Contents

1 Livre des Solutions

Ce document contient les solutions completes de tous les exercices du simulateur web Codex.

Note: Ces solutions sont fournies pour reference apres avoir tente les exercices.

1.1 A. Solutions HDL (Portes Logiques)

1.1.1 Inv

```
-- Inverter (NOT gate)
-- Inv(a) = Nand(a, a)

entity Inv is
  port(
    a : in bit;
    y : out bit
  );
end entity;

architecture rtl of Inv is
  component nand2
    port(a : in bit; b : in bit; y : out bit);
  end component;
begin
  u0: nand2 port map (a => a, b => a, y => y);
end architecture;
```

1.1.2 And2

```
-- AND gate
-- And2(a,b) = Inv(Nand(a,b))

entity And2 is
  port(
    a : in bit;
    b : in bit;
    y : out bit
  );
end entity;

architecture rtl of And2 is
  component nand2
    port(a : in bit; b : in bit; y : out bit);
```

```
  end component;
  component Inv
    port(a : in bit; y : out bit);
  end component;
  signal t : bit;
begin
  u0: nand2 port map (a => a, b => b, y => t);
  u1: Inv port map (a => t, y => y);
end architecture;
```

1.1.3 Or2

```
-- OR gate
-- Or2(a,b) = Nand(Inv(a), Inv(b))

entity Or2 is
  port(
    a : in bit;
    b : in bit;
    y : out bit
  );
end entity;

architecture rtl of Or2 is
  component nand2
    port(a : in bit; b : in bit; y : out bit);
  end component;
  component Inv
    port(a : in bit; y : out bit);
  end component;
  signal na, nb : bit;
begin
  u0: Inv port map (a => a, y => na);
  u1: Inv port map (a => b, y => nb);
  u2: nand2 port map (a => na, b => nb, y => y);
end architecture;
```

1.1.4 Xor2

```
-- XOR gate
-- Xor2(a,b) = Or2(And2(a, Inv(b)), And2(Inv(a), b))

entity Xor2 is
  port(
    a : in bit;
    b : in bit;
    y : out bit
  );
end entity;

architecture rtl of Xor2 is
  component Inv
    port(a : in bit; y : out bit);
```

```
  end component;
  component And2
    port(a : in bit; b : in bit; y : out bit);
  end component;
  component Or2
    port(a : in bit; b : in bit; y : out bit);
  end component;
  signal na, nb, t1, t2 : bit;
begin
  u0: Inv port map (a => a, y => na);
  u1: Inv port map (a => b, y => nb);
  u2: And2 port map (a => a, b => nb, y => t1);
  u3: And2 port map (a => na, b => b, y => t2);
  u4: Or2 port map (a => t1, b => t2, y => y);
end architecture;
```

1.1.5 Mux

```
-- 2-way Multiplexer
-- if sel=0 then y=a else y=b

entity Mux is
  port(
    a   : in bit;
    b   : in bit;
    sel : in bit;
    y   : out bit
  );
end entity;

architecture rtl of Mux is
  component Inv
    port(a : in bit; y : out bit);
  end component;
  component And2
    port(a : in bit; b : in bit; y : out bit);
  end component;
  component Or2
    port(a : in bit; b : in bit; y : out bit);
  end component;
  signal nsel, t1, t2 : bit;
begin
  u0: Inv port map (a => sel, y => nsel);
  u1: And2 port map (a => a, b => nsel, y => t1);
  u2: And2 port map (a => b, b => sel, y => t2);
  u3: Or2 port map (a => t1, b => t2, y => y);
end architecture;
```

