



The Nintendo Entertainment System

CS433

Processor Presentation Series
Prof. Luddy Harrison



Note on this presentation series

- These slide presentations were prepared by students of CS433 at the University of Illinois at Urbana-Champaign
- All the drawings and figures in these slides were drawn by the students. Some drawings are based on figures in the manufacturer's documentation for the processor, but none are electronic copies of such drawings
- You are free to use these slides provided that you leave the credits and copyright notices intact



Agenda

- Historical Perspective
- Background: The 6502 Processor
- The NES 2A03
- The NES PPU (Picture Processing Unit)



Ancient History - 1975

- Several designers of Motorola 6800 left the company
- Joined MOS Technology and produced the MOS 6501, a simpler, faster processor pin-compatible with the 6800
- Motorola sued, so MOS releases the 6502, same chip but not pin-compatible
- 6502 goes on sale in 1975 for \$25 – at a time when Intel 8080, Motorola 6800 cost \$179
 - MOS improved yield rate (working chips/total chips), allowing cheaper costs



Slightly Later History - 1977

- Atari 2600 (released in 1977 as “Atari VCS”) used a modified MOS 6502 processor
 - MOS 6507 vs. MOS 6502
 - Fewer address pins (13 vs. 16 pins, so can address 8K of memory – OK, since Atari cartridge only allows 4K addressing)
 - Unable to service external interrupts



More Recent History - 1983

- NES (released in Japan in 1983 as “Famicom”, in US in 1985) used a modified 6502 processor
 - Ricoh RP 2A03 G vs. MOS 6502
 - Differences discussed later



Huh?

- NES built on “old” technology – why?
 - Keep the price low for competition
 - Offload graphics to PPU for wow factor
 - Effective strategy: Nintendo most profitable company in Japan in 1990



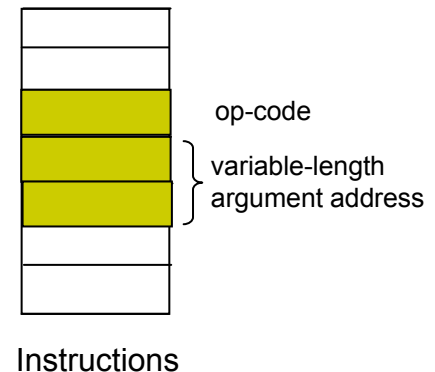
NES Foundation: The MOS 6502

- 6502 hugely influential
 - Forces processor price reduction
 - Drives Atari 2600
 - Drives Commodore 64
 - Drives Apple II
 - Drives Nintendo Entertainment System
 - Simplicity and efficiency inspire ARM designers
- So understanding 6502 gives you an entry to all of those systems



Foundation: 6502 Instruction Set

- Variable-length instructions
 - Read one byte at a time, addressed by PC
 - First instruction byte identifies operation, addressing mode
 - Bytes 2-n are the argument addresses (including immediate addressing)
- 56 total instructions
 - 12 jump/branch/return, 16 Arithmetic/Logical, 11 status flag interaction, 12 load/store/move, 4 stack manipulation, NOP
 - About half of the possible encodings are unused
 - Several undocumented features have been added to various versions of the 6502 to take advantage of unused encodings





Foundation: 6502 Instruction Set (cont.)

- Three types of instructions
 - Group One: General purpose instructions
 - Load, Add, Store, etc.
 - Eight Addressing Modes (to be discussed...)
 - Immediate, Zero Page, Zero Page Indexed by X, Absolute, Absolute Indexed by X, Absolute Indexed by Y, Indexed Indirect, Indirect Indexed
 - Group Two: Read, Modify, Write Instructions
 - Shift Right, Rotate Left, Increment, Decrement
 - Five Addressing Modes
 - Zero Page, Zero Page Indexed by X, Absolute, Absolute Indexed by X, Accumulator, Indexed Indirect
 - Group Three: Remaining instructions not as easily classified
 - Various Addressing Modes



Foundation: 6502 Instruction Set: Branching Rules

- Branch offset limit is -127 to 128
- If further branches are required, invert condition and branch to local unconditional jump
 - 8 branch instructions are based on whether C, V, Z, N flags are set or reset, so all inverted conditions available
- All branches are assumed not taken
 - One-cycle penalty if taken
 - Two-cycle penalty if taken and offset crosses memory page boundary



Foundation: 6502 Instruction Set: Stack Operations

- Push, Pull both interact with memory and change the stack pointer
- Jumps to Subroutines (essentially) push the next instruction address to the stack
- Returns from Interrupt/Subroutines pop stack and restore PC.
- Return from Interrupt also restores status register P



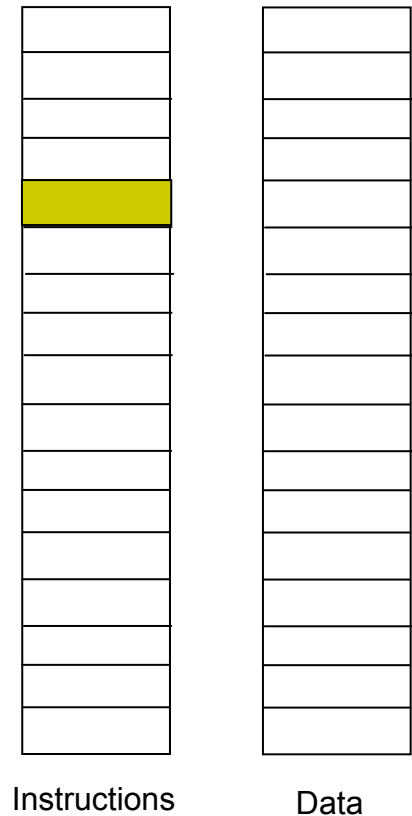
Foundation: 6502 Instruction Set: Problems

- Some Limitations of the Instruction Set
 - No multiply, divide
 - No Floating Point – Only signed binary or BCD (binary-coded decimal) data types
 - All arguments are one byte. For higher precision, programmer must manually combine separate one-byte operations.

Foundation: 6502 Addressing Modes: Implied Addressing

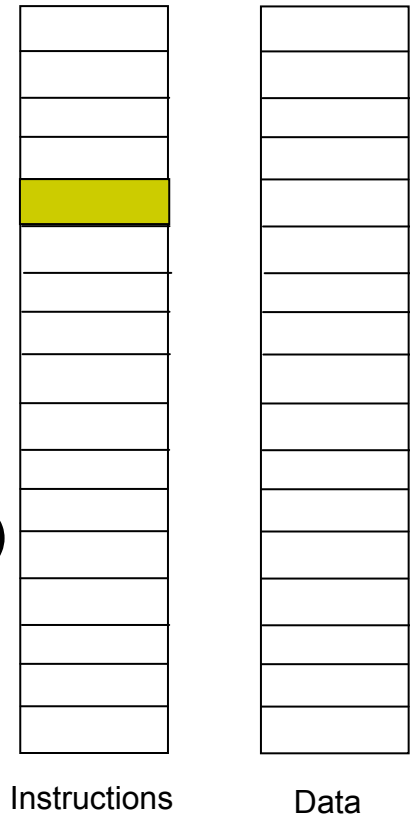
- One-byte instructions with no arguments
- Examples:
 - CLV (clear overflow flag)
 - TAY (Transfer accumulator to index Y)

(These are going to get hairy. Be prepared)



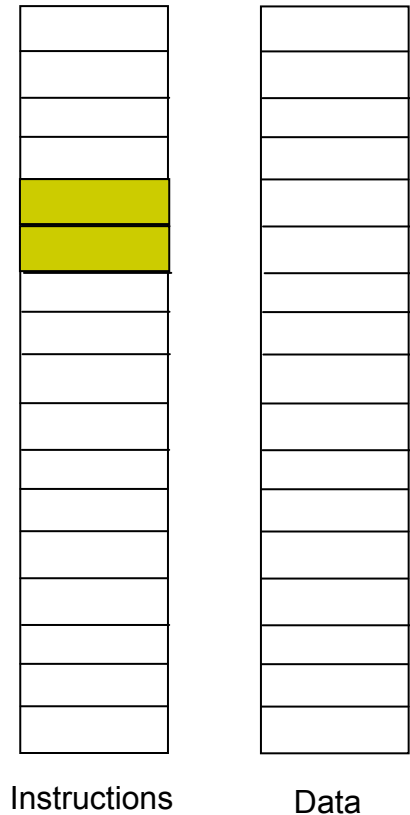
Foundation: 6502 Addressing Modes: Accumulator Addressing

- One-byte instructions, Similar to implied addressing
- For operations that could take an in-memory argument, but in this case are operating on the current accumulator value
- Only shift and rotate instructions use this mode
- **Examples** ('\$' prefix indicates hexadecimal format):
 - ASL A (shift the accumulator left one bit) vs.
 - ASL \$nnnn (shift the value in \$nnnn left one bit)



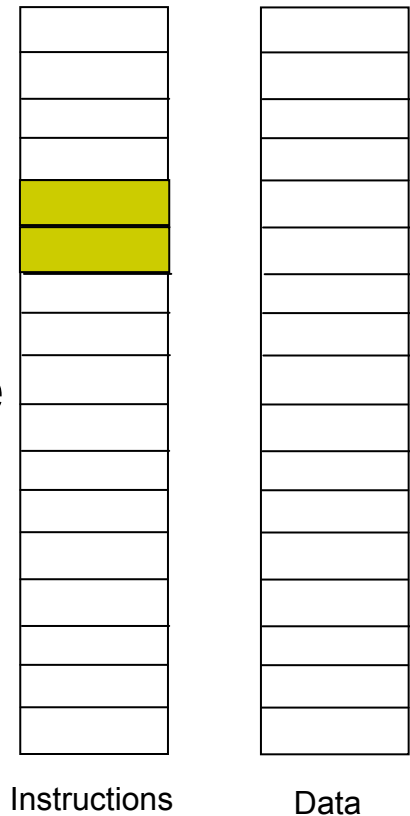
Foundation: 6502 Addressing Modes: Relative Addressing

- Two-byte instructions used by all branch instructions:
 - | op-code | offset |
- Examples:
 - BCS #50 (branch forward 50 if carry flag set)
 - BCC #-50 (branch backward 50 if carry flag clear)



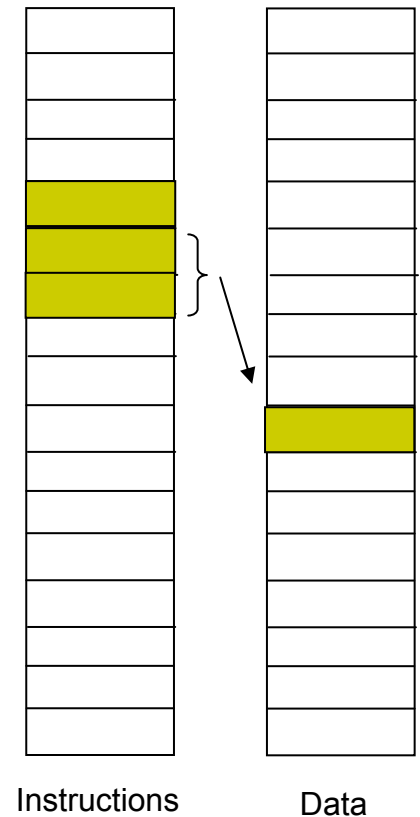
Foundation: 6502 Addressing Modes: Immediate Addressing

- Two-byte instructions with argument in second byte
 - | op-code | argument |
- Examples
 - ADC #29 (Add 29 to the value in the accumulator)
 - LDA #65 (Place the value 65 in the accumulator)



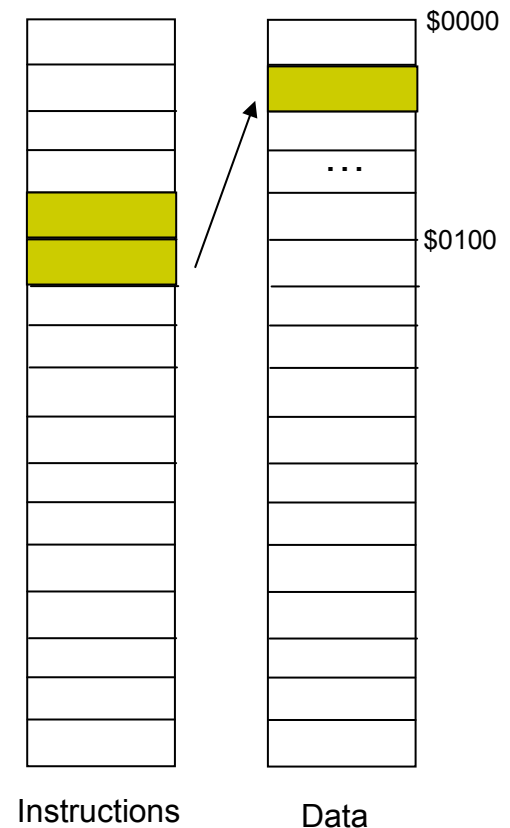
Foundation: 6502 Addressing Modes: Absolute Addressing

- Three-byte instructions
 - | op-code | low address byte | high address byte |
- Note: 16-bit addresses are stored in little-endian mode
- Read argument from given memory location
- Four read cycles:
 - 1) read op-code 2-3) read address 4) read argument
- Examples:
 - STA \$nnnn (store the accumulator value at \$nnnn)
 - ADC \$nnnn (add the value at \$nnnn to accumulator)



