ECE 471 – Embedded Systems
Lecture 9

Vince Weaver

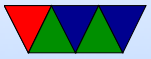
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

23 September 2019

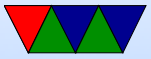
Announcements

- How is HW#3 going?



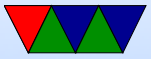
HW3 Notes

- Writing int to string conversion is a complex task
  There are lots of ways to do it.
- Good reverse engineering experience. Block of code from one of my older projects when I wasn't quite as good at ARM assembly.
- What does .lcomm do? Reserves region in the BSS. `.lcomm buffer,20` is similar to C `char buffer[20]`
- Went over algorithm. Need to divide by 10, put remainder into array backwards, then keep dividing the



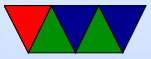
quotient. Also need to convert to ASCII.

- Corner cases: leading zero suppression?
- Dividing by 10 on system that has no divide? Use 32.32 fixed point multiply by 1/10. (429496730). ARM has umull instruction that will do a 32x32 multiply and give you the top half of the 64-bit result.



Setting Flags

- `add r1,r2,r3`
- `adds r1,r2,r3` – set condition flag
- `addeqs r1,r2,r3` – set condition flag and prefix compiler and disassembler like `addseq`, GNU as doesn't?

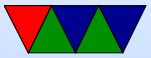


Conditional Execution

Why are branches bad?

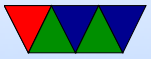
```
if (x == 5 )
    a+=2;
else
    b-=2;
```

```
        cmp     r1 , #5
        bne     else
        add     r2 , r2 ,#2
        b       done
else :
        sub   r3 , r3 ,#2
```



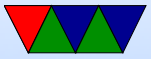
```
done:

        @ equivalent w/o branches
        cmp     r1, #5
        addeq   r2,r2,#2
        subne   r3,r3,#2
```



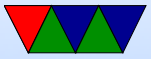
Why Code Density

- Smaller code can be better
- Lower resources: Cheaper? If you can fit more features into smaller RAM or disk you can save money
- Faster? It depends. Modern chips are really hard to predict, but if your processor has Caches and you can fit better in instruction cache it can potentially speed things up a lot