1.1.6 DMux

```
-- Demultiplexer
-- if sel=0 then {a,b}={x,0} else {a,b}={0,x}
```

```
entity DMux is
  port(
    x   : in bit;
    sel : in bit;
    a   : out bit;
    b   : out bit
  );
end entity;

architecture rtl of DMux is
  component Inv
    port(a : in bit; y : out bit);
  end component;
  component And2
    port(a : in bit; b : in bit; y : out bit);
  end component;
  signal nsel : bit;
begin
  u0: Inv port map (a => sel, y => nsel);
  u1: And2 port map (a => x, b => nsel, y => a);
  u2: And2 port map (a => x, b => sel, y => b);
end architecture;
```

1.1.7 Inv16

```
-- 16-bit Inverter
entity Inv16 is
  port(
    a : in bits(15 downto 0);
    y : out bits(15 downto 0)
  );
end entity;

architecture rtl of Inv16 is
  component Inv
    port(a : in bit; y : out bit);
  end component;
begin
  u0: Inv port map (a => a(0), y => y(0));
  u1: Inv port map (a => a(1), y => y(1));
  u2: Inv port map (a => a(2), y => y(2));
  u3: Inv port map (a => a(3), y => y(3));
  u4: Inv port map (a => a(4), y => y(4));
  u5: Inv port map (a => a(5), y => y(5));
  u6: Inv port map (a => a(6), y => y(6));
  u7: Inv port map (a => a(7), y => y(7));
  u8: Inv port map (a => a(8), y => y(8));
  u9: Inv port map (a => a(9), y => y(9));
  u10: Inv port map (a => a(10), y => y(10));
  u11: Inv port map (a => a(11), y => y(11));
  u12: Inv port map (a => a(12), y => y(12));
  u13: Inv port map (a => a(13), y => y(13));
  u14: Inv port map (a => a(14), y => y(14));
  u15: Inv port map (a => a(15), y => y(15));
end architecture;
```

1.1.8 And16

```
-- 16-bit AND
entity And16 is
  port(
    a : in bits(15 downto 0);
    b : in bits(15 downto 0);
    y : out bits(15 downto 0)
  );
end entity;

architecture rtl of And16 is
  component And2
    port(a : in bit; b : in bit; y : out bit);
  end component;
begin
  u0: And2 port map (a => a(0), b => b(0), y => y(0));
  u1: And2 port map (a => a(1), b => b(1), y => y(1));
  u2: And2 port map (a => a(2), b => b(2), y => y(2));
  u3: And2 port map (a => a(3), b => b(3), y => y(3));
  u4: And2 port map (a => a(4), b => b(4), y => y(4));
  u5: And2 port map (a => a(5), b => b(5), y => y(5));
  u6: And2 port map (a => a(6), b => b(6), y => y(6));
  u7: And2 port map (a => a(7), b => b(7), y => y(7));
  u8: And2 port map (a => a(8), b => b(8), y => y(8));
  u9: And2 port map (a => a(9), b => b(9), y => y(9));
  u10: And2 port map (a => a(10), b => b(10), y => y(10));
  u11: And2 port map (a => a(11), b => b(11), y => y(11));
  u12: And2 port map (a => a(12), b => b(12), y => y(12));
  u13: And2 port map (a => a(13), b => b(13), y => y(13));
  u14: And2 port map (a => a(14), b => b(14), y => y(14));
  u15: And2 port map (a => a(15), b => b(15), y => y(15));
end architecture;
```