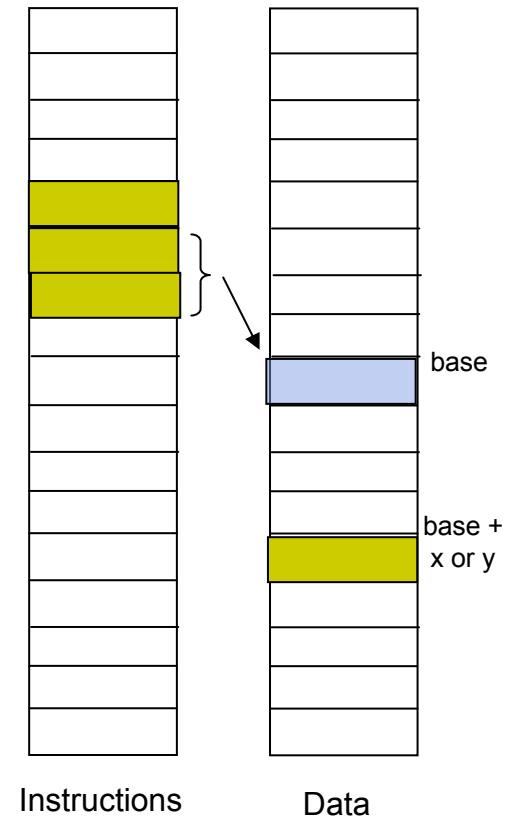
Foundation: 6502 Addressing Modes: Zero Page Addressing

- Two-byte instructions:
 - | op-code | low address byte |
- Read argument from given offset to memory page 0 (address \$00nn)
- Three read cycles:
 - 1) read op-code 2) read address 3) read argument
- The 6502 rewards smart memory management:
 - Zero-page addressed instructions execute 1 1/3 times faster than absolute addressed instructions
 - You don't have to spend a cycle reading the high address bits
- Examples:
 - STA \$nn (store the accumulator value at \$00nn)
 - ADC \$nn (add the value at \$00nn to accumulator)



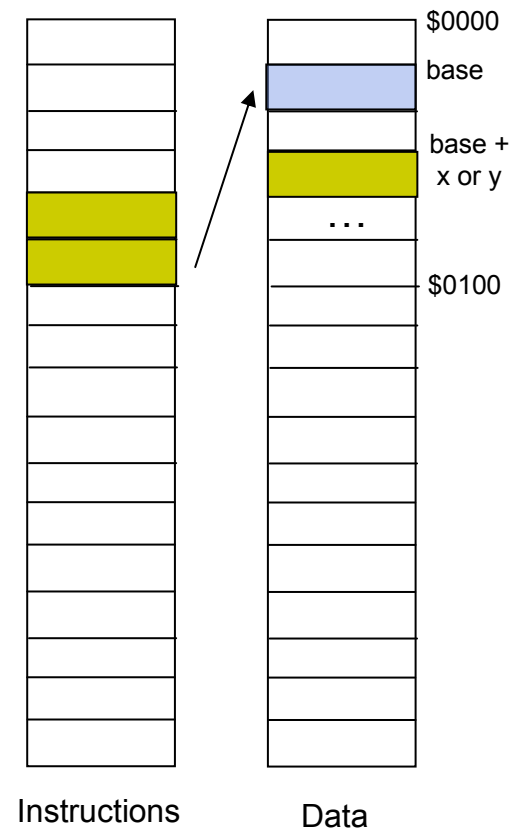
Foundation: 6502 Addressing Modes: Absolute Indexed by X / Y Addressing

- Three-byte instructions
 - | op-code | low address byte | high address byte |
- Read argument from given memory location + value of X or Y index register
 - This method allows, for example, arrays to be processed in a loop by incrementing the index register after each iteration
- Four read cycles:
 - 1) read op-code 2-3) read address 4) read argument
- Examples:
 - STA \$nnnn, X (store the accumulator value at (\$nnnn + X))
 - ADC \$nnnn, Y (add the value at (\$nnnn + Y) to accumulator)



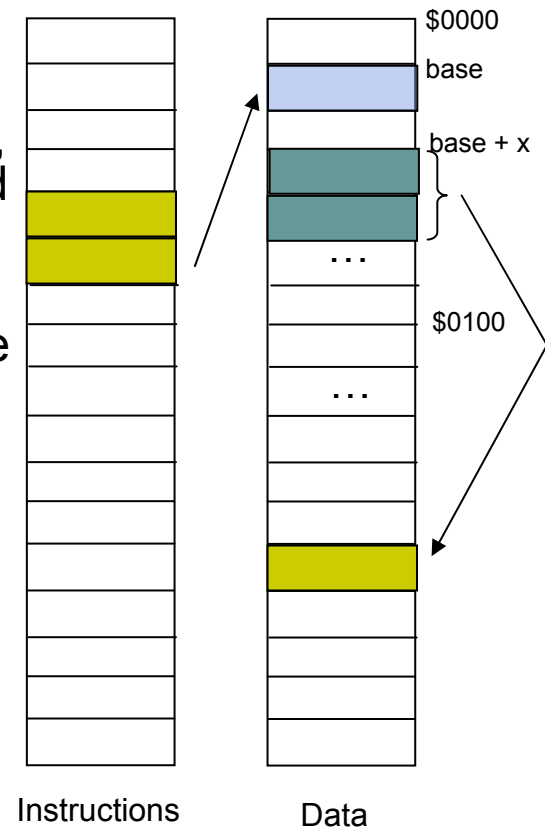
Foundation: 6502 Addressing Modes: Zero Page Indexed by X / Y Addressing

- Two-byte instructions:
 - | op-code | low address byte |
- Read argument from given offset + X/Y index to memory page 0 (address \$00nn)
- Three read cycles:
 - 1) read op-code 2) read address 3) read argument
- As in Zero Page addressing, faster than absolute indexed counterparts
- Zero Page Indexed by Y only available for LDX, STX
- Examples:
 - STX \$nn, Y (store X value at (\$00nn + Y))
 - ADC \$nn, X (add value at (\$00nn + X) to accumulator)



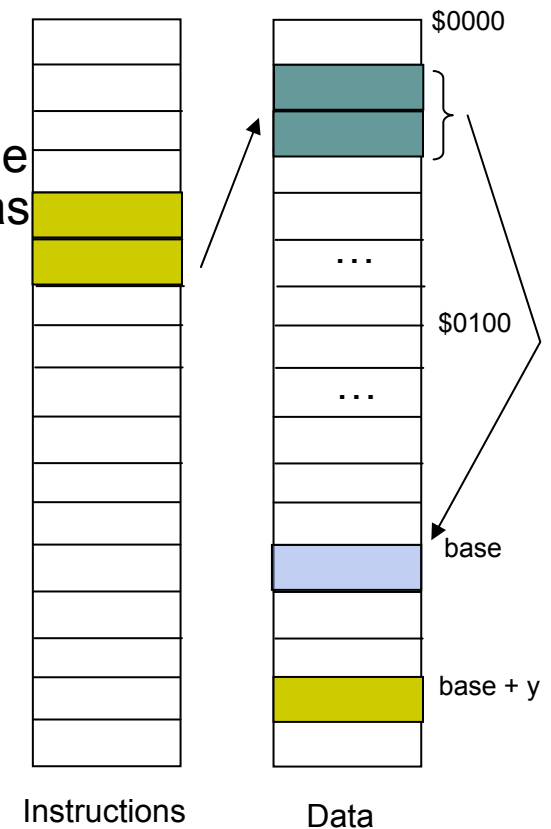
Foundation: 6502 Addressing Modes: Indexed Indirect Addressing

- Two-byte instructions:
 - | op-code | low address byte |
- Read an argument's address from a given location, offset by the X index register into page 0, then read argument
- Six read cycles:
 - 1) read op-code 2) read offset 3) stall (compute offset + X) 4-5) read address 6) read argument
 - Slow – always takes 6 cycles, but can save on code size
- Example:
 - ADC (\$nn, X)
 - 1). Read the two-byte address Z from location $\$00nn + X$, $\$00nn + X + 1$
 - 2). Add the value at address Z to the accumulator



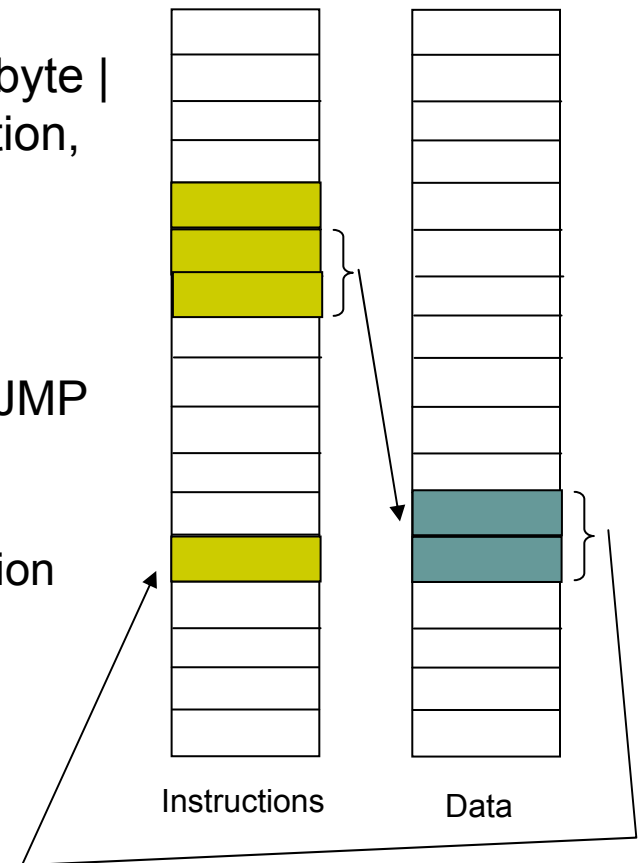
Foundation: 6502 Addressing Modes: Indirect Indexed Addressing

- Two-byte instructions:
 - | op-code | low address byte |
- Read an base address from a given offset into page 0, then read argument by adding Y index register as offset to base
- Five read cycles:
 - 1) read op-code 2) read offset 3-4) read base address 5) read argument
- Example:
 - ADC (\$nn), Y
 - 1). Read two-byte address Z from location \$00nn, \$00nn + 1
 - 2). Add the value at address Z+Y to the accumulator



Foundation: 6502 Addressing Modes: Absolute Indirect Addressing

- Three-byte instructions:
 - | op-code | low address byte | high address byte |
- Read an argument's address from a given location, then read the argument
- Four read cycles:
 - 1) read op-code 2-3) read address 4) read argument
- Absolute Indirect Addressing only available for JMP
- Example:
 - JMP (\$nnnn)
 - 1). Read the two-byte address X from location \$nnnn, \$nnnn+1
 - 2). Jump to address X





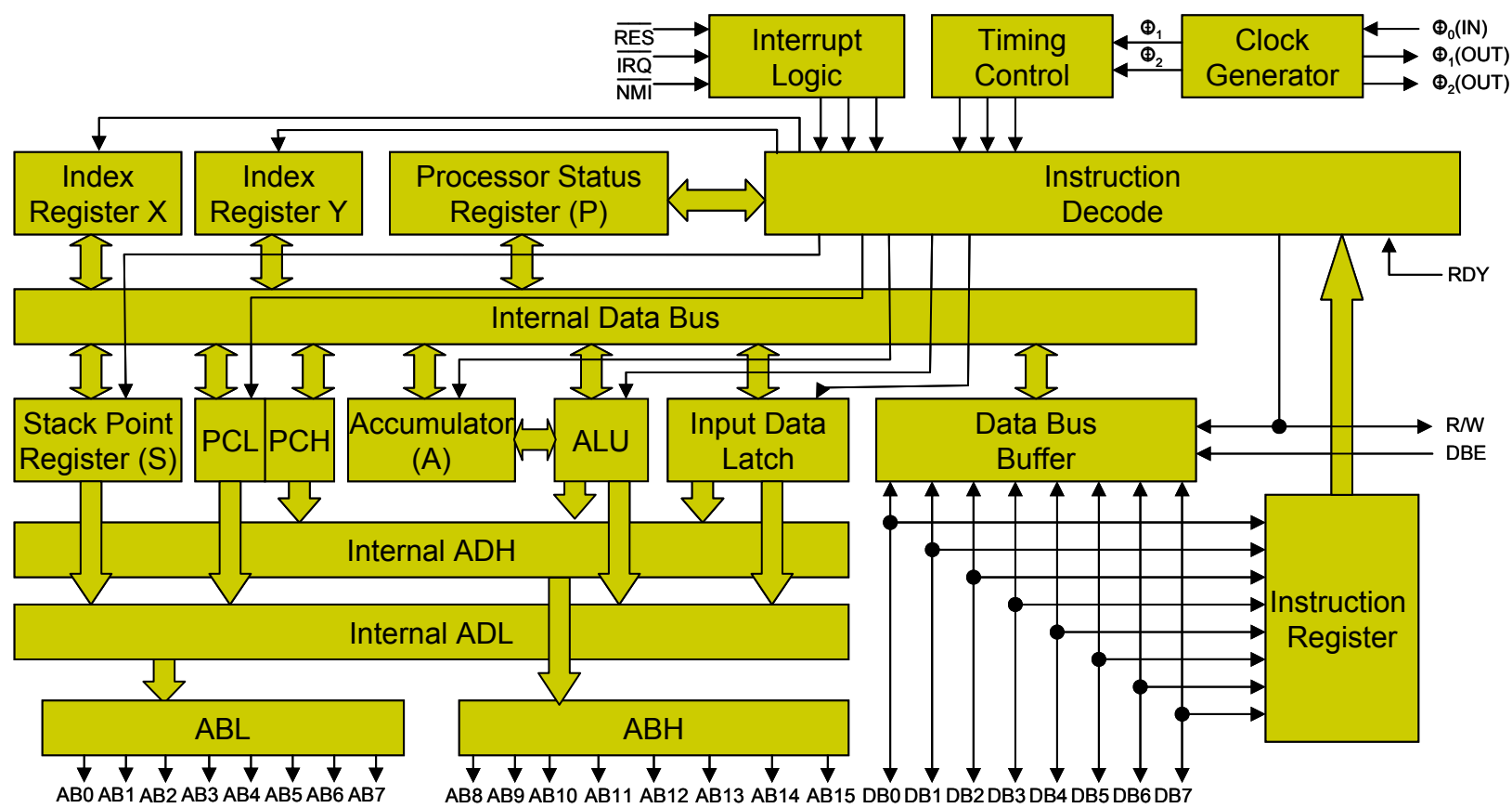
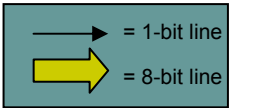
Foundation: 6502 Addressing Modes: Summary

- 13 addressing modes, all told
- Some pretty complex, some pretty simple
- You get lots of flexibility to manage your memory the way you want, trade space for execution speed

Breathe deeply. No more addressing modes, I promise.



