Ellie Makin (erm67)

Computer Design Supervision 1

2017 Paper 5 Question 1

a) (i)

```
module seven_bit_nor(
    input   [6:0] a,
    output  b);

    assign b = !( a[0] || a[1] || a[2] || a[3] || a[4] || a[5] || a[6] );

endmodule
```

(ii)

```
module four_bit_full_adder(
    input   [3:0] d,
    input   [3:0] e,
    output  [4:0] f );

    assign f = d+e;

endmodule
```

(iii)

```
module fsm(
    input   [3:0] g,
    input   load,
    input   clk,
    input   r,
    output  [3:0] h );

    logic [3:0] add_out;
    logic [3:0] mux_out;
    reg [3:0] ff;

    assign h = ff;
    assign add_out = h + 4'd1;

    always @(posedge clk) begin
        if (load)
            mux_out = g;
        else
            mux_out = add_out;
```

```
        if (r)
            ff <= 4'd0;
        else
            ff <= mux_out;
    end
endmodule
```

(b) (i)

Ain	Bin	start	a	b	a'	b'	done'	answer'
21	15	1	X	X	21	15	0	X
21	15	0	21	15	6	15	0	X
21	15	0	6	15	6	9	0	X
21	15	0	6	9	6	3	0	X
21	15	0	6	3	3	3	0	X
21	15	0	3	3	3	0	0	X
21	15	0	3	0	3	0	1	3

(ii) If `Ain` is zero, and `Bin` is non-zero, then the module will fail to terminate, since the final branch of the `if` statement will continually set `b = b - 0`.

2010 Paper 5 Question 2

a) Transistor density has increased at a slower rate since the late 90s, but it is still increasing roughly exponentially, and is projected to continue for a few more years, so the answer is effectively yes, for now. From around 1980-2000, transistor density increased at a rate of about 50% per year, but since then the rate has dropped to closer to 20%. There is a fundamental limit on transistor size at the atomic level, which will ultimately restrict meaningful increase at some point in the near future.

b) Transistor density accounts for a significant part of processor performance, although as transistors become smaller, the wiring on the chip becomes a limiting factor as well. At such small time-scales, the charge time of a wire due to capacitance becomes relevant to the performance, and can be difficult to improve. As Moore's Law starts to slow, the processor architecture has become more important for processor performance as well.

Questions from the recommended text

1.14

Type	CPI	amount(M)	time(ms)
FP	1	50	25
INT	1	110	55
L/S	4	80	160
branch	2	16	16
total			256

1. I'm not sure if I've misunderstood this question, but the time taken up by FP instructions is less than half of the total time, so even if it took zero clock cycles we still couldn't make the program run two times faster. An unrealistic answer, assuming CPI can be negative and fractional, would be CPI $= -4.12$.

2. Assuming CPI can be fractional, CPI $= 0.8$, otherwise CPI $= 1$ is the closest you can get.

3. The new time is 171.2 ms, which is a 84.8 ms ($\approx 33\%$) time save.

Type	CPI	amount(M)	time(ms)
FP	0.6	50	15
INT	0.6	110	33
L/S	2.8	80	112
branch	1.4	16	11.2
total			171.2

2.7

```
# RV32I
  slli  s0, x28, 3        # s0 = i*8
  slli  s1, x29, 3        # s1 = j*8
  add   s0, x10, s0       # s0 = &A[i]
  add   s1, x10, s1       # s1 = &A[j]
  lw    s2, 0(s0)         # s2 = low-order word of A[i]
  lw    s3, 4(s0)         # s3 = high-order word of A[i]
  lw    s4, 0(s1)         # s4 = low-order word of A[j]
  lw    s5, 4(s1)         # s5 = high-order word of A[j]
  add   s6, s2, s4        # add low-order words
```

```
  bgeu  s6, s2, no_carry  # branch if no carry (assuming unsigned)
  addi  s3, s3, 1         # add the carry bit
no_carry:
  add   s7, s3, s5        # add high-order words
  addi  s0, x11, 64       # s0 = pointer to B[8]
  sw    s6, 0(s0)         # store low-order word
  sw    s7, 4(s0)         # store high-order word

# RV64I
# I've slightly optimised the order of the instructions here
# to avoid load-use data hazards when pipelining
  slli  s0, x28, 3        # s0 = i*8
  add   s0, x10, s0       # s0 = &A[i]
  ld    s2, 0(s0)         # load A[i]
  slli  s1, x29, 3        # s1 = j*8
  add   s1, x10, s1       # s1 = pointer to A[j]
  ld    s3, 0(s1)         # load A[j]
  addi  s5, x11, 64       # s5 = pointer to B[8]
  add   s4, s2, s3        # s4 = A[i] + A[j]
  sd    s4, 0(s5)         # store result
```

2.12

Regrouping the bits to align with the fields:

`0000000 00001 00001 000 00001 0110011`

The lowest 7 bits tell us this is an R-type instruction performing an ALU operation, such as `ADD`, `XOR`, `SLL`. The `funct3` and `funct7` fields then specify this instruction as an `ADD` instruction. Finally the `rd`, `rs1` and `rs2` fields tell us the source and destination registers are all `x1`. This gives us the following instruction:

`add x1, x1, x1`

Extra work

2016 Paper 5 Question 1

a) The `mystery` module is an implementation of a stack, of specified width and height (depth). `head` is the index just after the top value on the stack. When the `op` input is `opIn`, the value of `dataIn` is pushed onto the stack, and when it is `opOut`, the top value is popped from the stack, and fed to `dataOut`. For `opNone`, the stack remains the same. After any operation, `dataOut` always returns the top value on the stack. The stack grows from zero upwards until it is full. If an attempt is made to push when the stack is full, or pop when the stack is empty, the `error` output will be 1, and the stack will remain unchanged.

b) (The rest of this question hasn't been covered in the course yet).

2.4

C equivalent:

```
B[g] = A[f] + A[f+1]
```

Reasoning:

```
slli  x30, x5, 3    # x30 = f*8
add   x30, x10, x30 # x30 = &A[f]
slli  x31, x6, 3    # x31 = g*8
addi  x31, x11, x31 # x31 = &B[g]
ld    x5, 0(x30)    # x5 = A[f]

addi  x12, x30, 8   # x12 = &A[f+1]
ld    x30, 0(x12)   # x30 = A[f+1]
add   x30, x30, x5  # x30 = A[f] + A[f+1]
sd    x30, 0(x31)   # B[g] = A[f] + A[f+1]

# This also has the side effect of setting f = A[f],
# so more accurately it might be
# B[g] = A[f+1] + A[f = A[f]]
```

2.13

```
Instruction:

sd x5, 32(x30)

Binary:

0000001 00101 11110 011 00000 0100011

= 0000 0010 0101 1111 0011 0000 0010 0011

Hexadecimal:

02 5F 30 23
```