1.1.9 Or16

```
-- 16-bit OR
entity Or16 is
  port(
    a : in bits(15 downto 0);
    b : in bits(15 downto 0);
    y : out bits(15 downto 0)
  );
end entity;

architecture rtl of Or16 is
  component Or2
    port(a : in bit; b : in bit; y : out bit);
  end component;
begin
  u0: Or2 port map (a => a(0), b => b(0), y => y(0));
  u1: Or2 port map (a => a(1), b => b(1), y => y(1));
  u2: Or2 port map (a => a(2), b => b(2), y => y(2));
```

```
  u3: Or2 port map (a => a(3), b => b(3), y => y(3));
  u4: Or2 port map (a => a(4), b => b(4), y => y(4));
  u5: Or2 port map (a => a(5), b => b(5), y => y(5));
  u6: Or2 port map (a => a(6), b => b(6), y => y(6));
  u7: Or2 port map (a => a(7), b => b(7), y => y(7));
  u8: Or2 port map (a => a(8), b => b(8), y => y(8));
  u9: Or2 port map (a => a(9), b => b(9), y => y(9));
  u10: Or2 port map (a => a(10), b => b(10), y => y(10));
  u11: Or2 port map (a => a(11), b => b(11), y => y(11));
  u12: Or2 port map (a => a(12), b => b(12), y => y(12));
  u13: Or2 port map (a => a(13), b => b(13), y => y(13));
  u14: Or2 port map (a => a(14), b => b(14), y => y(14));
  u15: Or2 port map (a => a(15), b => b(15), y => y(15));
end architecture;
```

1.1.10 Mux16

```
-- 16-bit 2-way Multiplexer
entity Mux16 is
  port(
    a   : in bits(15 downto 0);
    b   : in bits(15 downto 0);
    sel : in bit;
    y   : out bits(15 downto 0)
  );
end entity;

architecture rtl of Mux16 is
  component Mux
    port(a : in bit; b : in bit; sel : in bit; y : out bit);
  end component;
begin
  u0: Mux port map (a => a(0), b => b(0), sel => sel, y => y(0));
  u1: Mux port map (a => a(1), b => b(1), sel => sel, y => y(1));
  u2: Mux port map (a => a(2), b => b(2), sel => sel, y => y(2));
  u3: Mux port map (a => a(3), b => b(3), sel => sel, y => y(3));
  u4: Mux port map (a => a(4), b => b(4), sel => sel, y => y(4));
  u5: Mux port map (a => a(5), b => b(5), sel => sel, y => y(5));
  u6: Mux port map (a => a(6), b => b(6), sel => sel, y => y(6));
  u7: Mux port map (a => a(7), b => b(7), sel => sel, y => y(7));
  u8: Mux port map (a => a(8), b => b(8), sel => sel, y => y(8));
  u9: Mux port map (a => a(9), b => b(9), sel => sel, y => y(9));
  u10: Mux port map (a => a(10), b => b(10), sel => sel, y => y(10));
  u11: Mux port map (a => a(11), b => b(11), sel => sel, y => y(11));
  u12: Mux port map (a => a(12), b => b(12), sel => sel, y => y(12));
  u13: Mux port map (a => a(13), b => b(13), sel => sel, y => y(13));
  u14: Mux port map (a => a(14), b => b(14), sel => sel, y => y(14));
  u15: Mux port map (a => a(15), b => b(15), sel => sel, y => y(15));
end architecture;
```

1.1.11 Or8Way

```
-- 8-way OR
entity Or8Way is
  port(
    a : in bits(7 downto 0);
    y : out bit
  );
end entity;

architecture rtl of Or8Way is
  component Or2
    port(a : in bit; b : in bit; y : out bit);
  end component;
  signal t1, t2, t3, t4, t5, t6 : bit;
begin
  u0: Or2 port map (a => a(0), b => a(1), y => t1);
  u1: Or2 port map (a => a(2), b => a(3), y => t2);
  u2: Or2 port map (a => a(4), b => a(5), y => t3);
  u3: Or2 port map (a => a(6), b => a(7), y => t4);
  u4: Or2 port map (a => t1, b => t2, y => t5);
  u5: Or2 port map (a => t3, b => t4, y => t6);
  u6: Or2 port map (a => t5, b => t6, y => y);
end architecture;
```

