ECE 271 – Microcomputer Architecture and Applications Lecture 7

Vince Weaver

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

vincent.weaver@maine.edu

8 February 2022

Announcements

- Read Chapter 5

Subtract with Borrow (8-bit example)

Say we want to subtract 16-bit 0x8000 - 0x0001 but we only have 8-bit wide ALU. You can chain together 8-bit subtracts using the carry (borrow) flag.

Use `subs` to subtract the low bytes and update flags. Carry is 1 if no borrow, 0 if there is a borrow.

$$\begin{array}{ccccccccccc}
 & & 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 \\
 & - & 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 1 \\
\hline
\text{0 (borrow) (C)} & \leftarrow & 1 & 1 & 1 & 1 & & 1 & 1 & 1 & 1
\end{array}$$

Subtract the upper bytes using `sbc` which uses the existing carry flag (from the previous `subs`). Note the *inverse* of the carry is subtracted off.

$$\begin{array}{ccccccccccc}
1 & 0 & 0 & 0 & \quad & 0 & 0 & 0 & 0 & & \\
0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 & & \\
\hline
1 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 & & \\
 & & & & & & & \text{-} & 1 & \leftarrow & \text{inverse of (C)} \\
\hline
0 & 1 & 1 & 1 & & 1 & 1 & 1 & 1 & &
\end{array}$$

Why use inverted borrow? It makes the ALU easier if using twos complement. To SUB you just invert the second term, add them, then add 1. To SBC you add in the carry instead of the 1.

Barrel Shifts

- For ALU instructions, and some others.
- The third argument can optionally be shifted by a constant

  -
    ```
    add r1,r2,r3 LSR #2
    r1=r2+(r3<<2)
    ```
  - LSL, LSR, ASR, ROR, RRX
  - on arm32 could have a 4th register instead of a constant as shift amount

Multiply

- How big is your result? 32bit * 32bit has potentially 64bit result
  What happens to the high bits?
- MUL RD,RN,RM = rd=rn*rm (signed)
- UMUL RD,rn,rm = unsigned
- MLA rd,rn,rm,ra = multiply/add rd=rn*rm+ra
- MLS rd,rn,rm,ra = multiply/sub rd=rn*rm-ra

- UMULL rdlo,rdhi,rm,rn
- MULL rdlo,rdhi,rm,rn

Faster Multiply

- Other ways to multiply.
- Can multiply by two by adding
  `add r1,r2,r2`; r1=r2*2
- Can multiply by powers of two by just doing shift left
  `asl r1,r2,#2`; r1=r2*4
- With ARM barrel-shifters can do shit/add in one instruction. `add r1,r2,r2,asl #2`; r1=r2*5

Division

- `SDIV RD,rn,rm` = Signed divide rd=rn/rm
- `UDIV RD,rn,rm` = Unsigned divide rd=rn/rm
- Even slower than mul. Takes a lot of space, not used often, so some chips leave it off. For example, no divide on early Raspberry Pi
- For powers of two, can right-shift

Remainders

- Division gives you quotient: what if you want remainder?
  - If power of two, can use and with divisor - 1:
    5/4 : R = 5&(4-1)
    This is just masking off the bottom bits that get shifted off.
  - If have multiply instruction, R = original - (Q * divisor):
    5/4 : Q = 5 - (1*4)

How would you divide if not available in Hardware?

- For constant devisiors, can multiply by reciprocal. x/10 = x* (1/10). Have to set up the value and rounding right, but is often faster than dividing.
- Alternately, do shift and subtract, like long division.

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

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
```

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
```

Load/Stores to Memory

- ldr r0, [r1] – load 32-bit value from pointer r1 into r0
- Can you do something like
  ```
  ldr r0, variable
  ```
  sadly no
- Need to first
  ```
  ldr r1, =variable
  ```

Using registers as pointers

- The CPU makes no distinction between pointers and integer values
- You can use any register as either a pointer or integer
- When programming in assembly language it's up to you to keep track of what type of value is in each register

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)

Load with Offset (register)

- `ldr r0, [r1,r2]` – pre-index, load 32-bit value from pointer (r1+r2) into r0
- Index a byte array

```
char a[10];
int x=5;
r3=a[x];

ldr r1,=a
mov r2,#5
ldrb    r3,[r1,r2]
```

Load with Offset (constant)

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

```
r2=GPIOA->ODR

ldr r1,=GPIOA_BASE
ldr r2,[r1,#GPIO_ODR]
```

Load with Offset (shift)

- `ldr r0, [r1,r2,lsl #2]` – pre-index
  Load 32-bit value from pointer (r1+(r2*4)) into r0
- Index an integer array

```
int a[10];
int x=5;
r3=a[x];

ldr r1,=a
mov r2,#5
ldr r3,[r1,r2,lsl \#2]
```

Load with Offset (pre-index update)

- `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.

Load with Offset (post-index update)

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

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 as loads, i.e. post-index, etc

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}
```

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
```