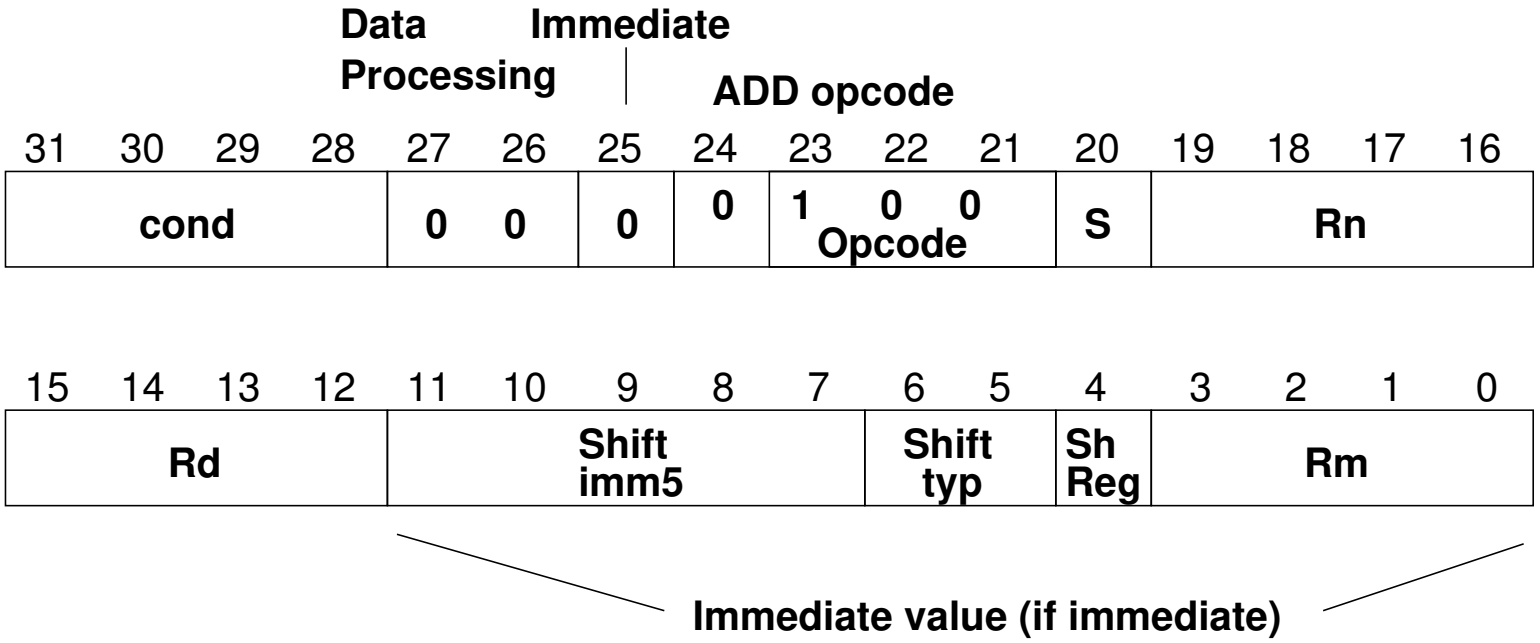
ARM Instruction Set Encodings

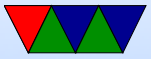
- ARM – 32 bit encoding
- THUMB – 16 bit encoding
- THUMB-2 – THUMB extended with 32-bit instructions
  - STM32L *only* has THUMB2
  - Original Raspberry Pis *do not* have THUMB2
  - Raspberry Pi 2/3 *does* have THUMB2
- THUMB-EE – extensions for running in JIT runtime
- AARCH64 – 64 bit. Relatively new. Completely different from ARM32



Recall the ARM32 encoding

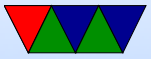
```
ADD{S}<c> <Rd>,<Rn>,<Rm>{,<shift>}
```



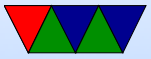


THUMB

- Most instructions length 16-bit (a few 32-bit)
- Only r0-r7 accessible normally
  `add`, `cmp`, `mov` can access high regs
- Some operands (sp, lr, pc) implicit
  Can't always update sp or pc anymore.
- No prefix/conditional execution
- Only two arguments to opcodes
  (some exceptions for small constants: `add r0,r1,#1`)
- 8-bit constants rather than 12-bit

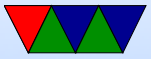


- Limited addressing modes: [rn,rm], [rn,#imm], [pc|sp,#imm]
- No shift parameter ALU instructions
- Makes assumptions about "S" setting flags
  (gas doesn't let you superfluously set it, causing problems if you naively move code to THUMB-2)
- new push/pop instructions (subset of ldm/stm), neg (to negate), asr,lsl,lsr,ror, bic (logic bit clear)

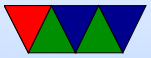


THUMB/ARM interworking

- See `print_string_armthumb.s`
- BX/BLX instruction to switch mode.
  Sets/clears the T (thumb) flag in status register
  If target is a label, *always* switchmode
  If target is a register, low bit of 1 means THUMB, 0 means ARM
- Can also switch modes with `ldrm`, `ldm`, or `pop` with PC as a destination
  (on armv7 can enter with ALU op with PC destination)

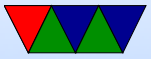


- Can use `.thumb` directive, `.arm` for 32-bit.



THUMB-2

- Extension of THUMB to have both 16-bit and 32-bit instructions
- The 32-bit instructions are *not* the standard 32-bit ARM instructions.
- Most 32-bit ARM instructions have 32-bit THUMB-2 equivalents *except* ones that use conditional execution. The `it` instruction was added to handle this.
- `rsc` (reverse subtract with carry) removed
- Most cannot have PC as src/dest



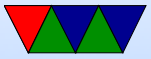
- Shifts in ALU instructions are by constant, cannot shift by register like in arm32
- THUMB-2 code can assemble to either ARM-32 or THUMB2

  The assembly language is compatible.

  Common code can be written and output changed at time of assembly.
- Instructions have “wide” and “narrow” encoding.

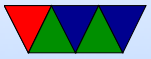
  Can force this (`add.w` vs `add.n`).
- Need to properly indicate “s” (set flags).

  On regular THUMB this is assumed.



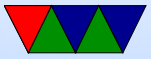
THUMB-2 Coding

- See `test_thumb2.s`
- Use `.syntax unified` at beginning of code
- Use `.arm` or `.thumb` to specify mode



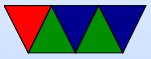
New THUMB-2 Instructions

- BFI – bit field insert
- RBIT – reverse bits
- movw/movt – 16 bit immediate loads
- TB – table branch
- IT (if/then)
- cbz – compare and branch if zero; only jumps forward



Thumb-2 12-bit immediates

```
top 4 bits 0000 -- 00000000 00000000 00000000 abcdefgh
           0001 -- 00000000 abcdefgh 00000000 abcdefgh
           0010 -- abcdefgh 00000000 abcdefgh 00000000
           0011 -- abcdefgh abcdefgh abcdefgh abcdefgh
rotate bottom 7 bits|0x80 right by top 5 bits
           01000 -- 1bcdefgh 00000000 00000000 00000000
            ...
           11111 -- 00000000 00000000 00000001 bcdefgh0
```



Compiler

- Original RASPBERRY PI DOES NOT SUPPORT THUMB2
- `gcc -S hello_world.c`
  By default is arm32
- `gcc -S -march=armv5t -mthumb hello_world.c`
  Creates THUMB (won't work on Raspberry Pi due to HARDFP arch)
- `-mthumb -march=armv7-a` Creates THUMB2