ECE 271 – Microcomputer Architecture and Applications Lecture 8

Vince Weaver

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

vincent.weaver@maine.edu

14 February 2019

Announcements

- Read Chapters 6 and 7
- Wednesday was a snow day. People with Wednesday lab try to check in code as soon as possible and get lab checked off when you can.
  Also if you need a keypad let me know.
- If you want/need an additional breadboard, you can pick one up in lab.

General Lab Update

- PAPI and good git commit anecdote

Control Flow

Branches/Jumps

- B – branch always
- BEQ/BNE – branch equal/not-equal – (Z set/clear)
- BCS/BCC (BHS/BLO) – higher or same / lower – (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)
- BGE – greater or equal – N set and V set or N clear and

V claer

BLT

- BGT – Z clear and either N set and V set, or N clear and V set

  BLE

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
- TST r0, r1 – test if bits set
  same as and, update flags
- TEQ – test if equal
  same as xor, update flags
- What use is TEQ vs CMP? Doesn't set C or V flags?

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
    addcs r1,r2
    addcc r1,r2
    addcs r1,r2
```

Other Obscure Instructions

Sign/Zero Extension

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

Bit/Byte Reversing

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

Nop / Sleep

- nop – no-operation
- wfi – wait for interrupt
- wfe – wait for exception

System Registers

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

Vector/FP/NEON/DSP

- We might discuss this later in class

Example Code Translation – If/Then/Elese

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

```
    ldr r0,x     ; load X into r0
    cmp r0,#0    ; compare with 0
    bne ELSE     ; if not equal, then branch ahead to ELSE
    mov r1,#1    ; load 1 into Y
    adr r3,y     ; pseudo-insn, 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
    adr r3,y      ; turns to pc-relative ldr
    str r1,[r3] ; store out to Y


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)
- "adr" is a pseudo-instruction. The assembler understands it (load address) but it is not a Thumb-2 instruction. The assembler will convert this to an

actual instruction (in this case probably a LDR with a PC-relative address)

- 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.

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       ; incrememnt 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

```c
    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,#99       ; compare to see if at end
    ble 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,#99
    ble LOOP
```

THUMB differences

- 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)

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

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 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
```