MOS 6502
Architecture
Lecture 3
History

- Origins lie in the Motorola 6800. Was very expensive for consumers. ($300, or about $1500 in 2017 $s)

- Chuck Peddle proposes lower-cost, lower-area 6800 design (~$25, ~$100 today). Motorola wont’ have it. Chuck moves to MOS tech.

- Chooses much simpler design, only about 3500 transistors (!)
History (contd.)

- 1975 sees a recession, sales of 6502 not good
- Peddle comes up with MDT-650, a single-board minicomputer/ dev. platform, which hobbyists eat up
- Other players see the potential, and the hobbyist game is on (Apple, Atari, Commodore, etc.)
- 6502 then used in Apple ][, Commodore PET, BBC Micro, Atari 800, and more
Arch. Overview

• Byte-addressable, 16-bit address width. Word size is one byte.

• 3 8-bit registers (1 accumulator, 2 index regs for addressing modes (e.g. array subscripting))

• 8-bit stack pointer (SP)

• 8-bit status register (I’ll call this PSW)

• 16-bit program counter (PC)

• Little Endian
Processor Memory Map

- **ROM**: 0x???? - 0xFFFF
- **I/O Space**: 0x???? - 0x????
- **Usable RAM**: 0x00200 - 0x????
- **Stack**: 0x0100 - 0x0200
- **Zero Page**: 0x0000 - 0x0100
Address Breakdown

We can get the page number of an address by shifting off the lower 8 bits.
Instruction Encoding

- 256 possible opcodes, only 56 are actually used by the architecture
- 1, 2, or 3-byte instructions
- Data movement (LD/ST)
- Arithmetic: ADD (with carry), SUB (with carry), DEC, INC, CMP
- Logic: AND, OR, XOR, shift/rotate
- Control Flow: branch cond, branch uncond, call subroutine, return
- Other: PS manipulation, bit testing, stacks ops
Program Counter

- (16 bits) Holds address of current instruction being executed
Stack Pointer

- Contains address or first empty location on stack (page 2)
Processor Status Word

- Holds status information for the processor. Each bit represents a flag.

- Bit 0: "C" - Carry, Carry out of MSB in arithmetic ops. Also set if borrow required in SUB. Also used for shift/rotate ops.

- Bit 1: "Z" - Set to 1 if any arith/logic operation produces a zero.

- Bit 2: "I" - Interrupt disable. If set, interrupts are disabled (ignored). (Except NMIs).

- Bit 3: "D" - decimal mode status. We will ignore this (like the NES). Indicates to arith. unit to use BCD representation.

- Bit 4: "B" - Set when software interrupt is executed (BRK instruction).

- Bit 5: unused.

- Bit 6: "V" - Overflow flag: when arithmetic operation overflows, this is set.

- Bit 7: "N" - Sign bit. Set when result of an operation is negative.
Addressing Modes

- **Immediate**: operand’s value is in the instruction
- E.G. LDA #$B7 —> load accumulator reg with value 0xb7
- The # symbol indicates that this is an immediate
Addressing Modes

- **Absolute**: memory address included as operand in instruction
  - E.g. LDA $07A3 —> Load accumulator with contents of memory at 0x07A3. (opcode 0xa3)
  - Note: If page number is zero, this means a “zero page” reference. Uses different opcode!
  - E.g. LDA $F7 —> LDA $00F7 (opcode 0xa5)
Addressing Modes

- **Implied**: no operand necessary, implied by instruction

- E.g. TAX instruction (transfer contents of accumulator to X register) (opcode 0xaa)
Addressing Modes

• **Indexed:** Use a base register (either X or Y) and add it to the address given as operand

• E.g. LDA $075A, X —> A <- Mem[X + 0x75A]

• Same zero page rules apply. Note that most instructions only use X with zero page!
Addressing Modes

• **Indirect:** Add a level of indirection to address operand. NOTE: only used for JMP instruction!

• E.g. JMP ($07A5) → Jump to the address stored in the bytes at $07A5
Addressing Modes

- **Indirect:** Add a level of indirection to address operand. NOTE: only used for JMP instruction!

- E.g. JMP ($07A5) —> Jump to the address stored in the bytes at $07A5
Interrupts

- **IRQ** - Maskable interrupt. When invoked, PC and PS stored on stack
- Further interrupts are disabled by the processor until handled
- Processor jumps to address of handler that is stored in 0xFFFFE (2 bytes). Handler (likely in ROM) returns with RTI instruction.
- IRQ is masked/unmasked with CLI/SEI instructions
Interrupts (contd.)

- **NMI** - Non-maskable interrupt. Same sequence, but processor jumps to handler addr stored at 0xFFFA. This interrupt can’t be disabled!
Interrupts (contd.)

- **BRK** - Software interrupt. Same operation, but B flag is set in PSW stored on stack. CPU fetches from 0xFFFE (same vector as IRQ!)
Interrupts (contd.)

- **RESET** - system reset. Nothing pushed on stack, fetch at vector 0xFFFFC, otherwise same