1.1.12 Mux4Way16

```
-- 4-way 16-bit Multiplexer
entity Mux4Way16 is
  port(
    a   : in bits(15 downto 0);
    b   : in bits(15 downto 0);
    c   : in bits(15 downto 0);
    d   : in bits(15 downto 0);
    sel : in bits(1 downto 0);
    y   : out bits(15 downto 0)
  );
end entity;

architecture rtl of Mux4Way16 is
  component Mux16
    port(a : in bits(15 downto 0); b : in bits(15 downto 0); sel : in bit; y : out bits(15 do
  end component;
  signal ab, cd : bits(15 downto 0);
begin
  u0: Mux16 port map (a => a, b => b, sel => sel(0), y => ab);
  u1: Mux16 port map (a => c, b => d, sel => sel(0), y => cd);
  u2: Mux16 port map (a => ab, b => cd, sel => sel(1), y => y);
end architecture;
```

1.1.13 Mux8Way16

```
-- 8-way 16-bit Multiplexer
entity Mux8Way16 is
```

```
  port(
    a   : in bits(15 downto 0);
    b   : in bits(15 downto 0);
    c   : in bits(15 downto 0);
    d   : in bits(15 downto 0);
    e   : in bits(15 downto 0);
    f   : in bits(15 downto 0);
    g   : in bits(15 downto 0);
    h   : in bits(15 downto 0);
    sel : in bits(2 downto 0);
    y   : out bits(15 downto 0)
  );
end entity;

architecture rtl of Mux8Way16 is
  component Mux4Way16
    port(a,b,c,d : in bits(15 downto 0); sel : in bits(1 downto 0); y : out bits(15 downto 0)
  end component;
  component Mux16
    port(a : in bits(15 downto 0); b : in bits(15 downto 0); sel : in bit; y : out bits(15 do
  end component;
  signal lo, hi : bits(15 downto 0);
begin
  u0: Mux4Way16 port map (a => a, b => b, c => c, d => d, sel => sel(1 downto 0), y => lo);
  u1: Mux4Way16 port map (a => e, b => f, c => g, d => h, sel => sel(1 downto 0), y => hi);
  u2: Mux16 port map (a => lo, b => hi, sel => sel(2), y => y);
end architecture;
```

1.1.14 DMux4Way

```
-- 4-way Demultiplexer
entity DMux4Way is
  port(
    x   : in bit;
    sel : in bits(1 downto 0);
    a   : out bit;
    b   : out bit;
    c   : out bit;
    d   : out bit
  );
end entity;

architecture rtl of DMux4Way is
  component DMux
    port(x : in bit; sel : in bit; a : out bit; b : out bit);
  end component;
  signal lo, hi : bit;
begin
  u0: DMux port map (x => x, sel => sel(1), a => lo, b => hi);
  u1: DMux port map (x => lo, sel => sel(0), a => a, b => b);
  u2: DMux port map (x => hi, sel => sel(0), a => c, b => d);
end architecture;
```

1.1.15 DMux8Way

```
-- 8-way Demultiplexer
entity DMux8Way is
  port(
    x   : in bit;
    sel : in bits(2 downto 0);
    a, b, c, d, e, f, g, h : out bit
  );
end entity;

architecture rtl of DMux8Way is
  component DMux
    port(x : in bit; sel : in bit; a : out bit; b : out bit);
  end component;
  component DMux4Way
    port(x : in bit; sel : in bits(1 downto 0); a,b,c,d : out bit);
  end component;
  signal lo, hi : bit;
begin
  u0: DMux port map (x => x, sel => sel(2), a => lo, b => hi);
  u1: DMux4Way port map (x => lo, sel => sel(1 downto 0), a => a, b => b, c => c, d => d);
  u2: DMux4Way port map (x => hi, sel => sel(1 downto 0), a => e, b => f, c => g, d => h);
end architecture;
```

1.1.16 HalfAdder

```
-- Half Adder
entity HalfAdder is
  port(
    a     : in bit;
    b     : in bit;
    sum   : out bit;
    carry : out bit
  );
end entity;

architecture rtl of HalfAdder is
  component Xor2
    port(a : in bit; b : in bit; y : out bit);
  end component;
  component And2
    port(a : in bit; b : in bit; y : out bit);
  end component;
begin
  u0: Xor2 port map (a => a, b => b, y => sum);
  u1: And2 port map (a => a, b => b, y => carry);
end architecture;
```

