CS 51 Homework 4

Jessie Li

October 14, 2024

1.

This circuit computes $Q = \text{Cnd}(CC, ifun)$, returning 1 if the condition codes CC satisfy the condition specified by $ifun$ for `cmovXX` or `jXX`, 0 otherwise. Cnd = 0 if $ifun$ is not 0 ... 6.

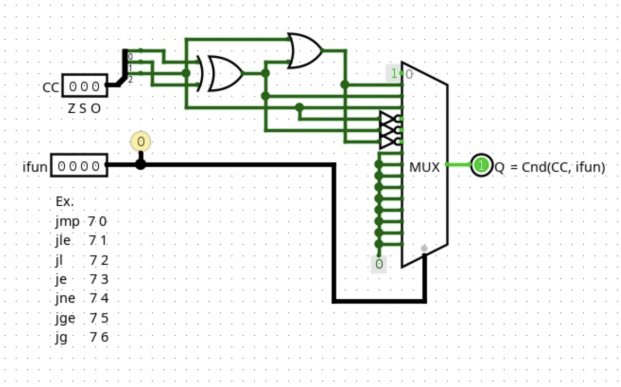

Figure 1: `q1.circ`

Testing

I wrote a shell script `test_q1.sh` to create a test vector `test_q1.txt` of the 56 cases with valid $ifun$ codes (7 $ifun$ codes $\times$ 8 CC combinations), plus two additional test cases with invalid $ifun$ codes ($ifun = 7$ and $ifun = 8$). For all cases, the circuit outputs the correct value of $Q = \text{Cnd}(CC, ifun)$.

2.

addr.	
0x100	30f3
0x102	0f00
0x104	0000
0x106	2031
0x108	4013
0x10a	fdff
0x10c	ffff
0x10e	6031
0x110	7008
0x112	0100
0x114	00

3.

See `q3.ys`. Subroutine `LSB` extracts the least significant byte of the 4-byte word stored at address `target` by calculating `0xff AND *target` and storing the result in `%eax`.

Testing

To test, I ran my program with different values at `target` and confirmed in each case that `%eax` contained the LSB after my subroutine. Some examples:

- `0x00000000`
- `0xffffffff`
- `0x00000001`
- `0x12345678`
- `0xffffff80`
- `0x9c23098e`

4.

See `q4.ys`.

Testing

I reused some test cases from HW2 Q4 and added some for the error/not valid BCD case, since each digit is now 8 bits instead of 4 (assume each digit in hex):

- $output =$ `0x1` $(NotBCD = 0, SixMult = 1)$
  - Zero: `00 00 00 00`

  - Maximum sum of digits: `09 09 09 06`
  - Individual digits not divisible by 3: `01 02 01 02`
  - Random: `00 04 05 00`

- *output* = `0x0` ($NotBCD = 0, SixMult = 0$)
  - Minimum sum of digits: `00 00 00 01`
  - Maximum sum of digits: `09 09 09 09`
  - Divides 3 but not 2: `05 09 00 01`
  - Divides 2 but not 3: `01 02 03 04`
  - Random: `07 00 02 05`

- *output* = `0xE` ($NotBCD = 1, SixMult = 0$)
  - Hex is divisible by 6: `01 07 07 0C`
  - Multiple not BCD: `0E 07 0A 02`
  - Digit > `0xF`: `02 10 01 02`
  - Negative digit: `90 01 01 02`
  - Random: `0A 09 01 0E`

5.

- My program and circuit both convey the same information — whether the input is BCD and a multiple of 6, but my circuit outputs two one-bit values $NotBCD$ and $SixMult$, whereas my program combines this information in one 8-bit *output*. In both, only three outputs are possible: BCD and multiple of 6 (*output* = `0x1`), BCD and not a multiple of 6 (*output* = `0x0`), or error (*output* = `0xE`).

  My approach was conceptually the same for both the circuit and the program. To determine whether the input is a valid BCD, I check if each digit is a BCD. For divisibility by 6, I check if the input is divisible by 2 and 3. Although the high-level logic is the same, there are some differences between my circuit and program implementations, including:

- **Divisibility by 2.** My circuit simply checks if the least significant bit of my input is 0 to determine whether the represented BCD is divisible by 2. Operating on an individual bit is more difficult in assembly. Rather than directly isolating the least significant *bit*, my assembly program determines whether the input is even by computing `LSB(input) AND 0x1`, where `LSB(input)` is the least significant *byte*. If 0, then divisible; otherwise not divisible.

- **Divisibility by 3.** My circuit determines whether the input is a multiple of 3 by summing each digit mod 3, then calculating the sum mod 3. I hardcoded the values of $n \mod 3$ for n from 0 to 15 as inputs to a MUX (with the sum as the selector), knowing that the input to my `mod3` subcircuit would never exceed 9 (for a digit, $2+2+2+2 = 8$ for the sum of each digit mod 3). In contrast, my assembly program sums the digits without mod 3. Then, instead of matching the sum to a hardcoded remainder, I use a loop to subtract 3 until the total is < 3.

- **Error-handling.** If a digit is not a BCD, my assembly program can set *output* to error `0xE` and `halt` early, but my circuit still calculates divisibility by 6 (ignores the result by feeding 0 through an `AND` gate). Similarly, if the input is not divisible by 2, my program will set *output* to `0x0` and stop, whereas my circuit still checks for divisibility by 3.

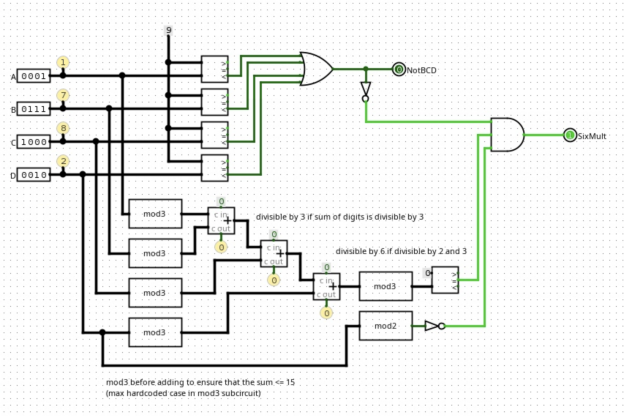

Figure 2: `My circuit from HW2 Q4.`

6.

A.

0x7819	0111 1000 0001 1001
0x829A	1000 0010 1001 1010
0x7819 AND 0x829A	0000 0000 0001 1000
	= 0x0018

B.

0xA281	1010 0010 1000 0001
0xF037	1111 0000 0011 0111
0xA281 OR 0xF037	1111 0010 1011 0111
	= 0xF2B7

C.

NOT((NOT 0x5478) AND (NOT 0xFEED))

= NOT(NOT(0x5478 OR 0xFEED))

= 0x5478 OR 0xFEED

0x5478	0101 0100 0111 1000
0xFEED	1111 1110 1110 1101
0x5478 OR 0xFEED	1111 1110 1111 1101
	= 0xFEFD

D.

0x8814	1000 1000 0001 0100
0x93FA	1001 0011 1111 1010
0x8814 XOR 0x93FA	0001 1011 1110 1110
	= 0x1BEE

E.

0x2871	0010 1000 0111 0001
0xCAFE	1100 1010 1111 1110
0x2871	0010 1000 0111 0001
NOT 0xCAFE	0011 0101 0000 0001
0x2871 NOR (NOT 0xCAFE)	1100 0010 1000 1110
	= 0xC28E