Instruction Set

Instruction set design puzzle

There were a number of design choices made to maximize versatility of the two-address instruction set while keeping the relay count as low as possible. The starting arbitrary decision is that the B field is the one which can be modified by self modified code. This implies:

- The B field needs to hold the destination address for jump instructions, otherwise we can't implement subroutine linkage.
  - But this means that the A field needs to be the one with the complementer (controlled by the "com" bit), so we can test the A argument for zero (by setting up the ALU to negate the A operand) and conditionally jump all in one instruction.
  - This also implies that the B operand of the ALU needs the zero-MUX (controlled by the "ben" bit), so that it can be masked off for this comparison.

But which field is the result address? It turns out that we need both.

- We need to be able to write to the B address in order to implement a data move instruction (the "st"ore instruction), since we need to be able to mask off the operand we are storing to (to pass the other argument unchanged), and the B side has the zero-MUX.
- But we need to be able to write to the A address in order to implement the single-cycle loop instructions (since the B address has the jump target): incjne and incjeq.
- More subtly, in order to support indirection (for support of pointers) with self-modified code, we need to be able to write both to the field with the modifying address for write indirection, and to the other field for read indirection. We can modify the B field of the store instruction for write indirection, and ideally we should have a complementary load instruction (which moves data from B to A) for read indirection. We don't have it, but we can make do with an add instruction which writes to A by requiring that the target address be pre-cleared.
  - Note that we don't ever have to write to both the A and B fields at the same time, so the memory system does not require multiple write ports- just a single port fed by a MUX.
  - Four more relays and an added control bit would be needed to provide a zero-MUX for the A operand to implement a load instruction for read indirection.

Other concerns:

- Shift and rotate left can be implemented by adding an operand to itself.
- Shift and rotate right require another 9-bit 2:1 MUX at the expense of 5 relays (we could have left these instructions out, but didn't).

Instruction Set

Here is a printable Reference card.pdf

The B field is always given on the right with these assembly mnemonics. This is the field which is subject to change by self-modifying code and is usually the result address.

If a two-operand instruction has a suffix of "to", as in "addto", it means the result is placed in the right-side operand (in the B field). If the "to" is missing, it means the result is stored in the left side (in the A field). There is one exception to this rule: "st" (store) saves its result in the right side even though it does not have the "to" suffix.

A hash (or pound sign) in front of the left operand signifies an immediate value. The A field is treated as containing the data itself instead of containing an address and the 'imm' bit of the instruction will be set.

There are a few instructions which store their result on the left (which means they have the "wra" bit set). These include add, rsb, incjne and incjeq. Add and rsb seem redundant since the addto and rsbto instructions exist, but they are needed to implement read indirection from pointers. A real "ld" (load) instruction which saves on the left would be better for this, but would require more relays and more control bits to implement.

Here are some of the possible instructions:

4010FF00 nop

No operation. Carry flag is preserved (ALU is set up to add 0xFF + 0x00 + carry flag: so if carry was set, it will be set again when this instruction completes).

C810FF00 halt

The clock is stopped until user presses step or run. Carry flag is preserved.

00000000 clc

Clear the carry flag.

4020FF00 stc

Set the carry flag.

4018FFbb jmp bb

Unconditional jump to bb. Carry flag is preserved. PC = bb.

8408aabb jsr aa, bb

Jump to subroutine located at bb. The return address is placed in memory location aa (which could be a jmp instruction located at the end of the subroutine). Carry is cleared.

0061aabb jmi aa, bb

0061aabb jlt aa, bb

Jump to bb if signed contents of memory location aa is negative (minus, or less than zero). If [aa].7 == 1 then PC = bb else PC = PC + 1. Carry flag is set if [aa] is zero.

0069aabb jpl aa, bb

0069aabb jge aa, bb

Jump to bb if signed contents of memory location aa is positive or zero (plus, or greater than or equal to zero). If [aa].7 == 0 then PC = bb else PC = PC + 1. Carry flag is set if [aa] is zero.

0062aabb jeq aa, bb

Jump to bb if contents of memory location aa is equal to zero. If [aa] == 0 then PC = bb else PC = PC + 1. Carry flag is set if [aa] is zero.

006Aaabb jne aa, bb

Jump to bb if contents of memory location aa is not equal to zero. If [aa] != 0 then PC = bb else PC = PC + 1. Carry flag is set if [aa] is zero.

0063aabb jle aa, bb

Jump to bb if signed contents of memory location aa is less than or equal to zero. If [aa] <= 0 then PC = bb else PC = PC + 1. Carry flag is set if [aa] is zero.

006Baabb jgt aa, bb

Jump to bb if signed contents of memory location aa is greater than zero. If [aa] > 0 then PC = bb else PC = PC + 1. Carry flag is set if [aa] is zero.

006400bb jcc bb

006400bb jlo bb

Jump to bb if carry flag is clear (or if unsigned left argument is lower than unsigned right argument of previous subtraction instruction). If C == 0 then PC = bb else PC = PC + 1.

006C00bb jcs bb

006C00bb jhs bb

Jump to aa if carry flag is set (or if unsigned left argument is higher or same as unsigned right argument of previous subtraction instruction). If C == 1 then PC = aa else PC = PC + 1.

0066aabb jls aa, bb

Jump to bb if unsigned left argument is lower or same as unsigned right argument of previous subtraction instruction. The result of the subtraction instruction must be available in memory location aa. If C ==0 or [aa] == 0 then PC = bb else PC = PC + 1.

006Eaabb jhi aa, bb

Jump to bb if unsigned left argument is higher than unsigned right argument of previous subtraction instruction. The result of the subtraction instruction must be available in memory location aa. If C == 1 and [aa] != 0 then PC = bb else PC = PC + 1.

