ECE 271 – Microcomputer Architecture and Applications Lecture 8

Vince Weaver

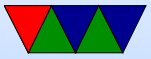
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

10 February 2022

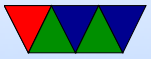
Announcements

- Read Chapters 6 and 7
- Gitlab directions/video have been posted



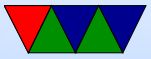
General Lab Update

- I hear that Lab #3 is taking longer than expected
- I will discuss with the TA and see if we can come up with some sort of solution

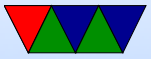


Loading a Constant

- `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



- `adr r0,varname` – load address of a variable/label, may only work if in same segment. pseudo instruction, PC relative add

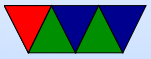


Thumb-2 12-bit immediates

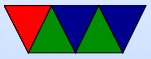
ADD and SUB can have a real 12-bit immediate (0..4095)
Or you can have flexible immediate (ADD and SUB can do this too):

- any constant that can be produced by shifting an 8-bit value left by any number of bits within a 32-bit word
- any constant of the form 0x00XY00XY
- any constant of the form 0xXY00XY00
- any constant of the form 0xXYXYXYXY.

```
top 4 bits 00.00 -- 00000000 00000000 00000000 abcdefgh
```



```
            00.01 -- 00000000 abcdefgh 00000000 abcdefgh
            00.10 -- abcdefgh 00000000 abcdefgh 00000000
            00.11 -- abcdefgh abcdefgh abcdefgh abcdefgh
rotate bottom 7 bits|0x80 right by top 5 bits
            01000 -- 1bcdefgh 00000000 00000000 00000000
             ...
            11111 -- 00000000 00000000 00000001 bcdefgh0
```

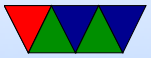


Load/Store Multiple

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

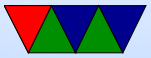
```
stmia r13, {r0,r1,r2,r3}
```

- 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?

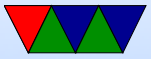


- LDM the opposite
- IA, IB (increment before / increment after)
- DA, DB (decrement before / decrement after)
- can use PUSH/POP to do the same but assume r13
- PUSH/POP
- returning from a function trick?

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

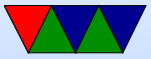


Control Flow



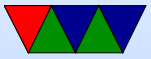
Comparison

- don't need S flag (always update flags)
- `CMP r0, r1` – compare two values, update flags
  same as subtract instruction, but result thrown away
- `CMN r0, r1` – compare negative (same as add)
- `TST r0, r1` – test if bits set
  same as AND, update flags
- `TEQ r0, r1` – test if equal
  same as xor, update flags
- What use is TEQ vs CMP? Doesn't set C or V flags?

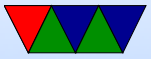


Branches/Jumps

- `B` – (BAL) branch always
- `BEQ/BNE` – branch equal/not-equal – (Z set/clear)
- `BGE/BLT` – signed greater or equal – N set and V set or N clear and V clear
- `BGT/BLE` – Z clear and either N set and V set, or N clear and V set
- `BCS/BCC (BHS/BLO)` – higher or same / lower (unsigned) – (C set/clear)



- `BMI/BPL` – minus/plus (N set/clear)
- `BVS/BVC` – overflow (V set/clear)
- `BHI/BLS` (c set and z clear) – higher or less/same –(c clear or z)

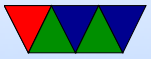


Example Code Translation – If/Then/Elese

```
    if (x==0) {
        y=1;
    }
    else {
        y=5;
    }

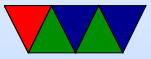
    ldr r0,=x    ; load address of X into r0
    ldr r0,[r0]  ; load value in X into r0
    cmp r0,#0    ; compare with 0
    bne ELSE     ; if not equal, then branch ahead to ELSE
    mov r1,#1    ; load 1 for placing into Y
    ldr r3,=y    ; get address of y in r3
    str r1,[r3]  ; store value to Y
    b   DONE     ; skip ahead to DONE (to avoid else code)
ELSE
    mov r1,#5    ; load 5 into Y
    ldr r3,=y    ; turns to pc-relative ldr
    str r1,[r3]  ; store out to Y
DONE

x
```

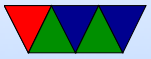


```
    .word 0
y
    .word 0
```

- The label names are arbitrary, you can pick ones that make sense for you. They don't have any special meaning (the assembler will just convert them to numbers)
- When you branch to a label, the assembler turns this into a jump offset.
  So it will really turn into something like "bne pc+X" where X is a positive or negative offset that will be added to the program counter, which will redirect execution to the new instruction.



- If a branch is not taken, it just "falls through" to the next instruction in order.
- Could we optimize this code? Hoist code before? Move store after? Use `ldrs` instead of compare? Conditional execution?



Example Code Translation – For Loop

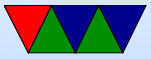
```
    for(i=0;i<100;i++) {
    }

    mov r0,#0        ; init loop index
LOOP
    cmp r0,#100      ; compare to limit
    bge DONE         ; if above or equal, done

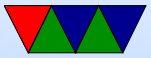
    ...          ; do whatever code in the loop

    add r0,r0,#1     ; increment index
    b   LOOP         ; branch always back to repeat loop
DONE
```

- again, the labels are arbitrary
- The compiler (if you do gcc -S to see assembly output) will change this to a while loop.



- Why? Maybe works better for branch predictor?



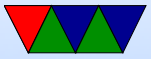
Example Code Translation – While Loop

```
    int x=0;

    while(x<100) {
        x++;
    }
```

```
    mov r0,#0        ; init loop index
    b   CHECK        ; skip ahead to condition check
LOOP
    ...
    add r0,r0,#1

CHECK
    cmp r0,#100      ; compare to see if at end
    blt LOOP         ; if less than equal, branch back to LOOP
```

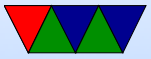


Example Code Translation – Do - While Loop

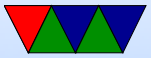
```
    int x=0;

    do {
        x++;
    } while(x<100);

    mov r0,#0
            ; this is just like while loop
            ; but no branch, so always executes once
LOOP
    ...
CHECK
    cmp r0,#100
    blt LOOP
```

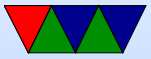


How would you do an infinite loop?



Lab #4 Preview

- Redo Lab#1, but in Assembly language
- ```
  RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
  ```
- ```
  #define PERIPH_BASE      ((uint32_t)0x40000000)
  #define AHB1PERIPH_BASE (PERIPH_BASE + 0x00020000)
  #define RCC_BASE         (AHB1PERIPH_BASE + 0x1000)
  #define RCC              ((RCC_TypeDef *)(RCC_BASE))
  ```
- ```
  ldr     r1,=RCC_BASE
  ldr     r3,[r1,#RCC_AHB2ENR]
  orr     r3,#RCC_AHB2ENR_GPIOBEN
  str     r3,[r1,#RCC_AHB2ENR]
  ```


  ```