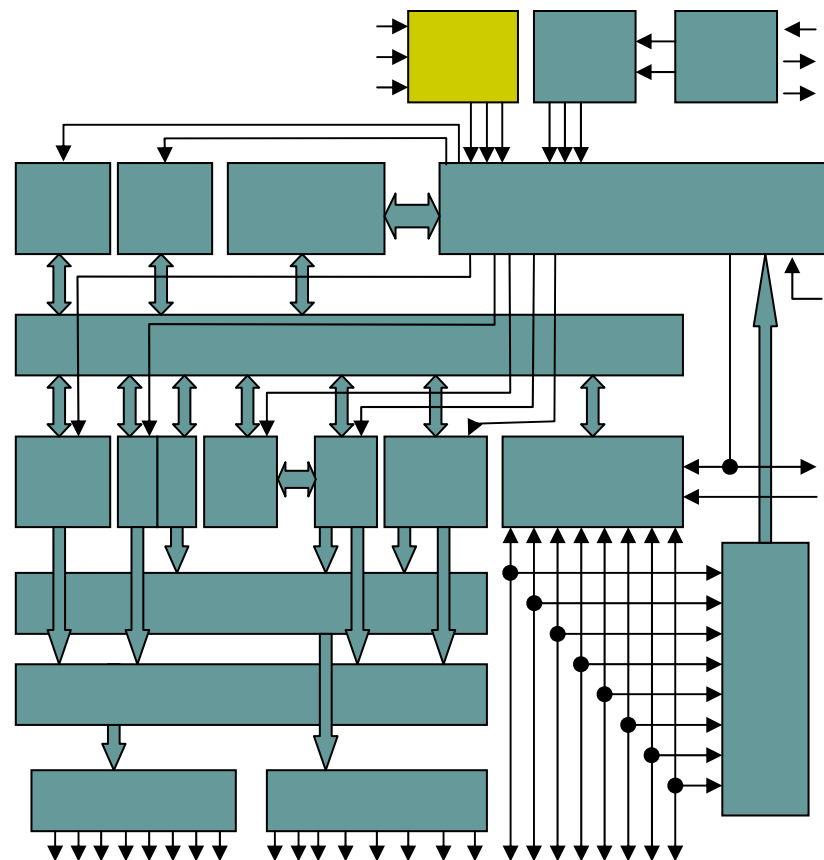
Foundation: 6502 Datapath



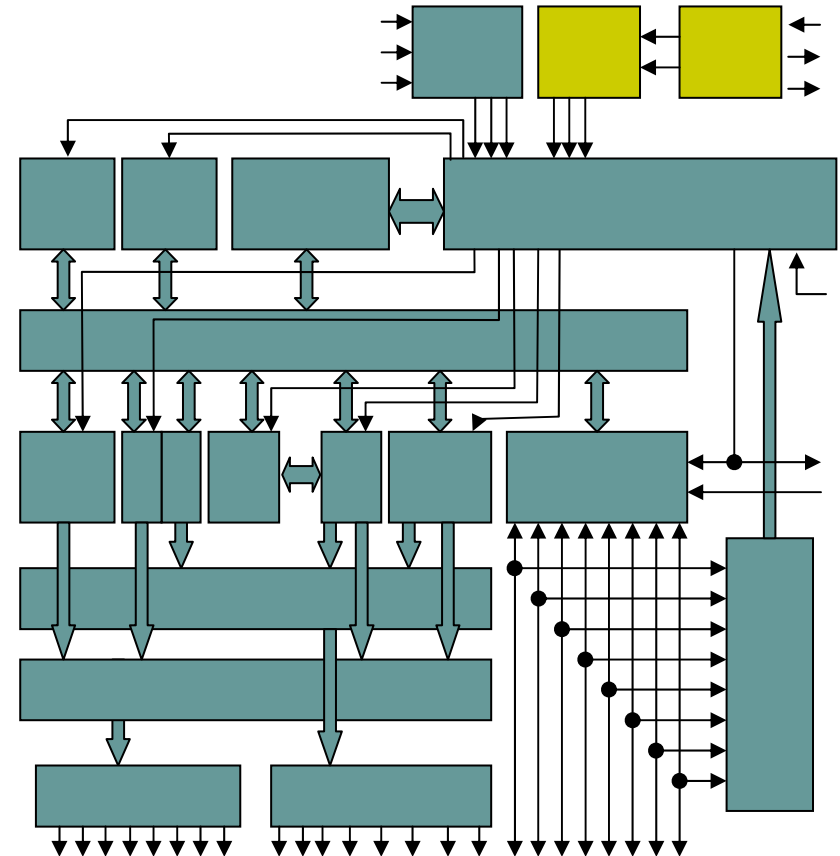
Foundation: 6502 Datapath

Interrupt Logic

- Three inputs
 - IRQ – regular interrupt can be masked via processor status register
 - NMI – non-maskable
 - Reset – used during startup
- Signals are negated
 - Processor gets next instruction address from interrupt vector at top of memory on low signal
- Different devices may cause simultaneous interrupt
 - Interrupt handlers should poll each device



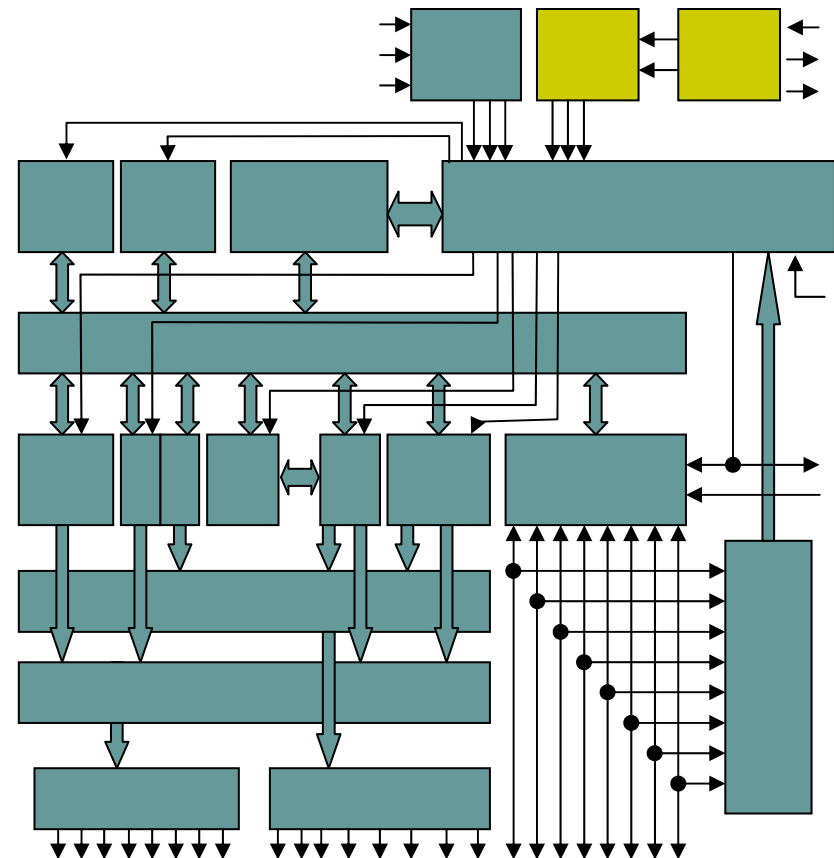
- Two phases in each clock cycle
 - Phase 1: Execution
 - ALU operations complete
 - Next address placed on address bus
 - Phase 2: Memory
 - Addressed location transferred via data bus
 - Could be next instruction read, or memory read/write
- Most instructions take multiple cycles to execute



Foundation: 6502 Datapath

Clock Generator and Timing Control (cont.)

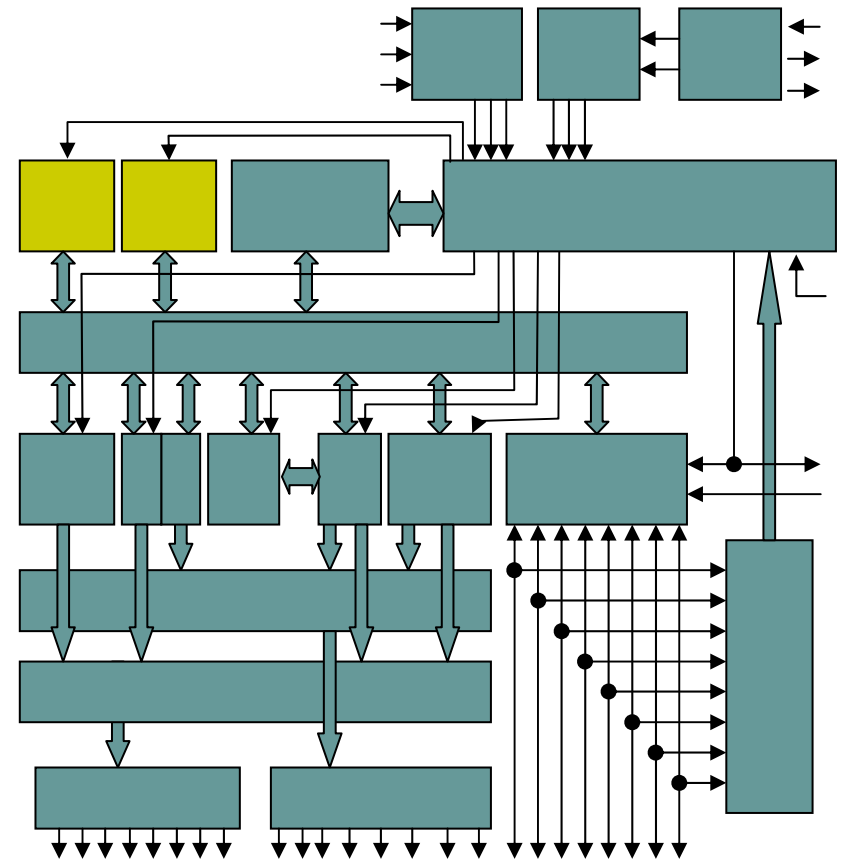
- Memory has to be close, because a byte is read or written each cycle
- Most operations take one argument from memory, one from accumulator
- Clock generator translates a single incoming clock into two phase signals
- Send both phase signals to off-chip devices



Foundation: 6502 Datapath

X and Y Index Registers

- Used chiefly for addressing
- Contents can be moved to and from accumulator
- But no Index Register / Accumulator ALU ops



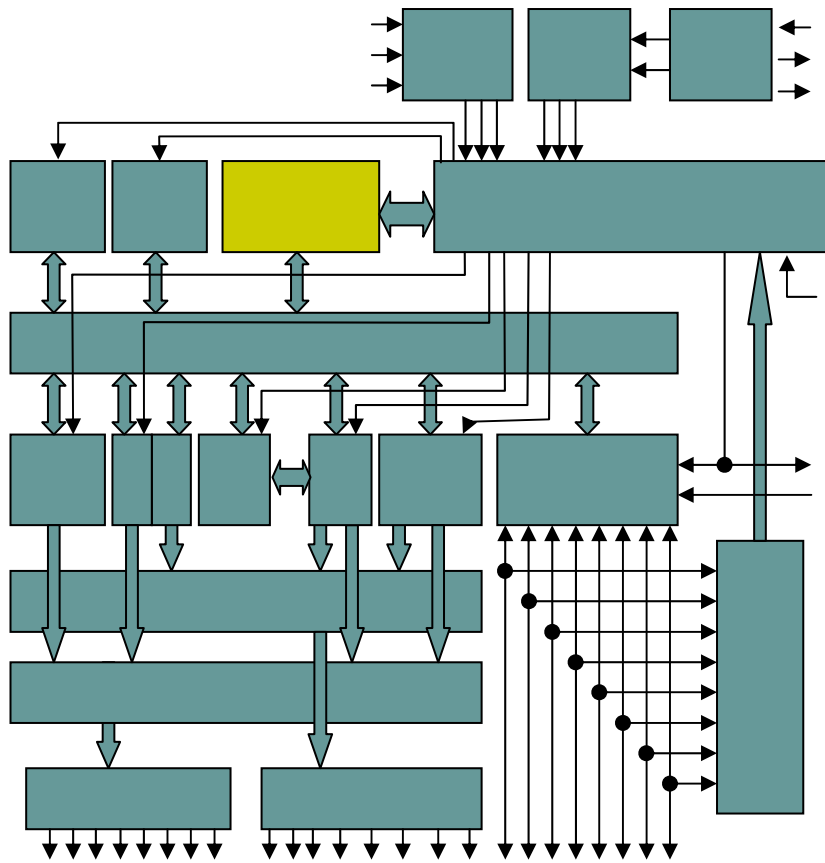


Foundation: 6502 Datapath

Processor Status Register (P)



