Operating Systems

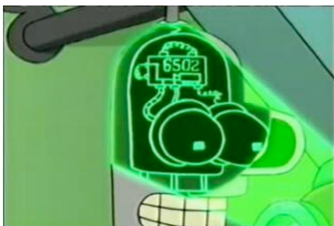
CMSC 422 and MSCS 515

Machine Language Instruction Set

The instruction set for our OS CPU simulation's machine language is based on the op codes of the classic 6502 microprocessor. It was the heart of the Commodore PET, Apple //, and Bender, so we're in good company using it ourselves.

There is an excellent virtual 6502 simulator, assembler, and disassembler at http://e-tradition.net/bytes/6502. Make frequent use of this tool so that you can test your machine code there before trying it in your OS. You should spend time debugging your OS, not the user programs. This also saves you the step of writing your own assembler, although we might want to do that later on.

```
                                                       Example      Example
Description                          Op Code Mnemonic  Assembly     Disassembly
------------------------------------ ------- --------  ------------ -----------
Load the accumulator with a constant   A9      LDA     LDA #$07      A9 07
Load the accumulator from memory       AD      LDA     LDA $0010     AD 10 00

Store the accumulator in memory        8D      STA     STA $0010     8D 10 00

Add with carry                         6D      ADC     ADC $0010     6D 10 00
 Adds contents of an address to
 the contents of the accumulator
 and keep result in acculator

Load the X register with a constant    A2      LDX     LDX #$01      A2 01
Load the X register from memory        AE      LDX     LDX $0010     AE 10 00

Load the Y register with a constant    A0      LDY     LDY #$04      A0 04
Load the Y register from memory        AC      LDY     LDY $0010     AC 10 00

No Operation                           EA      NOP     EA            EA

Break (which is really a system call)  00      BRK     00            00

System Call                            FF      SYS                   FF
 #$01 in X reg = print integer stored
 in the Y register.
```



The system call instruction (mnemonic `SYS`, op code `FF`) is not part of the real 6502 instruction set. But we need one, so let's invent it. When it comes time to execute SYS, your OS will examine the contents of the X register to see what system service is being requested by the user program. 01 = integer output. Once the OS knows that integer output has been requested, it reads the integer from the Y register and outputs it to standard out.

© 2008 Alan G. Labouseur
All Rights Reserved

alan@3NFconsulting.com
www.3NFconsulting.com

Operating Systems

CMSC 422 and MSCS 515

Machine Language Instruction Set

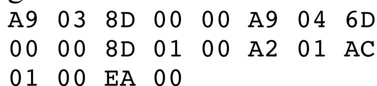
Example One

```
;
; OS/1.0 (OS class program one)
;
; Adds 3 and 4 and outputs result.
;
lda #$03    ; Load the accumulator (the "A register") with the constant 3.
sta $0000   ; Store A in location $0000;  (These are hex numbers.)
lda #$04    ; A <-- 4
adc $0000   ; Add the value in location $0000 to A and keep the result in A.
sta $0001   ; Store A (our result) in location $0001.
ldx #$01    ; Load the X register with the value 1 (for syscall)
ldy $0001   ; Load the Y register with our result.
sys         ; Make a system call to the OS (via a software interrupt)
brk         ; Software interrupt for normal termination
```

Assemble this into 6502 machine code at http://www.e-tradition.net/bytes/6502/assembler.html. Use only the assembly code. Comments will mess it up. You should get:

```
LDA #$03        A9 03
STA $0000       8D 00 00
LDA #$04        A9 04
ADC $0000       6D 00 00
STA $0001       8D 01 00  (Notice the low-order bytes are first ("little-endian"), so 0001 = address 01 00.)
LDX #$01        A2 01
LDY $0001       AC 01 00
SYS
BRK             00
```

Note that SYS does not cause an error (as the real 6502 did not have this), which is nice, but it also does not generate an op code. In order to make our code work in the emulator, let's use the op code for NOP (no operation) for SYS. That's EA. Inserting EA for SYS into the object code stream, we get:



```
A9 03 8D 00 00 A9 04 6D
00 00 8D 01 00 A2 01 AC
01 00 EA 00
```

Copy the object code and test it out at http://www.e-tradition.net/bytes/6502. You can see it run step by step. Be sure to set the start address to 0000. Also, once you load memory, click "show memory" to see the address-detailed display. You need to click "show memory" to see the updates as you step through the user program.

Test your code here so you can concentrate on getting the OS right, not worrying about the user code (for now). There is lots of cool stuff at that site, so check it all out.

© 2008 Alan G. Labouseur
All Rights Reserved

alan@3NFconsulting.com
www.3NFconsulting.com

Operating Systems

CMSC 422 and MSCS 515

Machine Language Instruction Set

Example Two

In the example one, we load the instructions beginning at location $0000. We also begin storing our values at $0000. This might be a bad idea, as we write over our own code with data. Let's store our data in locations elsewhere:

```
;
; OS/1.1 (OS class program one.1)
;
; Adds 3 and 4 and outputs result; doesn't overwrite our code in memory.
;
lda #$03    ; Load the accumulator (the "A register") with the constant 3.
sta $0018   ; Store A in location $0018;  (These are hex numbers.)
lda #$04    ; A <-- #$04
adc $0018   ; Add the value in location $0018 to A and keep the result in A.
sta $0019   ; Store A (our result) in location $0019.
ldx #$01    ; Load the X register with the value 1 (Used by syscall to denote integer output.)
ldy $0019   ; Load the Y register with our result.
sys         ; Make a system call to the OS (via a software interrupt)
brk         ; Software interrupt for normal termination
```

Assembly and Op-codes:

```
LDA #$03       A9 03
STA $0018      8D 18 00
LDA #$04       A9 04
ADC $0018      6D 18 00
STA $0019      8D 19 00
LDX #$01       A2 01
LDY $0019      AC 19 00
SYS
BRK            00
```

Remembering to substitute EA (nop) for out SYScall when using the emulator, we get object code:

```
A9 03 8D 18 00 A9 04 6D
18 00 8D 19 00 A2 01 AC
19 00 EA 00
```

Copy the object code and test it out at http://www.e-tradition.net/bytes/6502.

© 2008 Alan G. Labouseur
All Rights Reserved
alan@3NFconsulting.com
www.3NFconsulting.com