1.1.17 FullAdder

```
-- Full Adder
entity FullAdder is
  port(
```

```
    a     : in bit;
    b     : in bit;
    cin   : in bit;
    sum   : out bit;
    cout  : out bit
  );
end entity;

architecture rtl of FullAdder is
  component HalfAdder
    port(a : in bit; b : in bit; sum : out bit; carry : out bit);
  end component;
  component Or2
    port(a : in bit; b : in bit; y : out bit);
  end component;
  signal s1, c1, c2 : bit;
begin
  u0: HalfAdder port map (a => a, b => b, sum => s1, carry => c1);
  u1: HalfAdder port map (a => s1, b => cin, sum => sum, carry => c2);
  u2: Or2 port map (a => c1, b => c2, y => cout);
end architecture;
```

1.1.18 Add16

```
-- 16-bit Adder
entity Add16 is
  port(
    a    : in bits(15 downto 0);
    b    : in bits(15 downto 0);
    cin  : in bit;
    sum  : out bits(15 downto 0);
    cout : out bit
  );
end entity;

architecture rtl of Add16 is
  component FullAdder
    port(a,b,cin : in bit; sum,cout : out bit);
  end component;
  signal c : bits(16 downto 0);
begin
  u0: FullAdder port map (a => a(0), b => b(0), cin => cin, sum => sum(0), cout => c(1));
  u1: FullAdder port map (a => a(1), b => b(1), cin => c(1), sum => sum(1), cout => c(2));
  u2: FullAdder port map (a => a(2), b => b(2), cin => c(2), sum => sum(2), cout => c(3));
  u3: FullAdder port map (a => a(3), b => b(3), cin => c(3), sum => sum(3), cout => c(4));
  u4: FullAdder port map (a => a(4), b => b(4), cin => c(4), sum => sum(4), cout => c(5));
  u5: FullAdder port map (a => a(5), b => b(5), cin => c(5), sum => sum(5), cout => c(6));
  u6: FullAdder port map (a => a(6), b => b(6), cin => c(6), sum => sum(6), cout => c(7));
  u7: FullAdder port map (a => a(7), b => b(7), cin => c(7), sum => sum(7), cout => c(8));
  u8: FullAdder port map (a => a(8), b => b(8), cin => c(8), sum => sum(8), cout => c(9));
  u9: FullAdder port map (a => a(9), b => b(9), cin => c(9), sum => sum(9), cout => c(10));
  u10: FullAdder port map (a => a(10), b => b(10), cin => c(10), sum => sum(10), cout => c(11
  u11: FullAdder port map (a => a(11), b => b(11), cin => c(11), sum => sum(11), cout => c(12
  u12: FullAdder port map (a => a(12), b => b(12), cin => c(12), sum => sum(12), cout => c(13
  u13: FullAdder port map (a => a(13), b => b(13), cin => c(13), sum => sum(13), cout => c(14
```

```
  u14: FullAdder port map (a => a(14), b => b(14), cin => c(14), sum => sum(14), cout => c(15
  u15: FullAdder port map (a => a(15), b => b(15), cin => c(15), sum => sum(15), cout => cout
end architecture;
```

1.1.19 Inc16

```
-- 16-bit Incrementer
entity Inc16 is
  port(
    a : in bits(15 downto 0);
    y : out bits(15 downto 0)
  );
end entity;

architecture rtl of Inc16 is
  component Add16
    port(a,b : in bits(15 downto 0); cin : in bit; sum : out bits(15 downto 0); cout : out bi
  end component;
  signal zero16 : bits(15 downto 0);
  signal unused_cout : bit;
begin
  zero16 <= X"0000";
  u0: Add16 port map (a => a, b => zero16, cin => '1', sum => y, cout => unused_cout);
end architecture;
```

