ECE 271 – Microcomputer Architecture and Applications Lecture 10

Vince Weaver

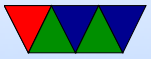
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

17 February 2022

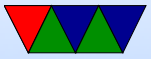
Announcements

- Read Chapter 2, Chapter 16
- Please have the prelab done by the beginning of the lab section
- The TA is handling deadlines for the labs so check with him first if you have questions about those. If you have concerns about his decisions then feel free to bring them up with me.



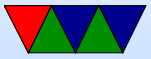
The Debugger / Debugging

- Debuggers let you track program behavior while code is running. Often the best way to find/fix problems in your code.
- Alternatives:
  - Code inspection – even the best programmers can struggle at that
  - Lots of printf()s – not really practical on embedded system



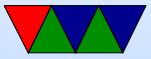
Keil Tools

- Is it the best debugging environment ever?
  No, but it's not the worst either
- As an aside, why does Keil give away their tools for "free"?



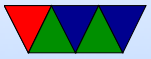
How does it work?

- A bit hardware dependent
- Usually a mechanism that allows halting running system and reading out all of the hardware state (most importantly the program counter, stack, registers, and memory contents)



Setting breakpoints

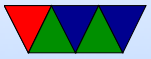
- Set a location in code where you want execution to stop
- Maybe you want to stop after you've done initialization so you can check the register contents match what they should be



More info on the Program Counter

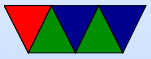
```
8000010      4990  ldr r1,[pc,#256]
8000012      6ccb  ldr r3,[r1,#76]
8000014      f043
8000016      0302  orr r3,r3,#2
8000018      69c4  str r3,[r1,#76]
...
8000110      40021000 (constant)
```

- PC is at 80000010, so loads the ldr instruction there from memory loads the memory value located at address of pc+256, stores in r1 instruction done, increments PC to 8000012
- PC is at 80000012, so loads the ldr instruction from memory and decodes loads the memory value located



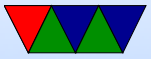
at address of r1+76, stores in r3 instruction done, increments (this insn was 2 bytes so to 8000016)

- PC is at 800000106, so loads the orr instruction from memory and decodes orrs the value in r3 with constant #3, stores in r3 instruction done, increments PC to 8000018



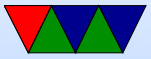
Looking at register contents

- This is when knowing assembly language can be useful



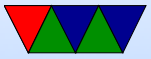
Setting watchpoints

- Can stop execution if value at a memory address is changed
- Useful if a variable ends up with a value and you have no idea how it's getting that value. Set a watchpoint and hit "run" until the value changes



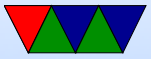
Single Stepping

- Once you have a breakpoint near the beginning of where things go wrong, you can single-step through each line of code (or assembly) to make sure the program is executing properly



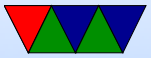
Callstack / Stack backtraces

- When you stop, the debugger can look on the stack and show you all functions that were called to get where your code is
- Often it can show you the calling parameters too, though sometimes these get optimized away/over-written so be careful of that



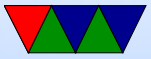
Processor Exceptions

- If your code does something invalid it can trigger an exception / interrupt that might unexpectedly jump your code to a handler
- Since we have no operating system you won't get notified of this, you'll just find your code stuck at one of the exception handler stubs
- A few that are easier to hit
  - Invalid instruction
  - Invalid memory access



More ARM/Thumb2 instructions/features

- These aren't needed for the lab, but can be useful to know about
- As always, working code is much more important than clever code

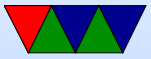


Conditional Execution

- Note: this is an advanced/obscure technique I am mentioning for completeness, you don't have to know how to use it for this class
- On ARM32 could prefix *any* instruction with condition flags, i.e.

```
addeq   r1,r2,r3    ; only does the add if Z=1
addmi   r1,r2,r3    ; only does the add if N=1
```

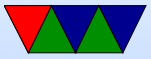
- On Thumb2 they re-used these encoding bits (the left 4 bits of the instruction) to implement the Thumb-2 set,



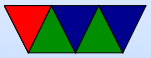
so you cannot do this anymore.

