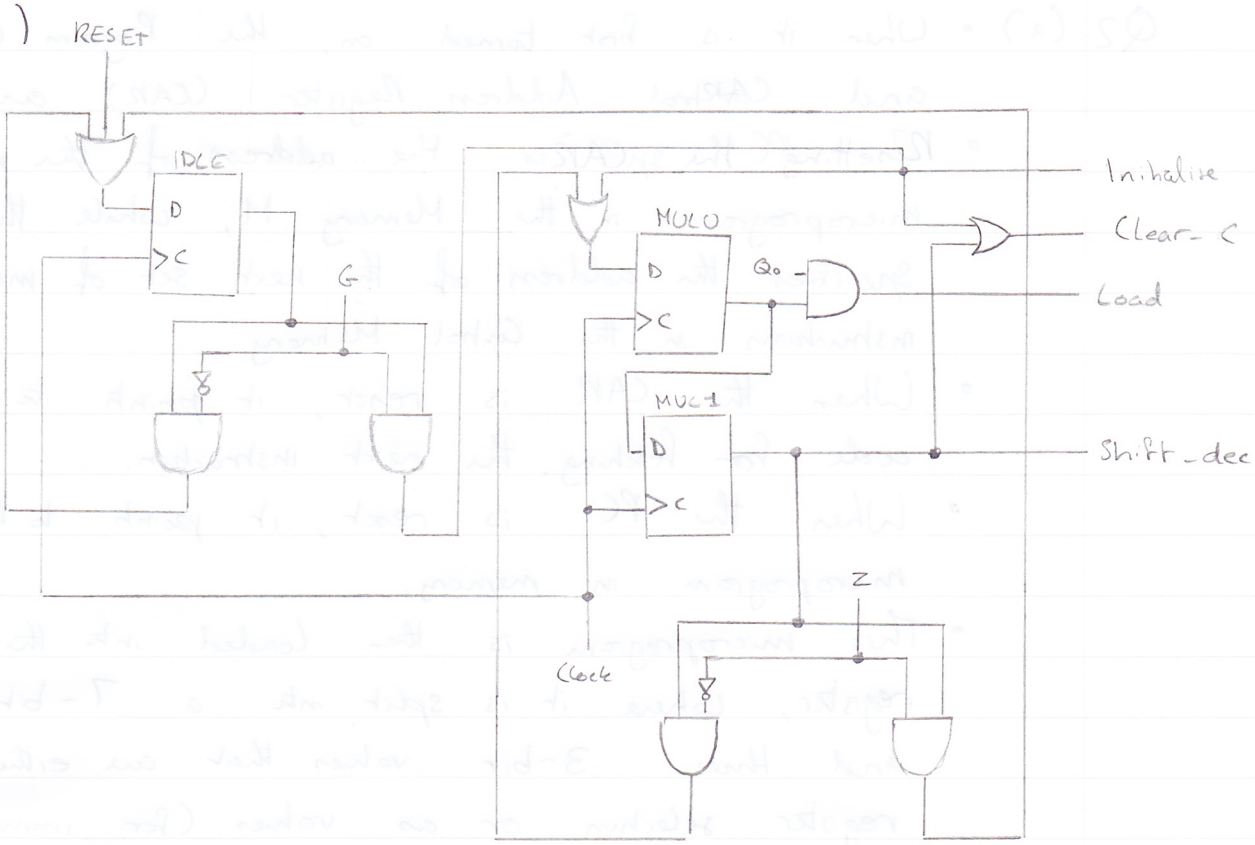
KVH CS2022 Paper 2013

Q1 (a)



With initialise = $A \leftarrow 0$, $P \leftarrow n-1$

Clear_C = $C \leftarrow 0$

Load = $A \leftarrow A+B$, $C \leftarrow C_{out}$

Shift_dec = $C \| A \| Q \leftarrow sr\ C \| A \| Q$, $P \leftarrow P-1$

(b) In a microcoded control solution, control words are now stored and as such they cannot be made dynamically depend on the value of control input. Hence, additional states must be introduced to supply these alternative control words.