1.1.20 Sub16

```
-- 16-bit Subtractor
entity Sub16 is
  port(
    a    : in bits(15 downto 0);
    b    : in bits(15 downto 0);
    diff : out bits(15 downto 0)
  );
end entity;

architecture rtl of Sub16 is
  component Add16
    port(a,b : in bits(15 downto 0); cin : in bit; sum : out bits(15 downto 0); cout : out bi
  end component;
  component Inv16
    port(a : in bits(15 downto 0); y : out bits(15 downto 0));
  end component;
  signal nb : bits(15 downto 0);
  signal unused_cout : bit;
begin
  u0: Inv16 port map (a => b, y => nb);
  u1: Add16 port map (a => a, b => nb, cin => '1', sum => diff, cout => unused_cout);
end architecture;
```

1.1.21 ALU

```
-- 16-bit ALU
-- op: 0=AND, 1=OR, 2=ADD, 3=SUB

entity ALU is
  port(
    a    : in bits(15 downto 0);
    b    : in bits(15 downto 0);
    op   : in bits(1 downto 0);
    y    : out bits(15 downto 0);
    zero : out bit;
    neg  : out bit
  );
end entity;

architecture rtl of ALU is
  component Add16
    port(a,b : in bits(15 downto 0); cin : in bit; sum : out bits(15 downto 0); cout : out bi
  end component;
  component And16
    port(a,b : in bits(15 downto 0); y : out bits(15 downto 0));
  end component;
  component Or16
    port(a,b : in bits(15 downto 0); y : out bits(15 downto 0));
  end component;
  component Inv16
    port(a : in bits(15 downto 0); y : out bits(15 downto 0));
  end component;
  component Mux4Way16
    port(a,b,c,d : in bits(15 downto 0); sel : in bits(1 downto 0); y : out bits(15 downto 0)
  end component;
  component Or8Way
    port(a : in bits(7 downto 0); y : out bit);
  end component;
  signal r_and, r_or, r_add, r_sub, nb, result : bits(15 downto 0);
  signal unused_cout1, unused_cout2 : bit;
  signal or_lo, or_hi : bit;
begin
  -- Compute all operations
  u_and: And16 port map (a => a, b => b, y => r_and);
  u_or: Or16 port map (a => a, b => b, y => r_or);
  u_add: Add16 port map (a => a, b => b, cin => '0', sum => r_add, cout => unused_cout1);

  -- SUB: a - b = a + (~b) + 1
  u_inv: Inv16 port map (a => b, y => nb);
  u_sub: Add16 port map (a => a, b => nb, cin => '1', sum => r_sub, cout => unused_cout2);

  -- Select result based on op
  u_mux: Mux4Way16 port map (a => r_and, b => r_or, c => r_add, d => r_sub, sel => op, y => r

  -- Output result
  y <= result;

  -- Zero flag: result == 0
  u_or_lo: Or8Way port map (a => result(7 downto 0), y => or_lo);
  u_or_hi: Or8Way port map (a => result(15 downto 8), y => or_hi);
```

```
  zero <= not (or_lo or or_hi);

  -- Negative flag: MSB of result
  neg <= result(15);
end architecture;
```

1.1.22 DFF1

```
-- D Flip-Flop
-- Uses the built-in dff primitive

entity DFF1 is
  port(
    clk : in bit;
    d   : in bit;
    q   : out bit
  );
end entity;

architecture rtl of DFF1 is
  component dff
    port(clk : in bit; d : in bit; q : out bit);
  end component;
begin
  u0: dff port map (clk => clk, d => d, q => q);
end architecture;
```