020Aaabb je aa, bb

Jump to bb if left argument is even. If [aa].0 == 0 then PC = bb else PC = PC + 1.

0202aabb jo aa, bb

Jump to bb if unsigned left argument is odd. If [aa].0 == 1 then PC = bb else PC = PC + 1.

802Aaabb incjne aa, bb

Increment [aa] and jump if there is no carry from this. If [aa] == 255 then PC = PC + 1 else PC = bb. [aa] = [aa] + 1.

8022aabb incjeq aa, bb

Increment [aa] and jump if there is a carry. If [aa] == 255 then PC = bb else PC = PC + 1. [aa] = [aa] + 1.

0800aabb st aa, bb

Store contents of memory location aa into memory location bb. Carry is cleared.

4800aabb st #aa, bb

Store constant from instruction field aa into memory location bb. Carry is cleared.

1000aa00 out aa

Store contents of memory location into output register.

5000aa00 out #aa

Store immediate data into output register.

9800aa00 outc aa

Write contents of memory location aa to serial port.

B800aa00 outc #aa

Write immediate data aa to serial port.

680000bb in bb

Read input port and save value in memory location bb.

E80000bb inwait bb

Pause CPU. When inputs change, resume CPU, read new input and save value in memory location bb. If instead one of the keypad keys is pressed, resume CPU and place the keypad hex code into bb (the hex keys give codes 0x00 - 0x0F and the others give codes in the range 0x10 - 0x17). Also, if there is a key-press on the console-serial the CPU is resumed and the character is placed into bb.

480000bb clr bb

Clear memory location bb. [bb] = 0.

8080aabb add aa, bb

Add [bb] to [aa]: [aa] = [aa] + [bb]

0880aabb addto aa, bb

Add contents of memory location aa to memory location bb. [bb] = [aa] + [bb]. Carry is modified.

4880aabb addto #aa, bb

Add constant located in aa field of instruction to memory location bb. Carry is modified. [bb] = aa + [bb].

488001bb inc bb

Increment memory location bb. [bb] = [bb] + 1.

48E001bb dec bb

Decrement memory location bb. [bb] = [bb] - 1.

0890aabb adcto aa, bb

Add memory location aa and carry flag to memory location bb. Carry is modified. [bb] = [aa] + [bb] + c.

4890aabb adcto #aa, bb

Add constant located in aa field of instruction and carry flag to memory location bb. Carry is modified.

0880bbbb lsl bb

Logical shift left memory location bb. Zero is shifted into bit 0. Bit 7 is saved in the carry flag.

08A0bbbb lslo bb

Shift memory location bb left. One is shifted into bit 0. Bit 7 is saved in the carry flag.

0A00bbbb lsr bb

Logical shift rights memory location bb. Zero is shifted into bit 7. Bit 0 is saved in the carry flag.

0A20bbbb lsro bb

Shift memory location bb right. One is shifted into bit 7. Bit 0 is saved in the carry flag.

0A00bbbb lsrto bb

Logical shift rights memory location aa and save the result in memory location bb. Zero is shifted into bit 7. Bit 0 is saved in the carry flag.

0A20bbbb lsroto bb

Shift memory location aa right and save the result in memory location bb. One is shifted into bit 7. Bit 0 is saved in the carry flag.

0080bbbb ntoc bb

Move sign bit of memory location bb into carry. This is useful to construct arithmetic (sign-preserving) shift right with this two instruction sequence:

```
ntoc  bb
ror   bb
```

0890bbbb rol bb

Rotate left memory location bb. Carry flag is shifted into bit 0. Bit 7 is saved in the carry flag.

0A10aabb rorto aa, bb

Rotate right memory location aa and save the result in memory location bb.

0A10bbbb ror bb

Rotate right memory location bb.

80E0aabb rsb aa, bb

Reverse subtract: [aa] = [bb] - [aa].

08E0aabb rsbto aa, bb

Subtract memory location aa from memory location bb. Carry is modified. [bb] = [bb] - [aa].

48E0aabb rsbto #aa, bb

Subtract constant located in aa field from memory location bb. [bb] = [bb] - aa.

08D0aabb rsbcto aa, bb

Subtract memory location aa and inverse of carry flag from memory location bb. Carry is modified. [bb] = [bb] - [aa] - ~c.

48D0aabb rsbcto #aa, bb

Subtract constant located in aa field and inverse of carry flag from memory location bb. [bb] = [bb] - aa - ~c.

0980aabb andto aa, bb

Bitwise AND contents of memory location aa into memory location bb. [bb] = [bb] & [aa].

4980aabb andto #aa, bb

Bitwise AND constant in aa field into memory location bb. [bb] = [bb] & aa.

09C0aabb bicto aa, bb

Clear each bit in memory location bb with a corresponding bit which is set in memory location aa: [bb] = [bb] & ~[aa].

49C0aabb bicto #aa, bb

[bb] = [bb] & ~aa.

0860aabb negto aa, bb

Load 2's complement inverse of memory location aa into memory location bb. [bb] = -[aa].

0850aabb ngcto aa, bb

Load 1's complement of memory location aa plus carry into memory location bb. [bb] = ~[aa] + c.

0860bbbb neg bb

2's complement memory location bb. [bb] = -[bb].

0850bbbb ngc bb

Complement and then add carry to location bb. [bb] = ~[bb] + c. Use for multi-precision negate.

0840aabb comto aa, bb

Load 1's complement of memory location aa into memory location bb: [bb] = ~[aa].

0840bbbb com bb

1's complement of memory location aa: [bb] = ~[bb].