- Set automatically during execution
- Special instructions to set/reset flags
- Some Familiar Bits
 - N: Negative
 - V: Overflow
 - Z: Zero
 - C: Carry
 - All of these can be tested for branching

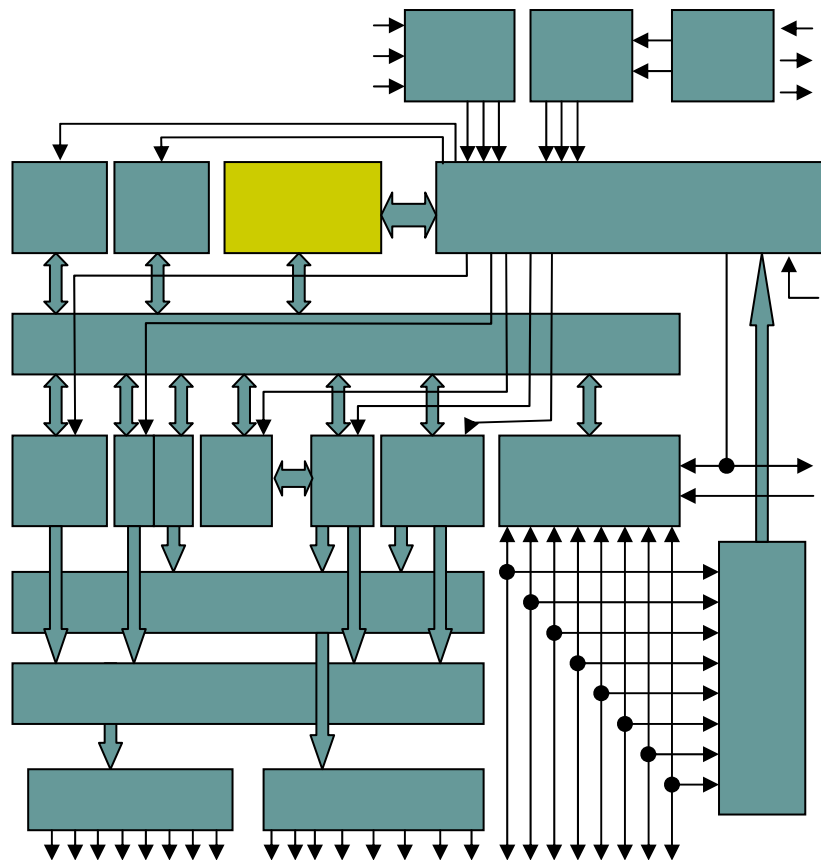


Foundation: 6502 Datapath

Processor Status Register (P) (cont.)



- Some Unfamiliar Bits
 - B: Break Command
 - Set on BRK instruction
 - Distinguishes hardware/software interrupts
 - D: Decimal Mode
 - Interpret arguments in BCD (binary-coded decimal)
 - I: IRQ Disable
 - Suppress interrupts
- Final bit unused

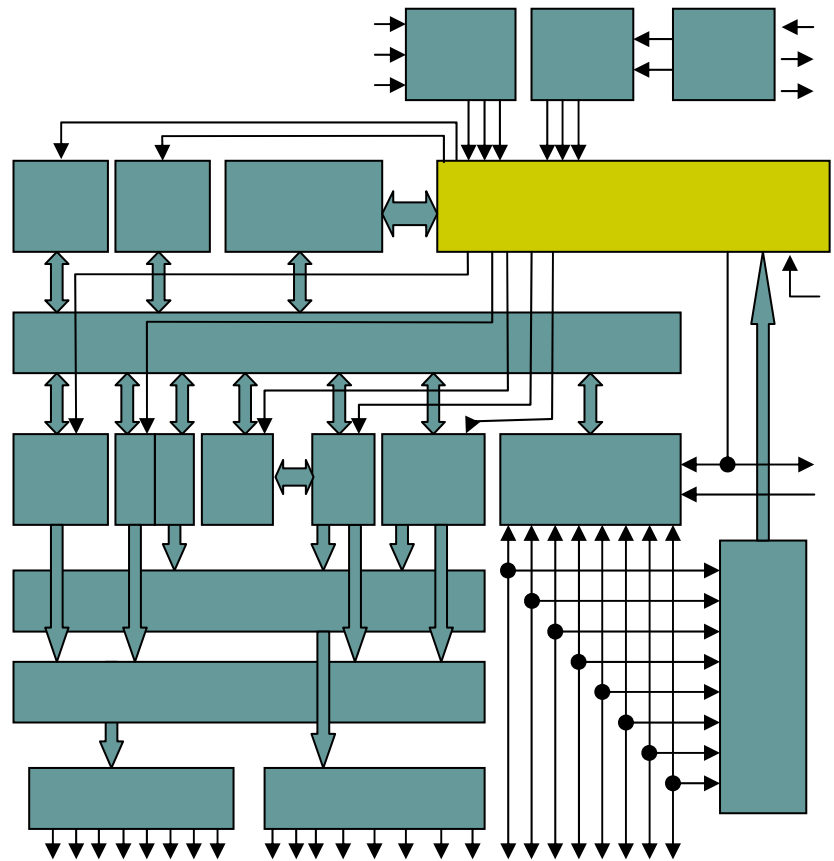




Foundation: 6502 Datapath

Instruction Decode

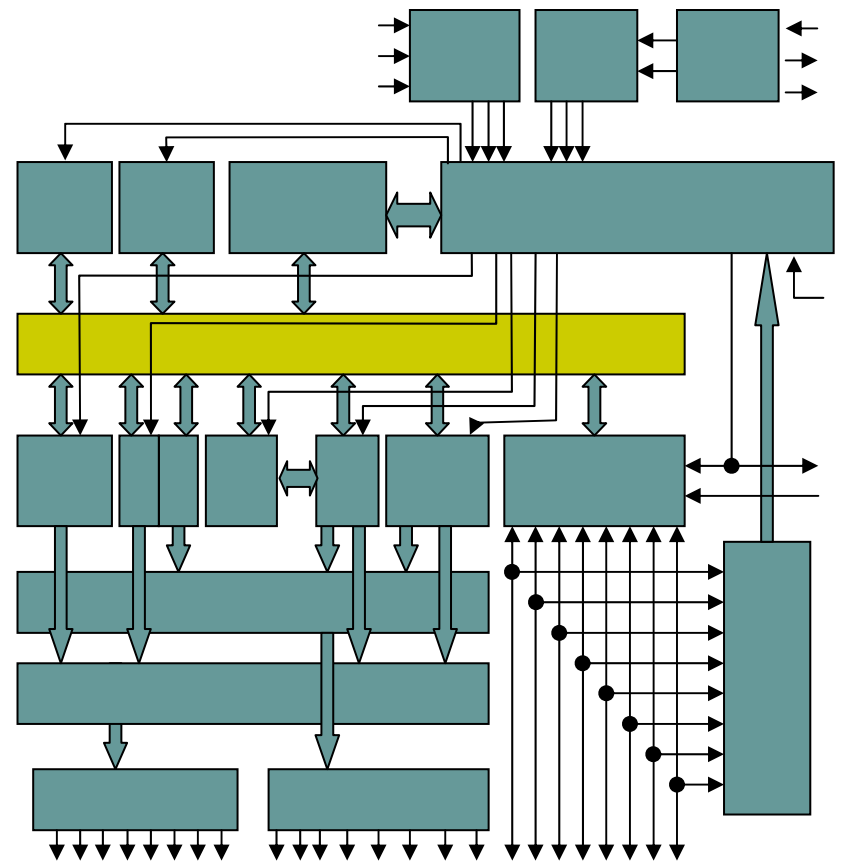
- Interprets variable-length Instructions
- Sends control signals to other components
- De-assert RDY Signal to
 - force single-step execution
 - wait for slow memory devices



Foundation: 6502 Datapath

Internal Data Bus

- 8 bytes wide – enough to carry everything this machine needs at any given moment
- Chief communication device between all internal components

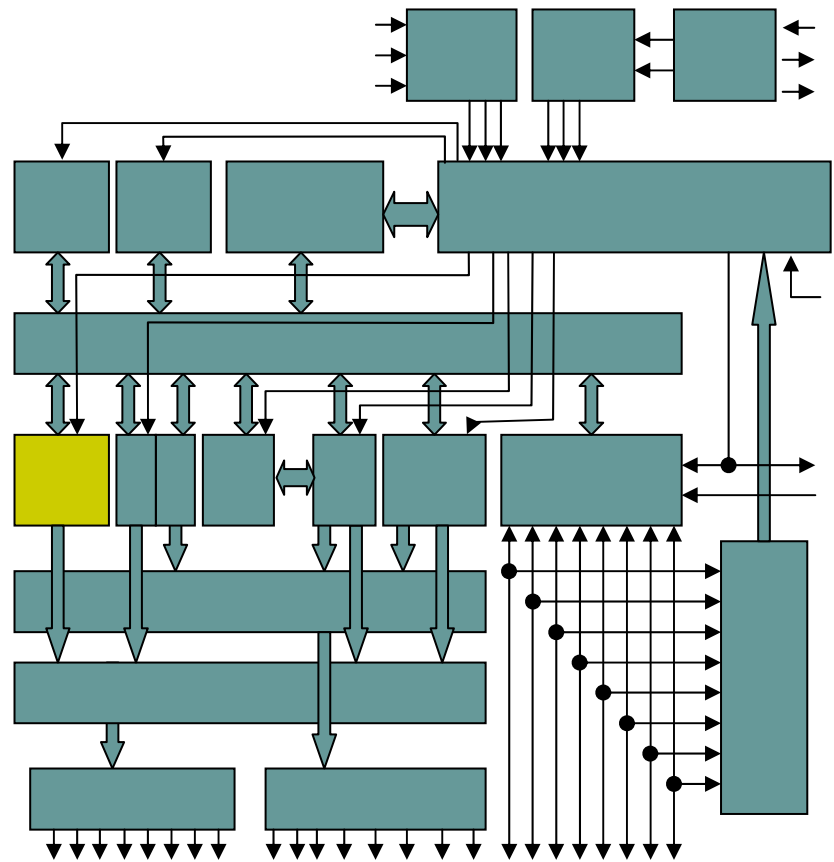




Foundation: 6502 Datapath

Stack Point Register

- Stack is one method of communicating subroutine arguments
- Stack stores machine state during interrupt handling
- Stack is always located in address space 0100-01FF
 - So only one byte needed to address stack location