- There is a hack called IT, where you can do up to four instructions. The condition has to be the same (Then) or the opposite (Else)

```
itete cc
    addcc r1,r2 @ then
    addcs r1,r2 @ else
    addcc r1,r2 @ then
    addcs r1,r2 @ else
```

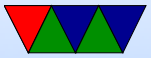


Other Obscure THUMB2 Instructions



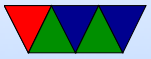
CBZ/CBNZ

- `CBZ r0, label`
- Special compare-and branch instruction, not change flags
- Can only branch forward



TBB

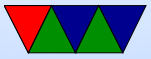
- `TBB [r0, r1]`
- Table-branch byte
- r0 is pointer to table, r1 is offset into table
- Can be used to make switch statements/jump tables



Saturating Arithmetic

Used in some algorithms, saturate at high or low value rather than wrapping around.

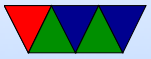
- QADD
- QSUB
- SSAT
- USAT



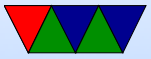
Parallel Arithmetic

Can split 32-bit register up into 16 or 8 bit chunks and do arithmetic in parallel (which is faster)

- ADD16
- SUB16
- ADD8
- SUB8
- ASX
- SAX
- USAD8

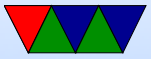


- USADA8
- SEL (select) – comparisons when doing parallel match can set special multiple-GE (greater/equal) register in the ASPR and bytes can be selected based on this



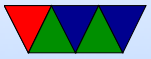
Count Leading Zeros

- CLZ – count leading zeros



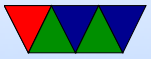
Sign/Zero Extension

- SXTB – sign extend a byte
- SXTH – sign extend a halfword
- UXTB – zero extend a byte
- UXTH – zero extend a halfword

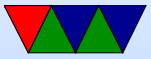


Sign/Zero Extension with add (Cortex-M4/DSP)

- sxtab rd, rn, rm, rotation
  rotate rm right by multiple of 8, extract bottom 7 bits, sign extend, add to rn, store in rd
- SXTAB – sign extend a byte
- SXTAB16 –
- SXTAH – sign extend a halfword
- UXTAB – zero extend a byte
- UXTAB16 – zero extend a byte

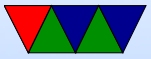


- UXTAH – zero extend a halfword



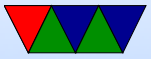
Pack Instruction (Cortex-M4/DSP)

- pkhbt rd, rn, rm, lsl value
  combine bits 15:0 of rn with bits 31:16 of shifted value from rm
- pkhtb rd, rn, rm, asr value
  combine bits 31:16 of rn with bits 15:0 of shifted value from rm
- PKHTB – pack halfword top and bottom



Bitfield

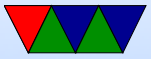
- BFC rd, #lsb, #width – bitfield clear
- BFI rd, rn, #lsb,#width – bitfield insert
- SBFX rd,rn, #lsb,#width – signed bitfield extract
- UBFX rd, rn, #lsb,#width– unsigned bitfield extract



Bit/Byte Reversing

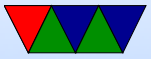
Useful for handling endianess, network packets, etc.

- RBIT – reverse bit oder
- REV – reverse byte order
- REV16 – reverse byte order halfword



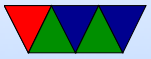
Nop

- nop – no-operation
- Why is this useful?
- Padding code?
- Temporarily commenting out code w/o changing size?
- Delays that do nothing?
- How would you implement this? Are there instructions you can think of that would do nothing? add r0,r0,#0?



Sleep

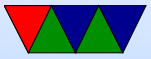
- What should your device do when not busy?
- Enter a busy infinite loop?
- Wouldn't it be better if you could let the chip know and it could go to sleep / enter low-power mode?
- These do that, and an interrupt (such as a timer or hardware change) can wake the processor and start running code again
  - wfi – wait for interrupt
  - wfe – wait for exception



System Registers

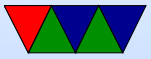
To configure the processor there are special registers beyond the standard ones. These instructions let you copy values into/out-of these special registers.

- MSR – move from system register
- MRS – move to system register



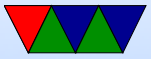
System Calls

- svc
- We don't deal with it in this class, but if you have an operating system you need some way to have your code raise an exception to change security level or let upper layers of software know you need something



Vector/FP/NEON/DSP

- The Cortex-M4 can handle floating point values
- We will discuss this later in class
- There's a whole separate set of instructions/registers for this



Non-ARM instructions you might see someday

- branch delay slots
- VLIW (Dsp, GPU, itanium)
- Register windowing
- Really CISC instructions