ECE 498 – Linux Assembly Language Lecture 5

Vince Weaver

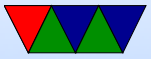
http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

29 November 2012

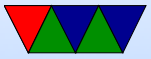
Clarifications from Lecture 4

- What is the **Q** "saturate" status bit?
- Some instructions on underflow or overflow instead of wrapping around saturate to 0xffffffff or 0x0.
  When they saturate like this, the Q bit is set.
- Only specialized instructions like `qadd` or `qsub` do this.

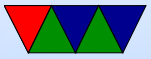


Put string example

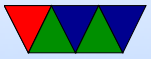
```
.equ SYSCALL_EXIT,      1
.equ SYSCALL_WRITE,     4
.equ STDOUT,            1

        .globl _start
_start:
        ldr     r1,=hello
        bl      print_string            @ Print Hello World
        ldr     r1,=mystery
        bl      print_string            @
        ldr     r1,=goodbye
        bl      print_string            /* Print Goodbye */

        #================================
        # Exit
        #================================
exit:
        mov     r0,#5
        mov     r7,#SYSCALL_EXIT        @ put exit syscall number (1) in eax
        swi     0x0                     @ and exit
```



```
        #====================
        # print string
        #====================
        # Null-terminated string to print pointed to by r1
        # r1 is trashed by this routine

print_string:
        push    {r0,r2,r7,r10}          @ Save r0,r2,r7,r10 on stack

        mov     r2,#0                   @ Clear Count

count_loop:
        add     r2,r2,#1                @ increment count
        ldrb    r10,[r1,r2]             @ load byte from address r1+r2
        cmp     r10,#0                  @ Compare against 0
        bne     count_loop              @ if not 0, loop

        mov     r0,#STDOUT              @ Print to stdout
        mov     r7,#SYSCALL_WRITE       @ Load syscall number
        swi     0x0                     @ System call

        pop     {r0,r2,r7,r10}          @ pop r0,r2,r7,r10 from stack

        mov     pc,lr                   @ Return to address stored in
```

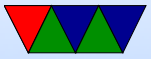


```
                                        @ Link register

.data
hello:          .string "Hello World!\n"    @ includes null at end
mystery:        .byte 63,0x3f,63,10,0       @ mystery string
goodbye:        .string "Goodbye!\n"        @ includes null at end
```



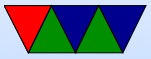
Clarification of Assembler Syntax

- @ is the comment character. # can be used on line by itself but will confuse assembler if on line with code. Can also use /* */
- Constant value indicated by # or $
- Optionally put % in front of register name

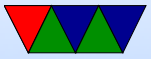


Instructions

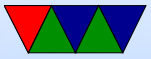
- adc, adcs – add with carry
- add, adds – add
- adr – add immediate to PC, store address in reg
- and, ands – bitwise and
- asr, asrs – arith shift right
- b – branch (can use all the condition code prefixes)



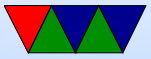
- bfc – bitfield clear, clear bits in reg
- bfi – bitfield insert
- bic, bics – bitfield clear: and with negated value
- bl, blx – branch and link (X means change to thumb)
- bx – branch and exchange to thumb
- clz – count leading zeros (armv7)
- cmp, cmn – compare, compare negative



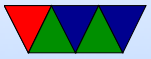
- eor, eors – exclusive or (name shows 6502 heritage)
- ldm, ldmia – load multiple memory locations into consecutive registers
- ldr – load register
- ldrb – load register byte
- ldrd – load double, into consecutive registers
- ldrh – load register halfword, zero extends
- ldrsb – load register signed byte, sign-extends



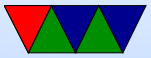
- ldrsh – load register halfword, sign-extends
- lsl – logical shift left
- lsr – logical shift right
- mla – multiply/accumulate (armv6). multiply two add in one
- mls – multiply and subtract
- mov, movs – move register
- movt – write value to top 16 bits of reg (armv6)



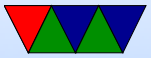
- mul – multiply two registers, only least sig 32bit saved
- mvn, mvns – move inverted
- nop – no-operation
- orn, orns – or not (armv6)
- orr, orrs – bitwise or
- pkh – pack halfword (armv6)
- pli etc – preload instructions



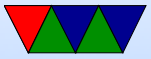
- pop – pop from stack. specify a list of registers
- push – push multiple registers
- qadd, qsub, etc. – saturating add/sub (armv6)
- rbit – reverse bit order (armv6)
- rbyte – reverse byte order (armv6)
- rev16, revsh – reverse halfwords (armv6)
- ror, rors – rotate right



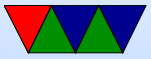
- rorx – rotate right extend, carry flag shift into bit 31
- rsb, rsbs – reverse subtract (immediate - rX)
- rsc, rscs – reverse subtract with carry
- sadd16 – do two 16-bit signed adds (armv6)
- sadd8 – do 4 8-bit signed adds (armv6)
- sasx – (armv6)
- sbc, sbcs – subtract with carry



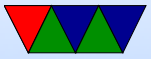
- sbfx – signed bit field extract (armv6)
- sdiv – signed divide (only armv7-R)
- sel – select bytes based on flag (armv6)
- setend – set endianess (armv6)
- sm* – signed multiply/accumulate
- stm – store multiple, can be used like a PUSH instruction
- str – store register



- strb – store byte
- strd – store double
- strh – store halfword
- sub, subs – subtract
- svc, swi – software interrupt
- swap – swap value between register and memory (deprecated armv7)
- sxtb – sign extend byte (armv6)

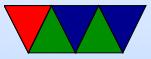


- tbb – table branch byte, jump table (armv6)
- teq – test equivalence (armv6)
- tst – test (and with value, don't save)
- u* – unsigned partial word instructions
- udiv – unsigned divide (armv7-R only)



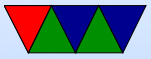
Thumb Instructions

- cbz, cbnz – compare and branch (zero/not-zero) can only branch forward?
- it – conditional block (see Lecture 6 notes)



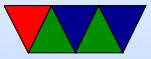
Instruction Sets

- ARM – 32 bit encoding
- THUMB – 16 bit encoding
- THUMB-2 – THUMB extended with 32-bit instructions
- THUMB-EE – some extensions for running in JIT runtime
- AARCH64 – 64 bit. Only currently exists in simulated form

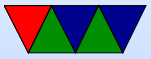


THUMB

- Most instructions length 16-bit (a few 32-bit)
- Some operands (sp, lr, pc) implicit
  Can't always update sp or pc anymore.
- Only r0-r7 accessible normally, add, cmp, mov can access high regs
- No prefix/conditional execution
- Only two arguments to opcodes



- 8-bit constants rather than 16-bit
- Limited addressing modes
- No shift parameter ALU instructions
- BX/BLX instruction to switch mode.
  If target is a label, *always* switch mode
  If target is a register, low bit of 1 means THUMB, 0 means ARM
- Can use `.thumb` directive, `.arm` for 32-bit.



AARCH64

Not all info has been released yet.

- 32-bit fixed instruction encoding
- 31 64-bit GP registers (x0-x30), zero register (x30)
- PC is not a GP register
- only branches conditional
- no load/store multiple