As a result, conditional output boxes will be replaced with state boxes in the ASM chart of the microcoded solution.

Q2. (a)
- When it is first turned on, the Program Counter (PC) and Control Address Register (CAR) are both reset.
- The PC specifies the address of the next microprogram in the Memory M, while the CAR specifies the address of the next set of machine code instructions in the Control Memory.
- When the CAR is reset, it points to the machine code for fetching the next instruction.
- When the PC is reset, it points to the first microprogram in memory.
- This microprogram is then loaded into the IR register, where it is split into a 7-bit opcode, and three 3-bit values that are either used for register selection or as values (for immediate values or PC offset).
- The opcode is loaded into MUX C, and, as the control memory is in the Instruction Fetch state, it is loaded from MUX C into the CAR and then used to select the next set of machine code in the Control Memory.
- The Control Memory will carry out one or more instructions before going back into instruction Fetch mode.
- The Control Memory controls the inputs to all of the multiplexers, the functional unit, the PC, IR, CAR, and memory.

NOTE Not sure how much detail is needed, so just adding in extra for the sake of it.

KVH CS2022 Paper 2013

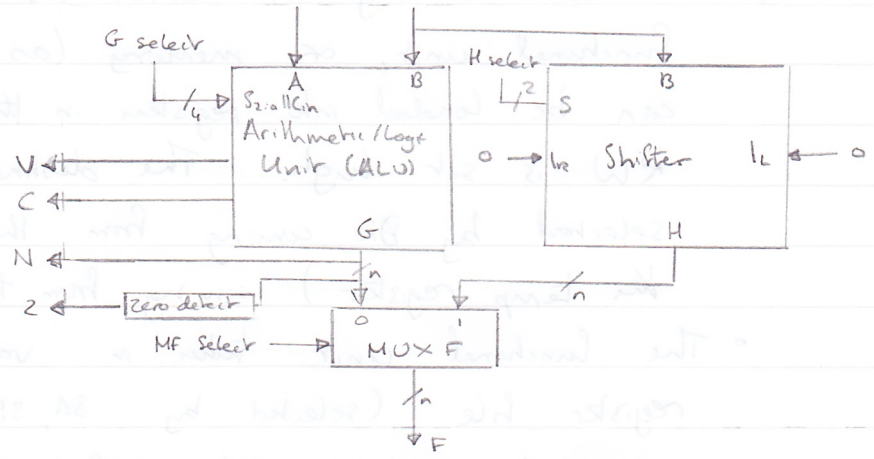
Q2. (a cont.)

- Values coming from either the output of the functional unit or memory (as selected by MUX D) can be loaded into registers in the register file if RW is set high. The destination register is selected by DR coming from the IR, or TD (for the temp register) coming from the control memory.
- The functional unit takes in values from the register file (selected by SA, SB, TA or TB) or immediate values and performs arithmetic, logic or shifting operations on these values. The status bits from the functional unit are used for conditional instructions in the microprogrammed control.
- Values from the register files can also be stored in memory if MW is set.
- The processor is pipelined, so a number of operations can be happening simultaneously, ie. instruction fetching, execution of a previous instruction, and writing of the results of a previous instruction to the register file.

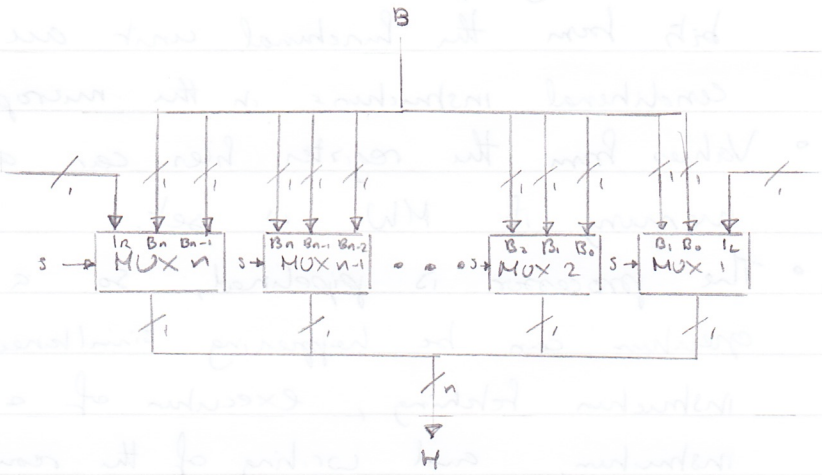
(b) NOTE Not sure what Immediate Instruction means

