CS 237 Meeting 25 — 11/7/12

Announcements

1. TA applications.

A Whole New Assembly Language

1. Our final project this semester will involve designing a digital circuit to implement a hypothetical machine language in the same way that the circuits presented recently in class and described in our text implement a subset of the MIPS machine language.

2. The machine language we will implement is a small subset of a hypothetical architecture Duane Bailey designed for 237 a few years ago. It is based on a real architecture known as ARM (which once stood for Acorn RISC Machine but now stands for Advanced RISC Machine).

3. Today, I will give you a brief introduction to the architecture we will be implementing. Duane named his machine "WARM" for Williams Academic RISC Machine. I was thinking of calling our subset of WARM LukeWARM, but Duane suggested TEPID. Out of respect for Duane (and because I could think of a corny interpretation for TEPID but not for LukeWARM), we will call our machine TEPID.

4. ARM[1], WARM, and TEPID all have a lot in common with MIPS.

   - They all are 32 bit machines. To simplify implementation, WARM and TEPID make the 32-bit word the fundamental unit of memory. Their memories are word addressable so that the first word is word 0 and the second is word 1 (rather than word 4 as in MIPS).
   - They are all RISC architectures. WARM and TEPID are in some sense extreme examples since they were stripped down to make for tractable final projects.
   - They are all load/store architectures. The arithmetic instructions of all of these machines can only read and write values in registers. Separate load and store instructions are used to move data between memory and registers.

5. So how do they differ? First there are a few simple differences.

   - WARM and TEPID have a 24-bit limitation on memory addresses and therefore a 16 megabyte limit on the size of the memories they can access (Surprise! Logisim has a 16 megabyte limit on the memories it will simulate.)
   - WARM and ARM only have 16 registers instead of 32.
     - They are named r0 through r15 (no dollar signs required).
     - r0 is not equal to 0. Instead, it is typically used for function return values.
     - r0 or r1 through r3 are typically used for function parameters.
     - r4 through r10 are typically used for local variables.
     - r13 is usually used as the stack pointer.
     - r14 is the link register (equivalent to $ra in MIPS).
     - r15 holds the program counter. This is very different from MIPS! Any instruction (including loads or adds) that sets r15 has the effect of taking a branch.

6. There are also several more interesting differences:

7. Conditional execution in WARM/TEPID depends on a set of bits know as condition codes or condition code registers. They record summary information about a recent ALU output:

   Z ZERO - The ALU output was zero.

   N NEGATIVE - The ALU output was negative (the high order bit was 1).

   C CARRY - There was a carry out of the high order digit as part of the last ALU operation.

   V OVERFLOW - The last operation involved an overflow (the carry out was different from the high order bit).

[1] To tell you the truth, I don't really know the ARM architecture very well at this point. With a little luck, I will learn more and we will discuss it in more detail later in the semester.

8. Every TEPID/WARM instruction begins with a 4 bit field that determines:

   (a) Whether the instruction should be executed based on the current values of the condition code bits.

   (b) Whether if the instruction executes, the condition code bits should be updated to reflect the result produced by the ALU during its execution.

Condition					
31	30	29	28	Ending	Meaning
0	0	0	–	–	always
0	0	1	–	`nv`	never
0	1	0	–	`eq`	equal (`Z`= 1)
0	1	1	–	`ne`	not equal (`Z`= 0)
1	0	0	–	`lt`	less than (`N`≠`V`)
1	0	1	–	`le`	less or equal ((`Z`= 1) or (`N`≠`V`))
1	1	0	–	`ge`	greater or equal (`N`=`V`)
1	1	1	–	`gt`	greater than ((`Z`= 0) and (`N`=`V`))
–	–	–	0	–	don't set the condition codes
–	–	–	1	`s`	set condition codes

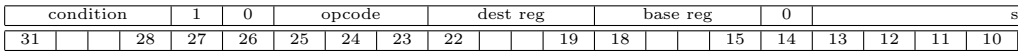
9. When applied to the branch instruction (b), this works as one might expect:

   beq only branches if the previous arithmetic operation with an s suffix (which is assumed for the cmp (compare) instruction produced 0.

10. It can, however, be applied to any instruction

    addeq only adds if the previous instruction that set the condition code set Z = 0.

11. TEPID has two instructions named LDR (load register) and STR (store register) that act quite a bit like the lw and sw instructions in the MIPS ILA.



12. There is also a ADR instruction that uses the same format but put the address of the memory argument into the destination register instead of the contents.

13. TEPID has 3-operand arithmetic instructions including add, sub, and, and orr. The destination and the first source operands must be specified as 4-bit register numbers. There are two (or three) options for the other source operand.

    - An immediate operand can be provided. This is stored in the last 14 bits of the instruction, but it is not a simple 14 bit signed number. Instead, the first 5 bits are treated as a base 2 exponent and the last 9 bits as a constant.
    - The second source operand can also be specified as the shifted value of a third register. 4 bits are used for the register number, 5 bits for the shift amount, and 2 bits for the shift operation (shift left, logical or arithmetic shift right, rotate).

14. Since there is no $0, there is a mov instruction that ignores its first source operand.

15. To deal with the limitations of the immediate mode, there is also a mvn which moves the complement of its operand.

16. There are also cmp and tst instructions to set the condition code without changing any register.

17. Consider the following version of GCD:

```
int gcd(int a, int b)
// pre: 0 <= a <= b
// post: return greatest common divisor of both a and b
{
    if (a == 0) return b;       // b divides 0 and b
    if (a > b) return gcd(b,a); // swap to meet pre-condition
```

```
        return gcd(b-a,a);          // b%a is < b
    }
```

18. Here is a TEPID program that implements this function:

```
main:   swi     #2
mov     r1,r0
        swi     #2
        mov     r2,r0
        bl      gcd
        swi     #4
        swi     #0


gcd:    sub     sp,sp,#1
        str     lr, [sp, #0]
        cmp     r1,#0
        bne     else1
        mov     r0,r2
        b       return
else1:  cmp     r1,r2
        ble     else2
        mov     r0,r2
        mov     r2,r1
        mov     r1,r0
        bl      gcd
        b       return
else2:  sub     r2,r2,r1
        bl      gcd
return: add     sp,sp,#1
        ldr     pc,[sp, #0]
```