1.1.23 BitReg

```
-- 1-bit Register
entity BitReg is
  port(
    clk  : in bit;
    d    : in bit;
    load : in bit;
    q    : out bit
  );
end entity;

architecture rtl of BitReg is
  component dff
    port(clk : in bit; d : in bit; q : out bit);
  end component;
  component Mux
    port(a,b : in bit; sel : in bit; y : out bit);
  end component;
  signal mux_out, q_int : bit;
begin
  u_mux: Mux port map (a => q_int, b => d, sel => load, y => mux_out);
  u_dff: dff port map (clk => clk, d => mux_out, q => q_int);
  q <= q_int;
end architecture;
```

1.1.24 Register16

```
-- 16-bit Register
entity Register16 is
  port(
    clk  : in bit;
    d    : in bits(15 downto 0);
    load : in bit;
    q    : out bits(15 downto 0)
  );
end entity;

architecture rtl of Register16 is
  component BitReg
    port(clk : in bit; d : in bit; load : in bit; q : out bit);
  end component;
begin
  u0: BitReg port map (clk => clk, d => d(0), load => load, q => q(0));
  u1: BitReg port map (clk => clk, d => d(1), load => load, q => q(1));
  u2: BitReg port map (clk => clk, d => d(2), load => load, q => q(2));
  u3: BitReg port map (clk => clk, d => d(3), load => load, q => q(3));
  u4: BitReg port map (clk => clk, d => d(4), load => load, q => q(4));
  u5: BitReg port map (clk => clk, d => d(5), load => load, q => q(5));
  u6: BitReg port map (clk => clk, d => d(6), load => load, q => q(6));
  u7: BitReg port map (clk => clk, d => d(7), load => load, q => q(7));
  u8: BitReg port map (clk => clk, d => d(8), load => load, q => q(8));
  u9: BitReg port map (clk => clk, d => d(9), load => load, q => q(9));
  u10: BitReg port map (clk => clk, d => d(10), load => load, q => q(10));
  u11: BitReg port map (clk => clk, d => d(11), load => load, q => q(11));
  u12: BitReg port map (clk => clk, d => d(12), load => load, q => q(12));
  u13: BitReg port map (clk => clk, d => d(13), load => load, q => q(13));
  u14: BitReg port map (clk => clk, d => d(14), load => load, q => q(14));
  u15: BitReg port map (clk => clk, d => d(15), load => load, q => q(15));
end architecture;
```

1.1.25 PC

```
-- Program Counter
entity PC is
  port(
    clk   : in bit;
    d     : in bits(15 downto 0);
    inc   : in bit;
    load  : in bit;
    reset : in bit;
    q     : out bits(15 downto 0)
  );
end entity;

architecture rtl of PC is
  component Register16
    port(clk : in bit; d : in bits(15 downto 0); load : in bit; q : out bits(15 downto 0));
  end component;
  component Inc16
    port(a : in bits(15 downto 0); y : out bits(15 downto 0));
```

```
  end component;
  component Mux16
    port(a,b : in bits(15 downto 0); sel : in bit; y : out bits(15 downto 0));
  end component;
  signal q_int, inc_out, mux1_out, mux2_out, mux3_out : bits(15 downto 0);
  signal zero16 : bits(15 downto 0);
  signal do_load : bit;
begin
  zero16 <= X"0000";

  -- Increment current value
  u_inc: Inc16 port map (a => q_int, y => inc_out);

  -- Priority: reset > load > inc
  -- First mux: inc or hold
  u_mux1: Mux16 port map (a => q_int, b => inc_out, sel => inc, y => mux1_out);
  -- Second mux: load overrides
  u_mux2: Mux16 port map (a => mux1_out, b => d, sel => load, y => mux2_out);
  -- Third mux: reset overrides all
  u_mux3: Mux16 port map (a => mux2_out, b => zero16, sel => reset, y => mux3_out);

  -- Always load the register
  do_load <= inc or load or reset;
  u_reg: Register16 port map (clk => clk, d => mux3_out, load => do_load, q => q_int);

  q <= q_int;
end architecture;
```

