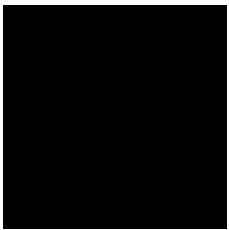
The Exclusive-OR Trick



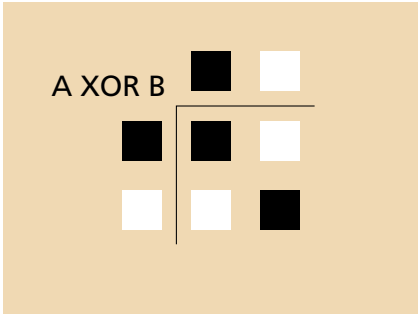


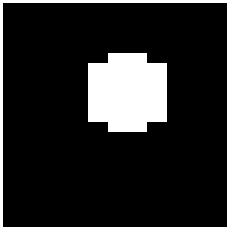
The Exclusive-OR Trick



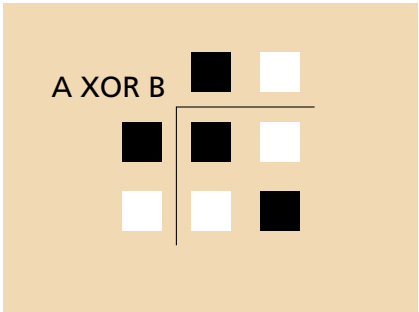


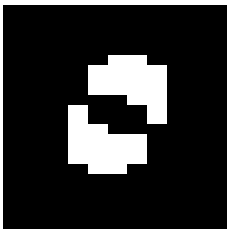
The Exclusive-OR Trick



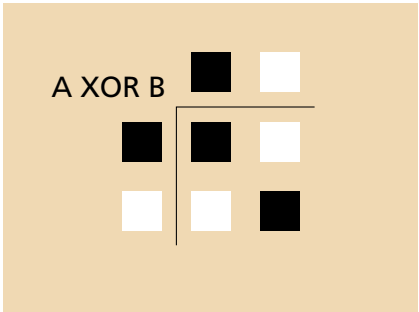


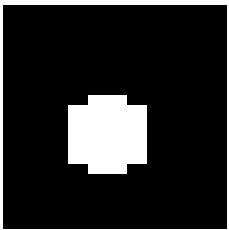
The Exclusive-OR Trick



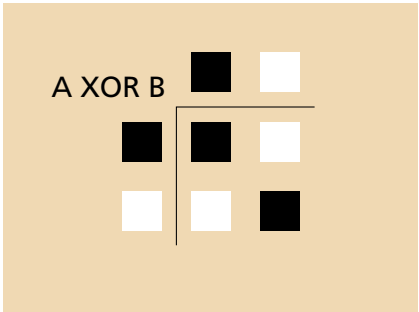


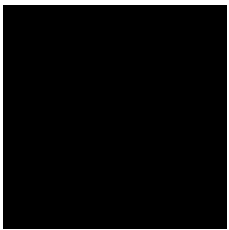
The Exclusive-OR Trick





The Exclusive-OR Trick





Draw the Ball: XOR Two Bytes

```
lda (GBASL),y
eor BALL0,x
sta (GBASL),y
iny
lda (GBASL),y
eor BALL1,x
sta (GBASL),y
```

Draw the Ball: XOR Two Bytes

(GBASL, GBASH) = row address
Y = byte offset into the row
X = index into sprite tables

```
        lda (GBASL),y
        eor BALL0,x
        sta (GBASL),y
        iny
        lda (GBASL),y
        eor BALL1,x
        sta (GBASL),y
```

Draw the Ball: XOR Two Bytes

(GBASL, GBASH) = row address
Y = byte offset into the row
X = index into sprite tables

```
        ldy HGRX
        lda (GBASL),y
        eor BALL0,x
        sta (GBASL),y
        iny
        lda (GBASL),y
        eor BALL1,x
        sta (GBASL),y
```

Bouncing Balls: Draw a ball

```
xsplot  ldy HGRY        ; Get the row address
        lda LKLO,y
        sta GBASL
        lda LKHI,y
        sta GBASH
        iny
        sty HGRY




        inx
        txa
        and #7
        bne xsplot      ; Stop at a multiple of 8 bytes
        rts
```

Bouncing Balls: Horizontal Position

Byte column (0–39) = $c_5\, c_4\, c_3\, c_2\, c_1\, c_0$

Bit/shift number (0.0–6.875) = $s_2\, s_1\, s_0 \cdot s_{-1}\, s_{-2}\, s_{-3}$

Bouncing Balls: Horizontal Movement 1

```
        lda BALLXL,x
        clc
        adc BALLDX,x
        bpl nounder

        adc #56         ; Correct for underflow; carry is clear
        sta BALLXL,x
        dec BALLXH,x
        bne xdone       ; Hit the left wall?
        beq bouncex     ; Yes: bounce
```

Bouncing Balls: Horizontal Movement 2

```
nounder sta BALLXL,x
        sec
        sbc #56
        bcc xdone       ; No overflow?

        sta BALLXL,x
        inc BALLXH,x
        ldy BALLXH,x
        cpy #COLUMNS-2  ; Hit the right wall?
        bne xdone

bouncex sec
        lda #0
        sbc BALLDX,x
        sta BALLDX,x
xdone
```

Bouncing Balls: Vertical Fall

```
        inc BALLDYH,x   ; Apply gravity

        clc             ; Update Y
        lda BALLYL,x
        adc BALLDYL,x
        sta BALLYL,x
        lda BALLYH,x
        adc BALLDYH,x
        sta BALLYH,x

        cmp #>BOTTOM    ; Did we hit the bottom?
        bcc nobounce
```

Bouncing Balls: Vertical Bounce

```
        sec             ; We bounced: subtract Y from 2 * BOTTOM
        lda #<BOTTOM2
        sbc BALLYL,x
        sta BALLYL,x
        lda #>BOTTOM2
        sbc BALLYH,x
        sta BALLYH,x

        sec             ; and negate the vertical velocity
        lda #0
        sbc BALLDYL,x
        sta BALLDYL,x
        lda #0
        sbc BALLDYH,x
        sta BALLDYH,x
nobounce
```