ECE 271 – Microcomputer Architecture and Applications
Lecture 7

Vince Weaver

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

vincent.weaver@maine.edu

12 February 2019

Announcements

- Read Chapter 5
- President's Day Monday: People in Monday lab should attend another lab if possible (Wednesday is often a good choice)
- In the unlikely event lab is cancelled due to snow, check in your git code normal time, and then show up at an alternate lab to get checked off.

General Lab Update

- Note: Keil compiler old. Can't use 0b1000 constants, can't declare in middle
- C is old. There are various versions and standards, and Keil implements an older one than gcc

LCD Lab Update

- Almost always the issue is you are setting one of the register fields wrong
- It is tough that everything has to be perfect for it to work, making debugging hard
- Sadly real-world programming can be like this

Keypad Lab

- Why I split the code up in 3 chunks
- How to debug.
  - ○ Use the debugger.
  - ○ Use a multi-meter?
  - ○ Print to the LCD
- Reminder in C of how strings work.

```
char s[7];  // 0..6, room for nul
s[6]=0;
s[0]=(!!(GPIOA->IDR&(1<<2)))+'0';
LCD_Display_String(s);
```

- Why can't you

```
LCD_Display_String(keypad_scan());
```

Moves

- mov r0,r1
- movn r0,r1

Loading a Constant

- mov r0,#8 – constant, up to 8 bits
- movw r0,#imm16 – move 16 bits to bottom of register (and clear top)
- movt r0,#imm16 – move 16 bits to top of register (leave bottom)
- ldr r0,=imm32 – old fashioned way, using global table Usually a PC relative load

Load

- ldr r0, [r1] – load 32-bit value from pointer r1 into r0

- ldr r0, [r1,#4] – pre-index, load 32-bit value from pointer (r1+4) into r0
  useful for structs, things like
  `GPIOA->ODR`

- ldr r0, [r1,#4]! – pre index with update. load 32-bit value from pointer (r1+4) put in r0. Then add 4 to r1 and update r1.

- ldr r0, [r1],#4 – post-index. Load 32-bit value from pointer r1 into r0. Then add 4 to r1 and store in r1.

Load Different Sizes

- What if you don't want to load 32-bits?
- ldrb – load byte into register
- ldrh – load half-word (16-bits)
- ldrsb – load signed byte (sign-extend to fill 32-bits)
- ldrsh – load signed half-word (sign-extend)

Stores

- str r0,[r1] – store 32-bit value in r0 to memory pointed to by r1
- strb
- strh
- any need for sign extend?
- can do same addressing modes, i.e. post-index, etc

PC Relative Load/Stores

- Remember that r15 is PC
- This is how the syntax

```
    ldr =0xdeadbeef

turns into

1005c:       480b            ldr     r0, [pc, #44]  ; (1008c
    ......
1008c:          0xdeadbeef
```

Load/Store Multiple

- Powerful
- STMIA rn!, register list
- for example

```
stmia r13, {r0,r1,r2,r3}
```

- if ! then writeback, meaning the address of the final thing is put into the register (like a stack)
- What happens if LR is in STM and then PC is in LDM?

- LDM the opposite
- IA, IB (increment before / increment after)
- DA, DB (decrement before / decrement after)
- can use PUSH/POP to do the same but assume r13
- PUSH/POP
- returning from a function trick?

```
push {r0,r1,r2,lr}
...
pop  {r0,r1,r2,pc}
```