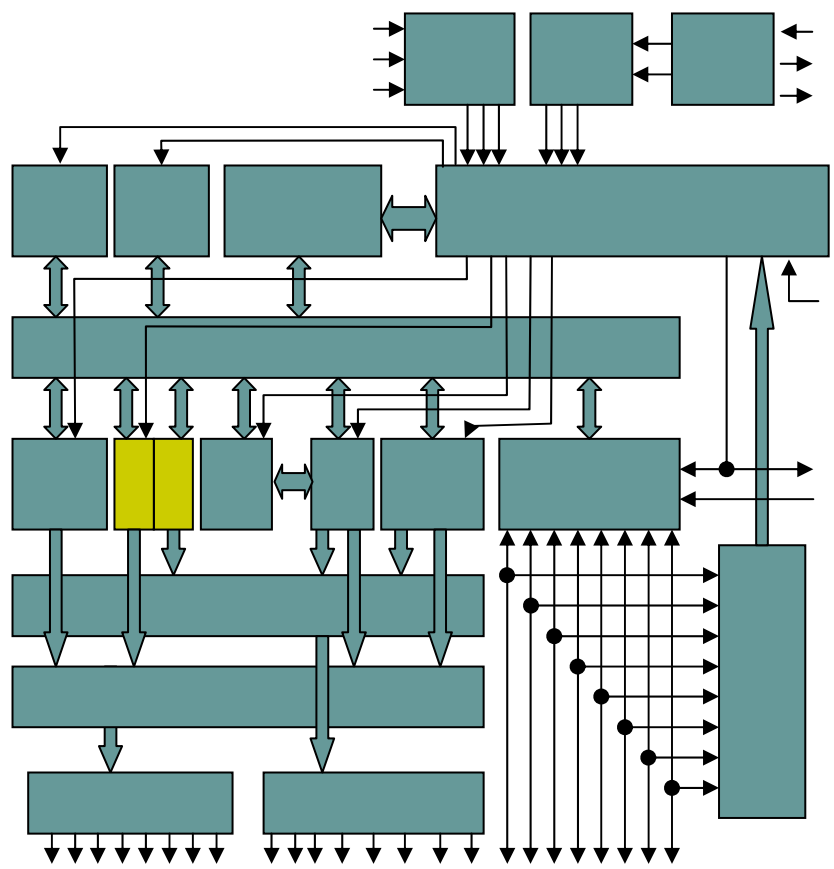




Foundation: 6502 Datapath

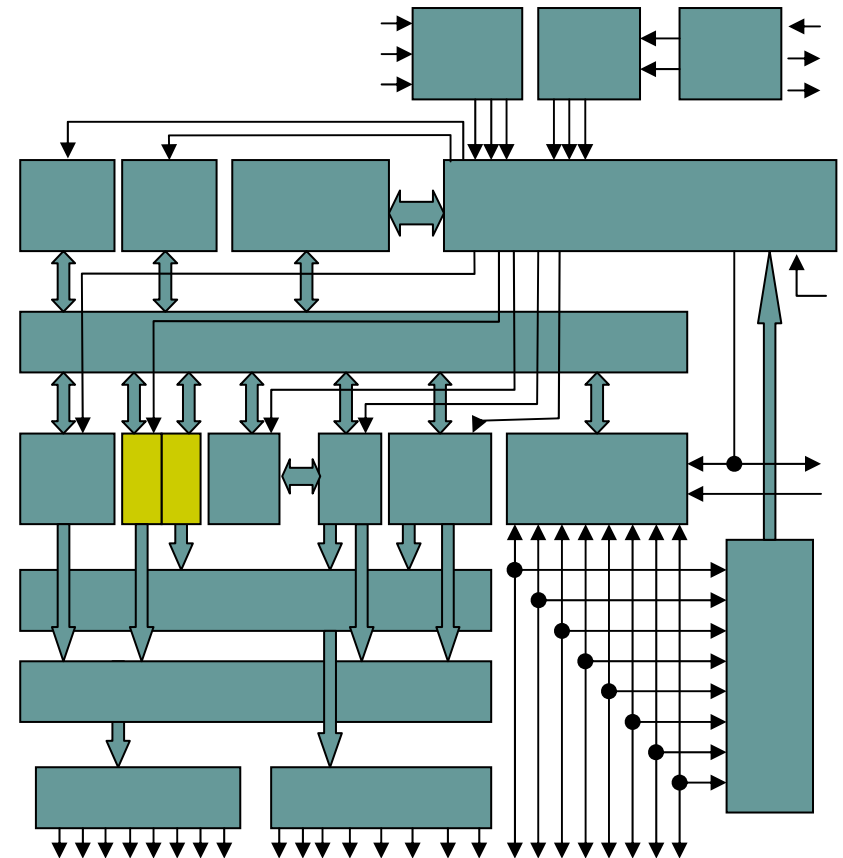
Program Counter

- Registers are 8 bits, we need 16 bits to address memory
- Separate PCH and PCL registers hold the full 16-bit PC
- Changed in several ways:
 - PCL incremented after each instruction, PCH incremented when PCL overflows



Foundation: 6502 Datapath Program Counter (cont.)

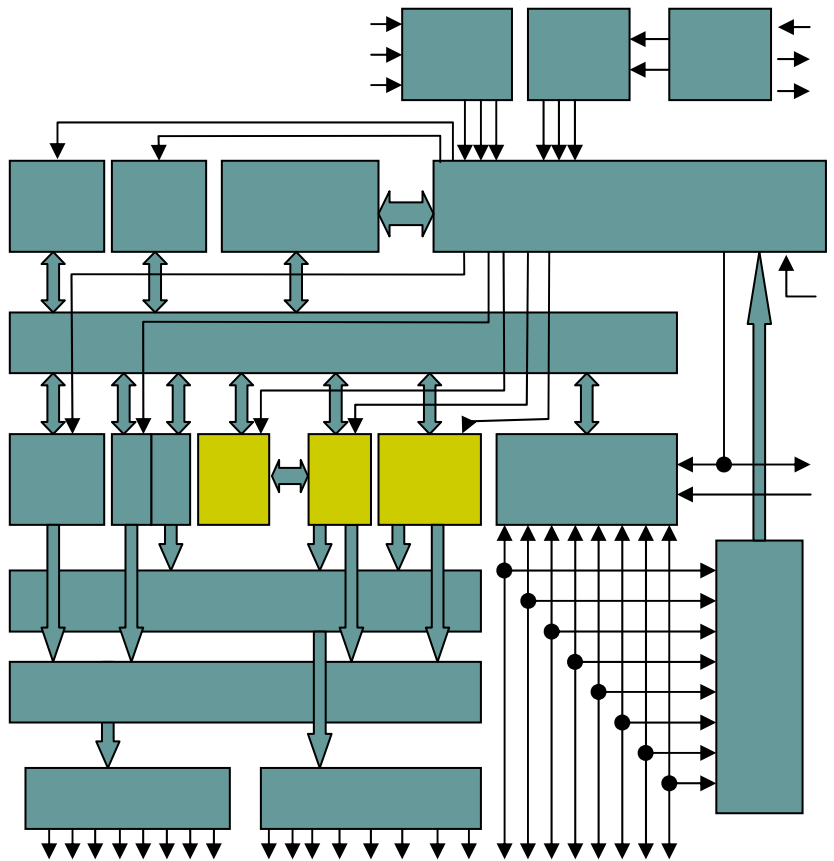
- Changed in several ways:
 - Jump instructions store next address to both PC registers
 - Branch instructions use relative addressing, alter PCL, and PCH on overflow
 - After state saved, set to interrupt handler vector address on interrupt





Foundation: 6502 Datapath Instruction Execution

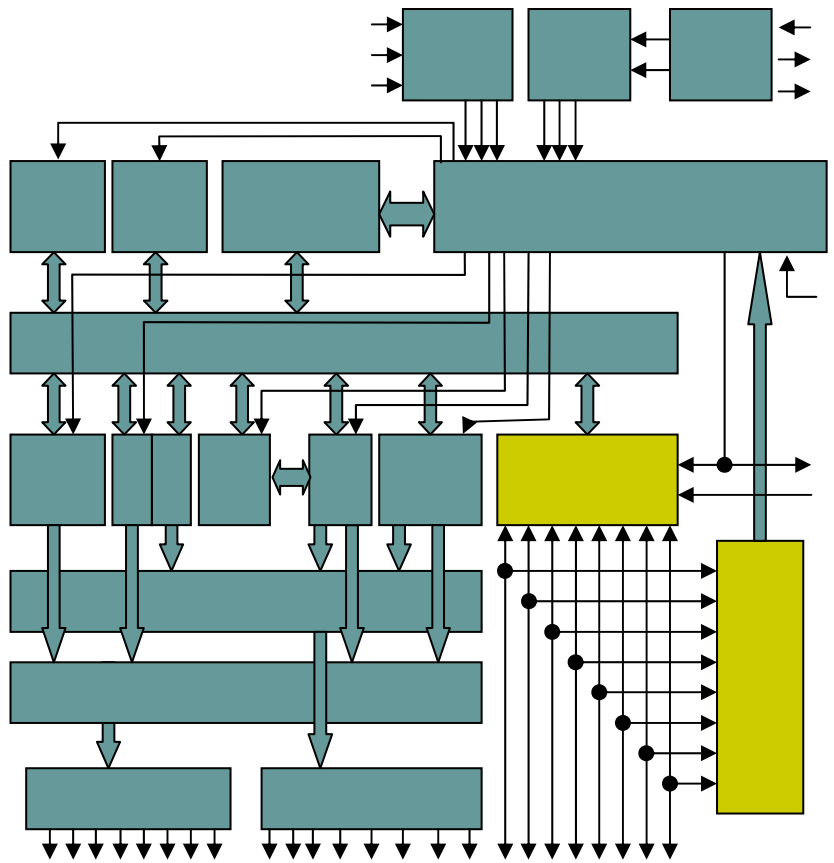
- Most operations have semantics
 $A \leftarrow A \text{ OP MEM}$
 - A is the accumulator register
 - MEM is the memory location of the second argument
- Input data latched from data bus



Foundation: 6502 Datapath

Data Bus

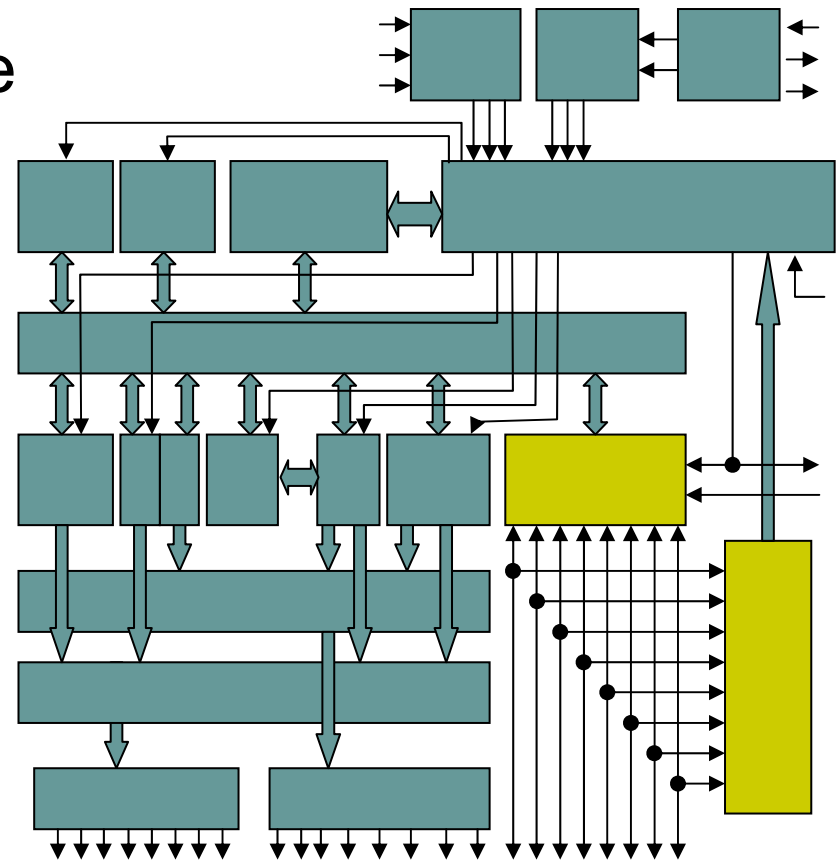
- One byte is read from or written to memory on every cycle – instruction or data
- Input broadcast to
 - Data Bus Buffer – to be placed on internal data bus
 - Instruction Register – to be read by instruction decode



Foundation: 6502 Datapath

Data Bus (cont.)

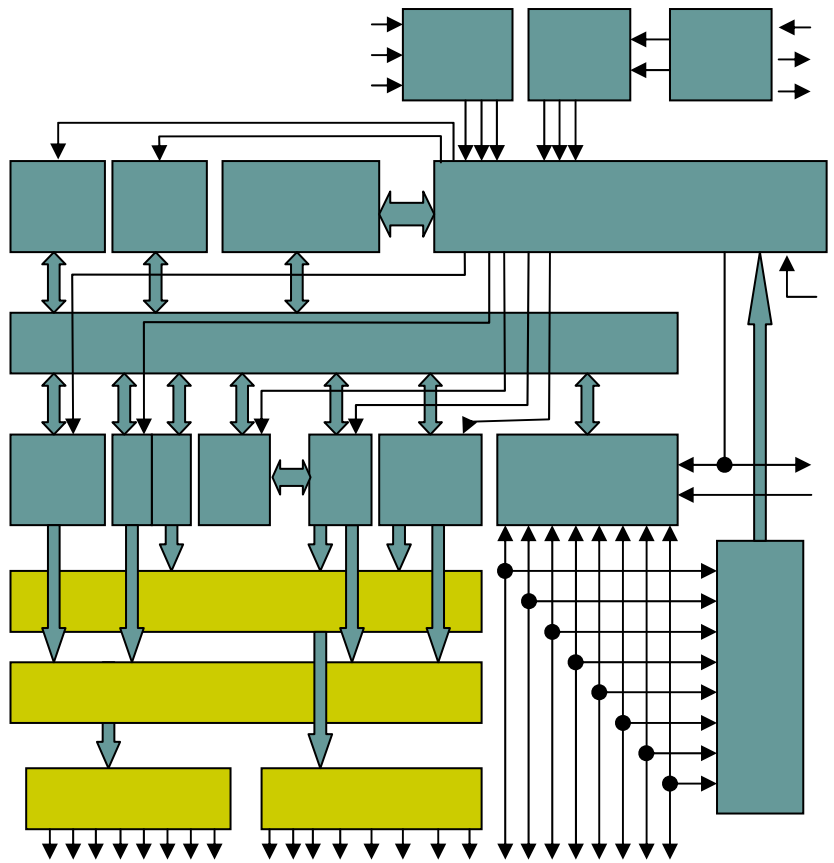
- Store data latched in Data Bus Buffer before output
- Decoder controls Read/Write mode, which is broadcast to external devices