Function Unit

Q3. (a)

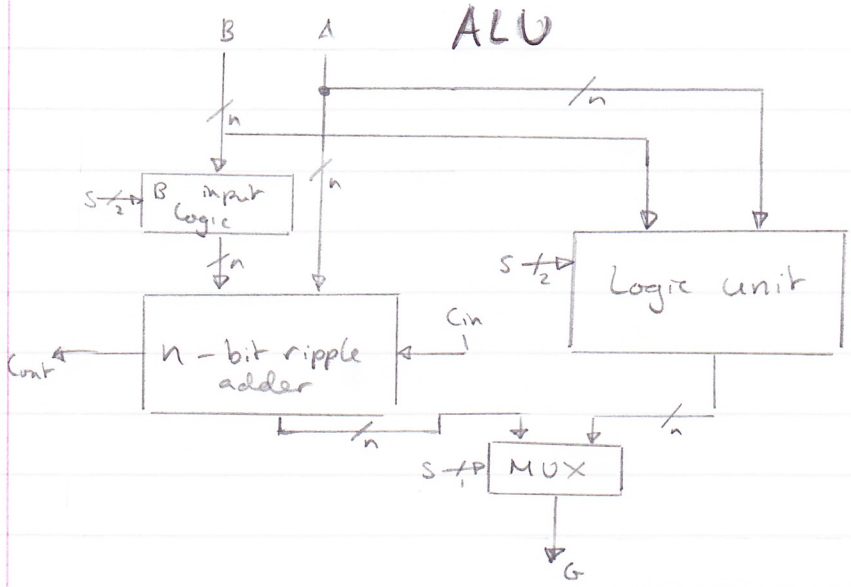


Shifter



Note: The shifter contains n multiplexers. The input B is split into individual bits which are input to the multiplexers, all of the multiplexers share the same select bits S.

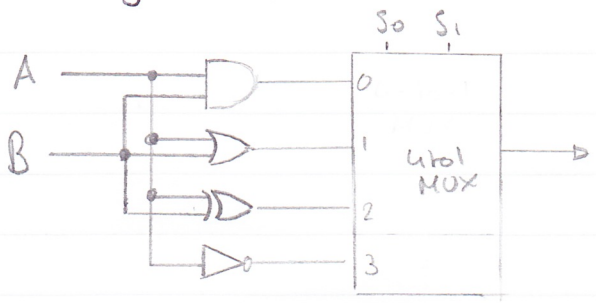
ALU



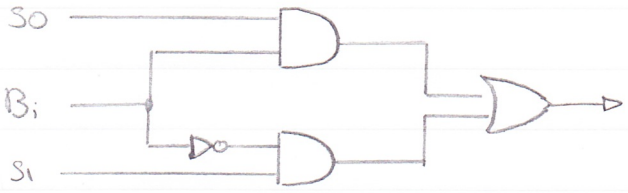
KVH CS2022 Paper 2013

Q3 (a cont.) Note Not sure how low level is needed.