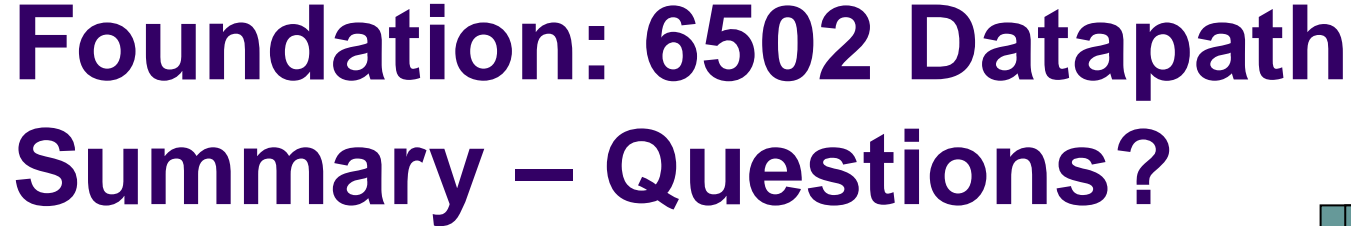



Foundation: 6502 Datapath

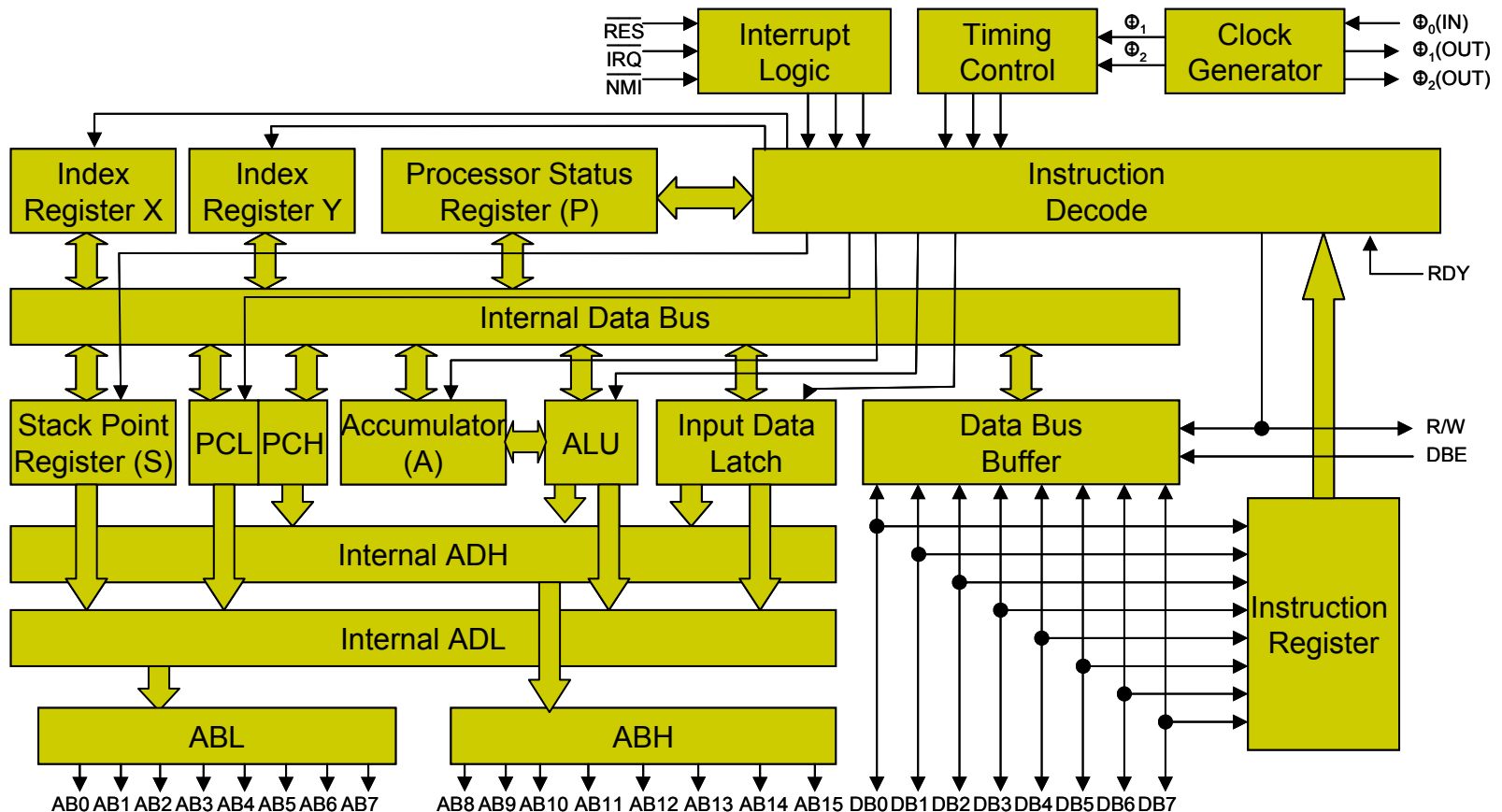
Address Bus

- 64k memory space addressed by high and low address bytes
- Several components can be source for address, e.g. PC, Stack, ALU
 - Dedicated high and low bus lines handle traffic to address registers





→ = 1-bit line
 = 8-bit line





Foundation: 6502 Timing

- External clock signal is converted into two-phase internal clock
 - Phase one: ALU operations executed, addresses placed on address bus
 - Phase two: Data transferred
 - Every single cycle includes one byte of instruction or data read or written – this places limits on how far away, slow memory can be

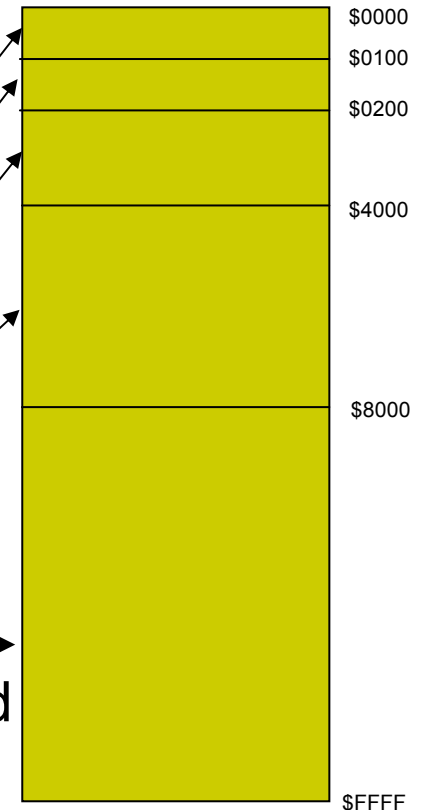


Foundation: 6502 Pipelining

- Ever so slightly parallel
 - Most of processor time is spent reading instructions and arguments
 - Most instructions only need one cycle to execute once arguments arrive
 - Instruction n can usually perform its execution “stage” while first byte of instruction $n+1$ is read.
 - But, before $n+1$ fetched, multiple cycles spent reading instruction n arguments, so no real appearance of parallelism

Foundation: 6502 Memory Map

- 16 bit addresses accessed every cycle
 - Entire memory space limited to 64k (NES works around that...)
- Memory allocation largely up to programmer
 - Suggested, however is as follows
 - \$0000-\$00FF: Zero Page RAM
 - \$0100-\$01FF: Stack
 - \$0200-\$3FFF: RAM
 - \$4000-\$7FFF: I/O Devices
 - \$8000-\$FFFF: ROM
- Most programs (including NES cartridges) held in ROMs



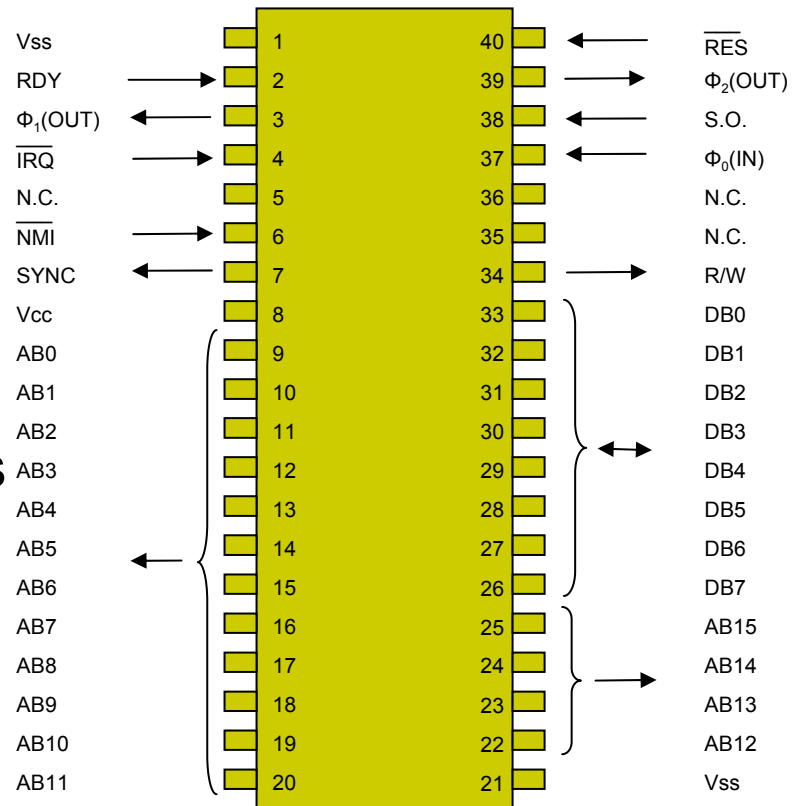


Foundation: 6502 Memory Map

- I/O is all memory-mapped. With one data transfer every cycle, I/O has to be fast
 - But can de-assert processor's RDY input to make processor wait for slow data
- No cache! During 6502 heyday, the memory gap hadn't yet widened. We go all the way to memory every single cycle
 - But our memory is only 64K, and our processor only runs at 1MHz

Foundation: 6502 Pinout

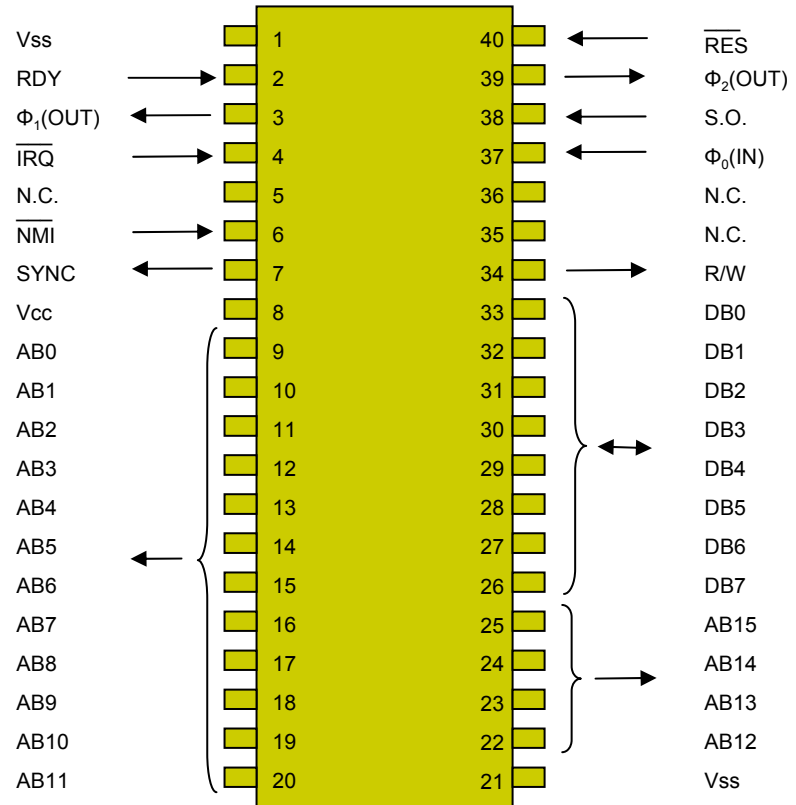
- AB0-AB15: Address Bus
- DB0-DB7: Data Bus
- Vcc, Vss: Voltage lines
- <not>IRQ, <not>NMI: Interrupt Lines
- <not>RES: de-asserted during machine startup
- S.O.: Set Overflow – allows external devices to set the overflow flag
- N.C.: No Connection





Foundation: 6502 Pinout (cont.)

- $\Phi_1(\text{OUT})$, $\Phi_2(\text{OUT})$: Allow external devices to monitor internal clock phases
- $\Phi_0(\text{IN})$: Clock input
- R/W: Controls direction of data transfer
- RDY: Delays execution when pulled low
- SYNC: Goes high during Opcode fetch
 - used with RDY to allow single-step execution





Enough, Already, about the 6502

- NES runs on the Ricoh RP2A03G (commonly 2A03)
 - A 6502 clone, but with some alterations to support Nintendo gaming
 - These differences, and the additional NES Picture Processing Unit, are the real subject of this presentation
 - Your new 6502 background will help you understand not only the NES, but also the Atari 2600, Commodore 64, Apple II, etc.



Processor Differences

- 2A03 uses same instruction set, but...
- Clocked a little faster (1.8Mhz)
- NES Processor (2A03) has no binary-coded decimal mode
 - Set/Clear Decimal Mode instructions are still there, but all arithmetic is always in binary
 - Executing these instructions doesn't cause an error, and status flag still exists
 - So there's an extra status flag you can program and branch on if you want
- 2A03 has no single-step capability

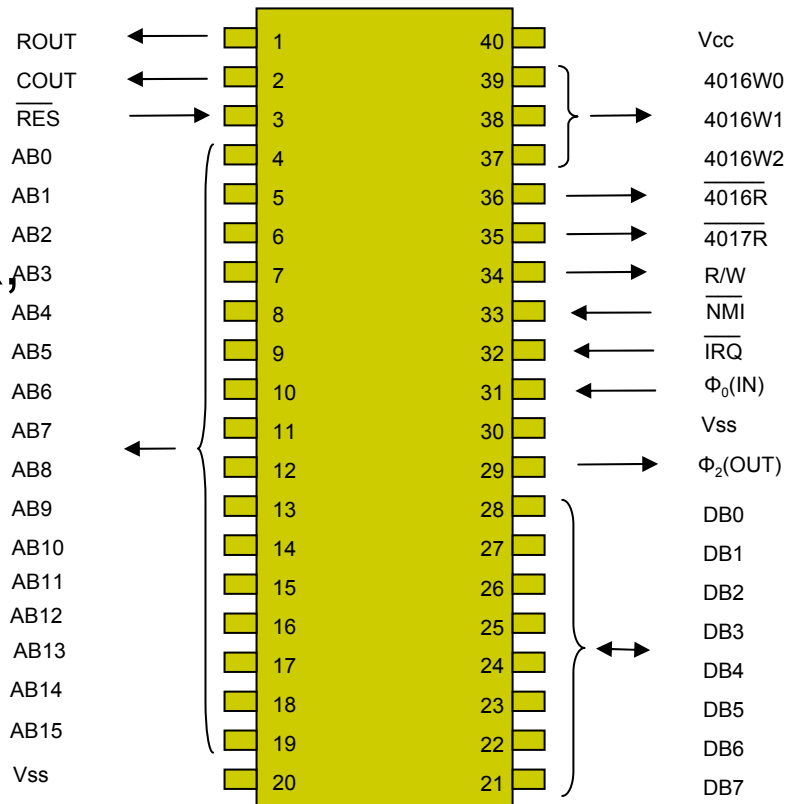


Processor Differences (cont.)

- 2A03 is not only the NES central processor, but also its pAPU
 - That is, psuedo-Audio Processing Unit
 - Five audio channels supported on processor
 - Essentially, 2 melody, 1 bass, 1 percussion, 1 for samples
 - We'll cover sound in more depth later
- 2A03 has internal programmable clock, for generating low frequency signals for audio
- Beyond these, 2A03 is a typical 6502

2A03 Pinout: A few new pins, a few different pin locations

- ROUT: Mixed output for audio channels 1-2
- COOUT: Mixed output for audio channels 3-5
- 4016W bus, <not>4016R, <not>4017R: Represent internal state during controller interaction





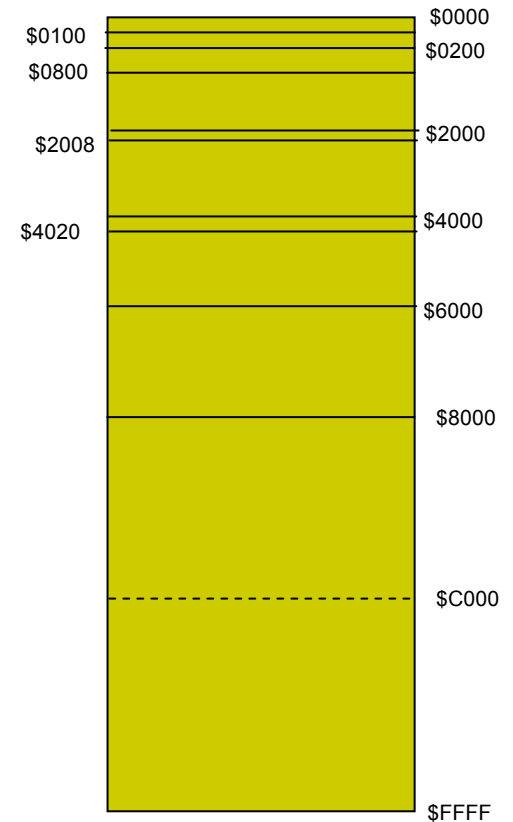
Wait, no new instructions?!

- If there are no new instructions, how does the NES
 - Read controllers?
 - Play sounds?
 - Display video?
- Memory Mapped I/O
 - By reading from and writing to special memory locations, processor controls display, audio, and input



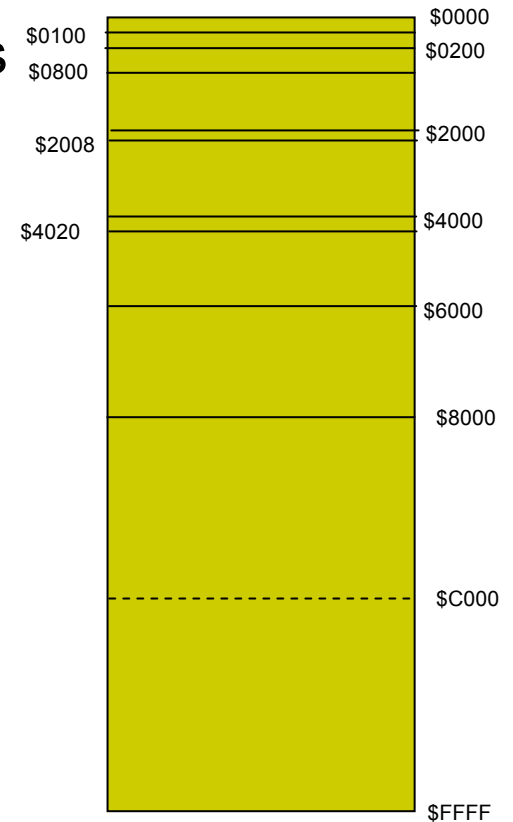
NES Memory Map

- \$0000-\$00FF: Zero Page RAM
- \$0100-\$01FF: Stack
- \$0200-\$07FF: RAM
- \$0800-\$1FFF: Mirrors \$0000-\$07FF
- \$2000-\$2007: I/O Registers
- \$2008-\$3FFF: Mirrors \$2000-\$2007
- \$4000-\$4019: I/O Registers
- \$4020-\$5FFF: Expansion ROM
- \$6000-\$7FFF: SRAM
- \$8000-\$BFFF: PRG-ROM (Lower Bank)
- \$C000-\$FFFF: PRG-ROM (Upper Bank)



NES Memory (cont.)

- Lots of mirroring
 - 64k addressable space, but much of this is mirrored due to limited physical space
- Game ROM limited to 32k
 - Some games as large as 1MB
 - Overcome limit with “Memory Management Chips (MMCs)”
 - MMCs intercept write to ROM, switch to desired memory bank
- SRAM: Battery-backed cartridge RAM used to store saved games





I/O Registers: Controllers

- Memory locations \$4016 and \$4017 correspond to controllers 1 and 2
- Begin process by writing strobe to \$4016:
- Then, read from either \$4016 or \$4017
 - For 8 reads, LSB indicates button pressed:
 - Next 8 reads used if 4 controllers attached
 - Next 8 reads return status
 - Is controller attached to this port?
 - What kind of controller?
- Higher bits of \$4017 also used in sound processing

```
LDA #1  
STA $4016  
LDA #0  
STA $4016
```

```
LDA $4016 // A Button  
LDA $4016 // B Button  
LDA $4016 // Select  
LDA $4016 // Start  
LDA $4016 // Up  
LDA $4016 // Down  
LDA $4016 // Left  
LDA $4016 // Right
```




NES Audio Processor

- 2A03 includes ability to process 5 audio channels
 - 2 square-wave channels (Pulse#1, Pulse#2)
 - Used for melody lines, sound effects
 - 1 Triangle-wave channel (Triangle)
 - Used for bass line
 - 1 Random-wave channel (Noise)
 - Used for percussion
 - 1 Digital channel (PCM)
 - Used for sample playback



I/O Registers: Audio

- \$4000-\$4013,\$4015,\$4017 control audio
 - Pulse#1: \$4000-\$4003
 - Pulse#2: \$4004-\$4007
 - Triangle: \$4008-\$400B
 - Noise: \$400C-\$400F
 - DMC: \$4010-\$4013
 - Audio Status Register: \$4015
 - Audio Clock Control: \$4017
- Registers have multiple values crammed into each byte
 - frequency shift, decay, volume, sample address, wavelength, etc.
- This is very low-level sound processing
 - Fortunately, there are audio tools to help compose music



NES Video

- US Televisions are 256x240 pixels
- Each pixel is drawn 60 times per second
 - Period when TV beam resets from right to left: HBLANK
 - Period when TV beam resets from bottom to top: VBLANK
- The NES offloads this work to a dedicated Picture Processing Unit (PPU), the Ricoh 2C02
- Driven by a 21.48Mhz clock, outputs unbuffered composite video



NES Video: Painting

- PPU uses four tables to paint each pixel
 - Saves memory by limiting duplication
 - Pattern Tables (one for backgrounds, one for sprites)
 - 2k 8x8 pixel patterns with low two bits of color key
 - Name Tables (two in memory)
 - 32x30 tables of pointers into backgrounds pattern table
 - Attribute Tables (one for each name table)
 - High 2 bits of color key for 4x4 tile blocks of background tiles
 - Color Palettes (one for background, one for sprites)
 - 13 background, 12 sprite colors keyed by attribute/pattern bits



NES Video: Pattern Tables

- Two 2k indexed tables of 8x8 pixel pattern tiles, with low two bits of color
- Two layer table, bit 0 in first layer, bit 1 in second

00000011		00000000		00000011
00001111		00000000		00001111
00011111		00000000		00011111
00011111		00000000		00011111
00000011	+	00011111	=	00022233
00011011		00111111		00233233
00011001		00111111		00233223
00011001		00111111		00233223



NES Video: Name Tables

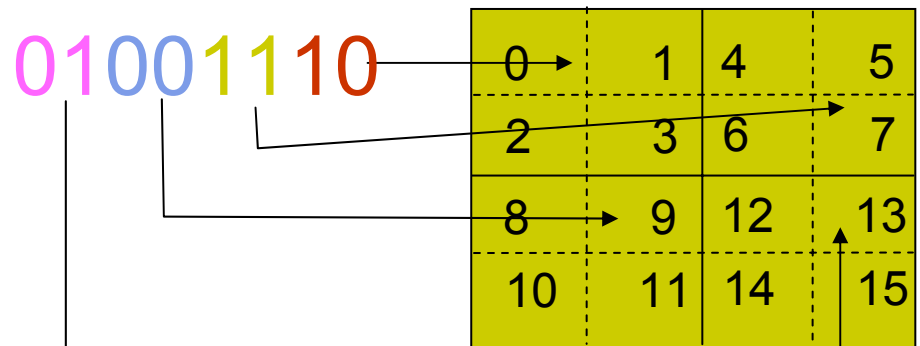
- Tables of 32x30 indexes into Pattern Table
- NES can address 4 name tables
 - But only enough physical memory for 2
 - So 2 are mirrors, unless cartridge supplies additional video RAM
- Entries in name tables correspond to 960 8x8 screen positions
 - 32 left to right, 30 top to bottom
 - A few more lines than televisions actually display



NES Video: Attribute Tables

- One attribute table for each name table
- Divide name tables into 2x2 blocks of 4x4 pattern tiles (or 32x32 pixels)
- Each entry in attribute table supplies high two color bits for tiles in blocks
- Attribute Table

Entry:





NES Video: Color Palettes

- NES capable of displaying 64 colors
 - But not all at one time
- Color palettes convert color keys from pattern/attribute tables to displayed screen colors
- 4 bit color key allows 16 background, 16 foreground colors
 - But color at \$0 is the transparent background color, and is mirrored at position \$0, \$4, \$8, and \$C of each palette
 - So effectively 13 background, 12 foreground colors available at any one time

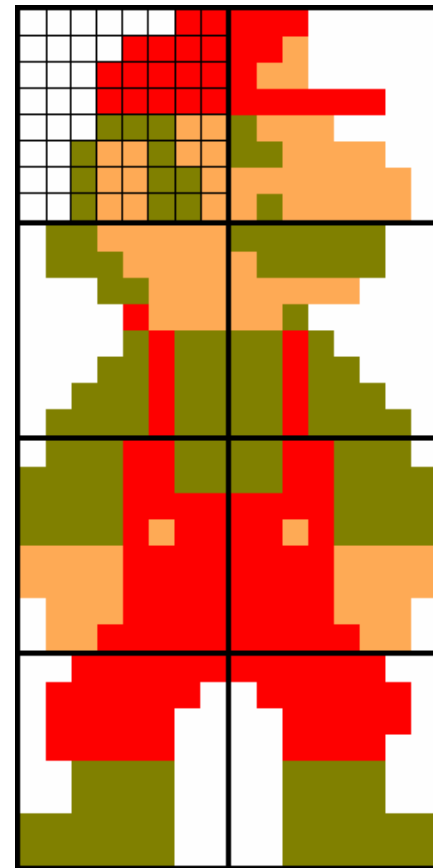


NES Video: Sprites

- Sprites are patterns (8x8 or 8x16) that are painted over the background
- Separate 256 byte sprite memory contains 64 4-byte sprites
 - For each sprite, byte 0 holds (Y coordinate – 1)
 - -1 because sprites are read while the previous scan line is drawn
 - Byte 1 contains index into pattern table
 - 8x16 sprites index into both pattern tables
 - Byte 2 contains sprite attributes:
 - Bits 0-1: upper two bits of color key
 - Bits 2-4: unused
 - Bit 5: behind background?
 - Bit 6: flipped horizontally?
 - Bit 7: flipped vertically?
 - Byte 3 contains X coordinate