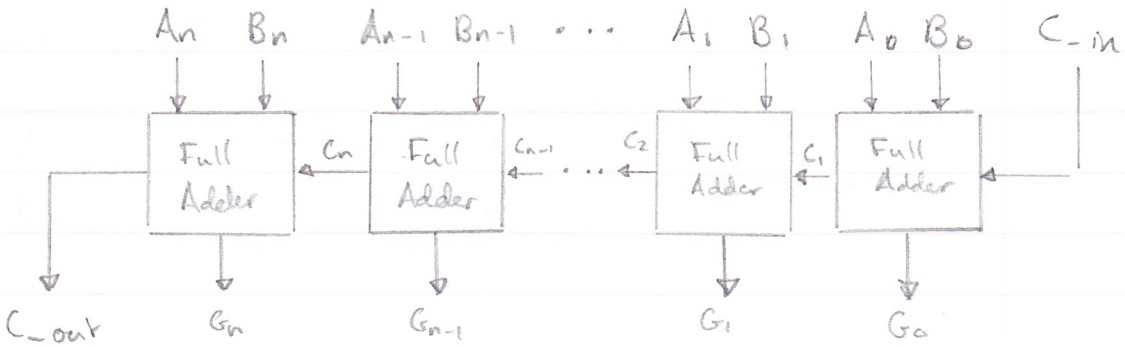
Logic unit



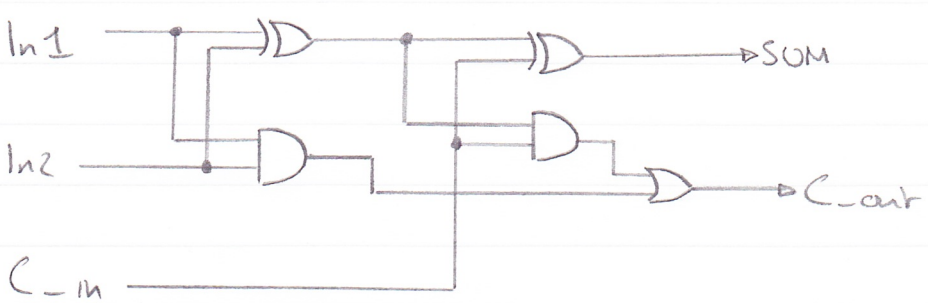
B input Logic (1-bit slice)



Ripple Adder



Full Adeler



KVH CS2022 Paper 2013

Q3 (b) The boolean expressions implement a carry look-ahead adder. The equation $C_{i+1} = g_i + p_i C_i$ defines the carry out for each stage of the adder.

A carry out occurs when either both inputs are 1 ($g_i = x_i y_i$) or when there is a carry in for that stage and at least one of the inputs is 1 ($p_i = x_i + y_i$).

At each stage, the equation is fully expanded, so as to reduce the propagation delay, at the expense of redundant gates.

The alternative (as in the ripple adder) is to simply take the p_i AND C_i of the previous stage, which incurs a great propagation delay, linear relative to the adder length.

The carry lookahead design instead causes (at best) a logarithmic propagation delay relative to the adder length, with each further stage having a longer delay on its carry in.

The carry lookahead design ~~uses~~ uses increasingly large gates, which have a limited fanout, so the actual efficiency (in terms of propagation delay, power usage and physical size) of this method decreases the larger the adder is.

KVH CS2022 Paper 2013

Q4 (a)

reg4.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity reg4 is
    Port ( D : in std_logic_vector (3 downto 0);
           Load, Clk : in std_logic;
           Q : out std_logic_vector (3 downto 0));
end reg4;

architecture behavioural of reg4 is
begin
    process (Clk)
    begin
        if (rising_edge (Clk)) then
            if (Load = '1') then
                Q <= D after 5ns;
            end if;
        end if;
    end proc;
end behavioural;
```

mux2_4bit.vhd

```
Library ieee;
use ieee.std_logic_1164.all;

entity mux2_4bit is
    Port( In0, In1 : in std_logic_vector (3 downto 0);
          S: in std_logic;
          Z: out std_logic_vector (3 downto 0));
end mux2_4bit;

architecture behavioural of mux2_4bit is
begin
      Z <= In0 after 5ns when S = '0' else
           In1 after 5ns when S = '1' else
           x"0" after 5ns;
end behavioural;
```

decoder_2to4.vhd

```
Library ieee;
use ieee.std_logic_1164.all;

entity decoder_2to4 is
      Port( A0, A1 : in std_logic;
            Q0, Q1, Q2, Q3 : out std_logic);
end decoder_2to4;

architecture behavioural of decoder_2to4 is
begin
      Q0 <= ((not A0) and (not A1)) after 5ns;
      Q1 <= ( A0 and (not A1)) after 5ns;
      Q2 <= ((not A0) and A1) after 5ns;
      Q3 <= (A0 and A1) after 5ns;
end behavioural;
```

KVH CS2022 Paper 2013

Q4 (a cont.)

mux4_4bit.vhd

```
Library ieee;
use ieee.std_logic_1164.all;

entity mux4_4bit is
    Port ( In0, In1, In2, In3 : in std_logic_vector (3 downto 0);
           S0, S1 : in std_logic;
           Z : out std_logic_vector (3 downto 0));
end mux4_4bit;

architecture behavioural of mux4_4bit is
begin
    Z <= In0 after 5ns when S0 = '0' and S1 = '0' else
         In1 after 5ns when S0 = '1' and S1 = '0' else
         In2 after 5ns when S0 = '0' and S1 = '1' else
         In3 after 5ns when S0 = '1' and S1 = '1' else
         x"0" after 5ns;
end behavioural;
```

register_file.vhd

Q4.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity register_file is
    Port ( src_s0, src_s1, des_A0, des_A1 : in std_logic;
           Clk, data_src : in std_logic;
           data : in std_logic_vector (3 downto 0);
           reg0, reg1, reg2, reg3 : out std_logic_vector (3 downto 0) );
end register_file;

architecture behavioural of register_file is
    component reg4
        Port ( D : in std_logic_vector (3 downto 0);
               Load, Clk : in std_logic;
               Q : out std_logic_vector (3 downto 0));
    end component;

    component decoder_2to4
        Port( A0, A1 : in std_logic;
              Q0, Q1, Q2, Q3 : out std_logic );
    end component;

    component mux2_4bit
        Port (In0, In1 : in std_logic_vector (3 downto 0);
              S : in std_logic;
              Z : out std_logic_vector (3 downto 0);
    end component;
```

KVH CS2022 Paper 2013

```
component mux4_4bit
    Port (In0, In1, In2, In3 : in std_logic_vector (3 downto 0);
          S0, S1 : in std_logic;
          Z: out std_logic_vector (3 downto 0));
end component;

signal load_reg0, load_reg1, load_reg2,
                  load_reg3 : std_logic;
signal reg0_q, reg1_q, reg2_q, reg3_q,
       data_src_mux_out, src_reg : std_logic_vector (3 downto 0);

begin
    reg00 : reg4 port map(
            D => data_src_mux_out,
            load => load_reg0,
            Clk => Clk,
            Q => reg0_q
    );

    reg01 : reg4 port map(
            D => data_src_mux_out,
            load => load_reg1,
            Clk => Clk,
            Q => reg1_q
    );

    reg02 : reg4 port map(
            D => data_src_mux_out,
            load => load_reg2,
            Clk => Clk,
            Q => reg2_q
    );
```

```
reg03: reg4 port map(
        D => data_src_mux_out,
        Load => Load_reg3,
        Clk => Clk,
        Q => reg3_q
);

des_decoder_2to4: decoder_2to4 port map(
        A0 => des_A0,
        A1 => des_A1,
        Q0 => Load_reg0,
        Q1 => Load_reg1,
        Q2 => Load_reg2,
        Q3 => Load_reg3
);

data_src_mux2_4bit : mux2_4bit port map(
        In0 => data,
        In1 => src_reg,
        S => data_src,
        Z => data_src_mux_out
);

Inst_mux4_4bit: mux4_4bit port map(
        In0 => reg0_q,
        In1 => reg1_q,
        In2 => reg2_q,
        In3 => reg3_q,
        S0 => src_S0,
        S1 => src_S1,
        Z => src_reg
);
```

KVH CS2022 Paper 2013

```
        reg0 <= reg0_q;
        reg1 <= reg1_q;
        reg2 <= reg2_q;
        reg3 <= reg3_q;
end behavioural;
```

Q4 (b) This register file can load data from external sources using the data(3:0) input, one register at a time. Alternatively, it can copy data from one register to another using the src and des selection inputs.

The data_src input decides whether external source or register source data will be loaded into the destination register.

On every clock cycle, one of the registers will be updated, so it is difficult to have all registers keep their values unless one is careful to always copy the value of a register back to itself.