NES Video: Sprites and Characters

- Sprites are pattern tiles that are painted over background
- Most characters are made up of multiple sprites
 - This plumber is made up of 8 sprites and a pendant for saving princesses



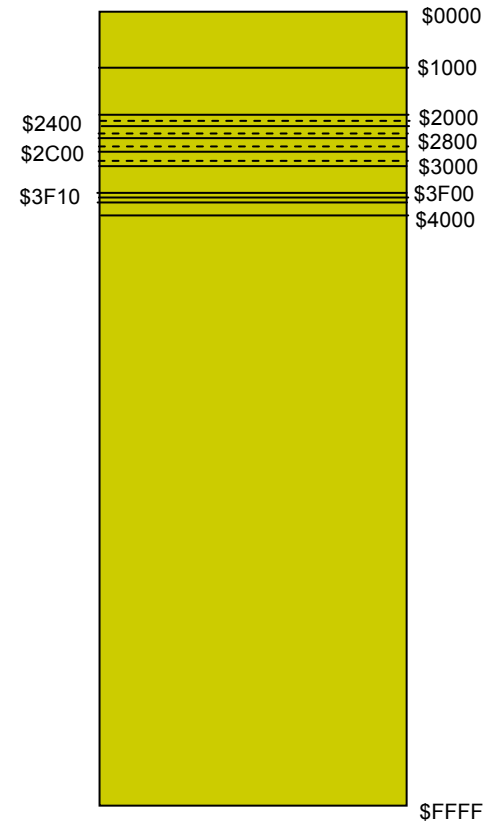


NES Video: Painting Sprites

- Before rendering each line, sprites checked to see if they are on that line
 - Limit 8 sprites per scan line
- Active sprites rendered in memory-order
 - If non-transparent sprite 0 pixel collides with non-transparent background pixel, PPU Status Register bit is set
 - Monitor this from CPU to help implement scrolling

PPU Memory Map

- \$0000-\$0FFF: Pattern Table 1
- \$1000-\$1FFF: Pattern Table 2
- \$2000-\$23BF: Name Table 0
- \$23C0-\$23FF: Attribute Table 0
- \$2400-\$27FF: Name/Attribute Tables 1
- \$2800-\$2BFF: Name/Attribute Tables 2
- \$2C00-\$2FFF: Name/Attribute Tables 3
- \$3000-\$3EFF: Mirrors \$2000-\$2EFF
- \$3F00-\$3F0F: Image Palette
- \$3F10-\$3F1F: Sprite Palette
- \$3F20-\$3FFF: Mirrors \$3F00-\$3F1F
- \$4000-\$FFFF: Mirrors \$0000-\$3FFF
- Lots of mirroring shows limited physical memory





I/O Registers: Video

- Like the audio and controllers, the CPU controls the CPU by writing and reading to specific memory locations
- \$2000: PPU Control Register 1
 - Bits 0-1: Name table selection
 - Bit 2: Address increment size
 - Either 1 (0) for horizontal movement, or 32 (1) for vertical
 - Bit 3: Sprite pattern table selection
 - Bit 4: Background pattern table selection
 - Bit 5: Sprite pixel size (8x8 (0) or 8x16 (1))
 - Bit 6: PPU master/slave mode (not used in NES)
 - Bit 7: Should NMI occur on VBLANK?
 - CPU can get interrupt during screen refresh or poll \$2002.7



I/O Registers: Video (cont.)

- \$2001: PPU Control Register 2
 - Bit 0: Color (0) or monochrome (1) mode
 - Bit 1: Hide (0) or show (1) left 8 background pixels
 - Bit 2: Hide (0) or show (1) sprites in left 8 pixels
 - Bit 3: Hide (0) or show (1) background
 - Bit 4: Hide (0) or show (1) sprites
 - Bits 5-7: Monochrome mode background color or color mode color intensity
- \$2002: PPU Status Register
 - Bit 4: If set, writes to VRAM ignored
 - Bit 5: If set, current scan line had >8 sprites
 - Bit 6: Set when Sprite 0 hits
 - Bit 7: Set during VBLANK



I/O Registers: Video (cont.)

- \$2003: SPR-RAM Address Register
 - Holds address in sprite RAM to access on next write to \$2004
- \$2004: SPR-RAM I/O Register
 - Writes a byte to sprite RAM at address from \$2003
- \$4014: Sprite DMA Register
 - Writing to \$2003, \$2004 for all 64 sprites is tedious
 - NES allows DMA to transfer an entire page of memory to sprite RAM
 - STA #\$4014 transfers the memory page addressed by (accumulator value * \$100) into sprite RAM
 - Sprite DMA steals memory bus cycles and stalls CPU, but is faster than writing 256 bytes via LDA, STA calls



I/O Registers: Video (cont.)

- \$2005: VRAM Register 1
 - 2 consecutive writes tell PPU where to start next scan line in name table
 - X-index into name table, x-offset of pattern tiles
 - Y-index into name table, y-offset of pattern tiles
 - X-index is incremented as each tile is drawn.
 - Alternate name table used if X-index wraps over 31
 - Y-index is incremented as each row is drawn
 - Alternate name table used if Y-index wraps over 29



NES Video: Scrolling

- Implement scrolling by writing to \$2005 during VBLANK
 - Can have split-screen horizontal scrolling by writing to \$2005 during screen drawing:
 - 1) Detect Sprite 0 Hit (plant sprite 0 where you want mid-screen scroll to start)
 - 2) Increment the x-offset and write to \$2005
 - 3) Change will become apparent after next HBLANK



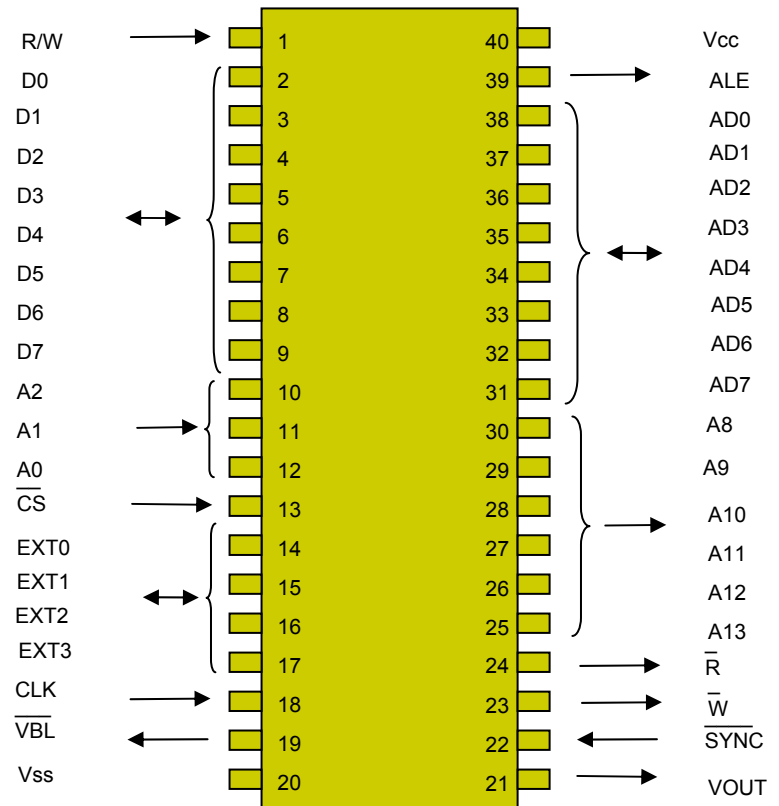
I/O Registers: Video (cont.)

- \$2006: VRAM Address Register
 - 2 consecutive writes give address in video RAM to access on next read/write to \$2007
 - Most significant address bits
 - Least significant address bits
- \$2007: VRAM Register
 - Reads/Writes a byte to VRAM at address from \$2006
 - Increments address in \$2006 by either 1 or 32, depending on \$2000.2
 - First read is invalid, subsequent reads return data beginning at specified address

```
LDA #%10001000
STA $2000 // Set increment
LDA #$04
STA $2006
LDA #$12
STA $2006
LDA $2007 // Invalid
LDA $2007 // VRAM at $0412
LDA $2007 // VRAM at $0413
LDA $2007 // VRAM at $0414
```

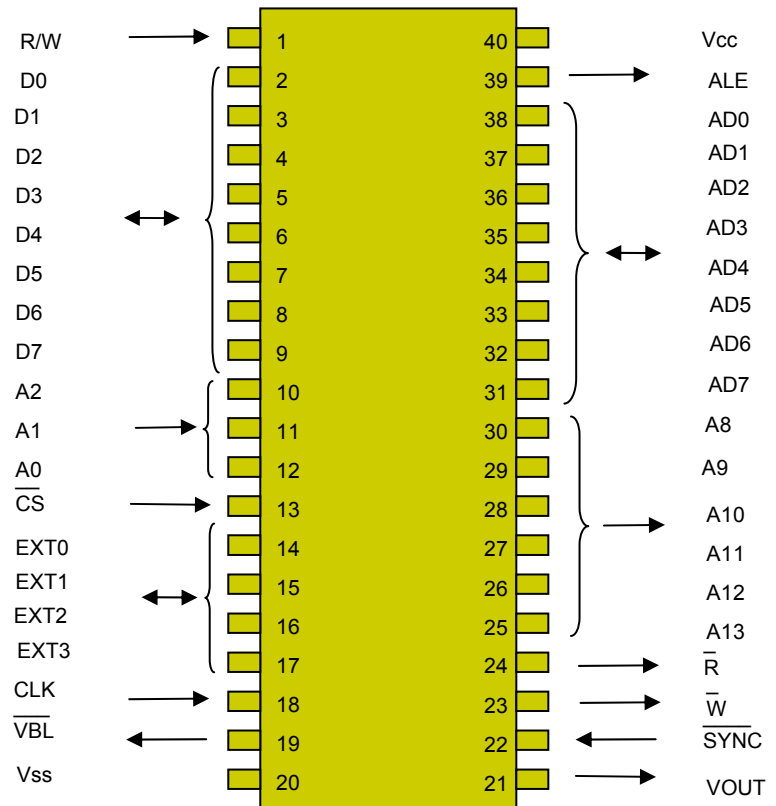
PPU Pinout

- R/W, A2-A0, D0-D7, <not>CS: Control Signals
 - R/W: Controls data direction
 - A0-A2: Choose internal register
 - D0-D7: Data bus
 - <not>CS: Activate transfer
- EXT0-EXT3: Interact with external video processors (disabled on NES)
- CLK: Clock input
- <not>VBL: Goes low during VBLANK period (display reset)
 - Tied to 2A03 <not>NMI line



PPU Pinout (cont.)

- VOUT: Composite video output
- $\overline{\text{not}}\text{SYNC}$: When low, forces internal state to reset/blank
- $\overline{\text{not}}\text{R}$, $\overline{\text{not}}\text{W}$, ALE, AD0-AD7, A8-A13: Data bus control
 - ALE goes high when low address bits on AD bus
 - R or W indicate to read high bits of address from A bus, drive data in indicated direction on D bus
 - Only one of R,W,ALE active at a time





Programming the NES

- Initialize memory during RESET
 - Pattern tables are loaded from cartridge at boot time
- Update/Read VRAM during VBLANK
 - If \$2000.7 is set, PPU signals NMI on VBLANK
 - Otherwise, program can create a spin lock on \$2002.7
 - Includes changing colors, name tables, sprite locations, etc.
- Do the rest of your processing on your own time
 - Handle controller input
 - Handle sprite collision detection
 - Produce audio
 - Sprite DMA can come in handy – calculate new positions in processor time, run DMA in VBLANK time
 - All in plain old 6502 assembly



NES Cartridges

- 72 pin connector between cartridge and NES
- PRG-ROM chip(s) contain the program (30 pins)
- CHR-ROM chip(s) contain pattern tables (30 pins)
- MMCs allow higher ROM sizes
- Last 12 pins connect to lockout chip
 - Internal NES security chip establishes communications protocol with cartridge lockout chip
 - NES sold lockout chips as part of licensing to game developers



Sources: 6502

- SY6500/MCS6500 Microcomputer Hardware Manual. Synertek Incorporated. May, 1978.
- SY6500/MCS6500 Microcomputer Programming Manual. Synertek Incorporated. August, 1976.
- 6502 Wikipedia entry.
http://en.wikipedia.org/wiki/MOS_Technology_6502



Sources: NES

- Nintendo Entertainment System Architecture. Marat Fayzullin. January, 2005.
- Nintendo Entertainment System Documentation. Patrick Diskin. August, 2004.
- Nintendo Entertainment System Documentation. Jeremy “Yoshi” Chadwick. October, 1999.
- NES Technical/Emulation/Development FAQ. Chris Covell. January, 2002.
- NES 101: A NES Tutorial for Otherwise Experienced Programmers. Michael Martin. March, 2002.
- 2A03 Technical Reference. Brad Taylor. April, 2004.
- NTSC 2C02 Technical Reference. Brad Taylor. April, 2004.
- The Skinny on NES Scrolling. Loopy. April, 1999.
- NesDev portal. <http://nesdev.parodius.com>.
 - Hosts many of above, including anonymous “How NES Graphics Work”