1.1.26 RAM8

```
-- 8-word RAM
entity RAM8 is
  port(
    clk  : in bit;
    din  : in bits(15 downto 0);
    addr : in bits(2 downto 0);
    we   : in bit;
    dout : out bits(15 downto 0)
  );
end entity;

architecture rtl of RAM8 is
  component Register16
    port(clk : in bit; d : in bits(15 downto 0); load : in bit; q : out bits(15 downto 0));
  end component;
  component DMux8Way
    port(x : in bit; sel : in bits(2 downto 0); a,b,c,d,e,f,g,h : out bit);
  end component;
  component Mux8Way16
    port(a,b,c,d,e,f,g,h : in bits(15 downto 0); sel : in bits(2 downto 0); y : out bits(15 d
  end component;
  signal load0,load1,load2,load3,load4,load5,load6,load7 : bit;
  signal r0,r1,r2,r3,r4,r5,r6,r7 : bits(15 downto 0);
begin
  u_dmux: DMux8Way port map (x => we, sel => addr, a => load0, b => load1, c => load2, d => l
```

```
  u_r0: Register16 port map (clk => clk, d => din, load => load0, q => r0);
  u_r1: Register16 port map (clk => clk, d => din, load => load1, q => r1);
  u_r2: Register16 port map (clk => clk, d => din, load => load2, q => r2);
  u_r3: Register16 port map (clk => clk, d => din, load => load3, q => r3);
  u_r4: Register16 port map (clk => clk, d => din, load => load4, q => r4);
  u_r5: Register16 port map (clk => clk, d => din, load => load5, q => r5);
  u_r6: Register16 port map (clk => clk, d => din, load => load6, q => r6);
  u_r7: Register16 port map (clk => clk, d => din, load => load7, q => r7);

  u_mux: Mux8Way16 port map (a => r0, b => r1, c => r2, d => r3, e => r4, f => r5, g => r6, h
end architecture;
```

1.1.27 RAM64

```
-- 64-word RAM
entity RAM64 is
  port(
    clk  : in bit;
    din  : in bits(15 downto 0);
    addr : in bits(5 downto 0);
    we   : in bit;
    dout : out bits(15 downto 0)
  );
end entity;

architecture rtl of RAM64 is
  component RAM8
    port(clk : in bit; din : in bits(15 downto 0); addr : in bits(2 downto 0); we : in bit; d
  end component;
  component DMux8Way
    port(x : in bit; sel : in bits(2 downto 0); a,b,c,d,e,f,g,h : out bit);
  end component;
  component Mux8Way16
    port(a,b,c,d,e,f,g,h : in bits(15 downto 0); sel : in bits(2 downto 0); y : out bits(15 d
  end component;
  signal we0,we1,we2,we3,we4,we5,we6,we7 : bit;
  signal r0,r1,r2,r3,r4,r5,r6,r7 : bits(15 downto 0);
begin
  u_dmux: DMux8Way port map (x => we, sel => addr(5 downto 3), a => we0, b => we1, c => we2,

  u_ram0: RAM8 port map (clk => clk, din => din, addr => addr(2 downto 0), we => we0, dout =>
  u_ram1: RAM8 port map (clk => clk, din => din, addr => addr(2 downto 0), we => we1, dout =>
  u_ram2: RAM8 port map (clk => clk, din => din, addr => addr(2 downto 0), we => we2, dout =>
  u_ram3: RAM8 port map (clk => clk, din => din, addr => addr(2 downto 0), we => we3, dout =>
  u_ram4: RAM8 port map (clk => clk, din => din, addr => addr(2 downto 0), we => we4, dout =>
  u_ram5: RAM8 port map (clk => clk, din => din, addr => addr(2 downto 0), we => we5, dout =>
  u_ram6: RAM8 port map (clk => clk, din => din, addr => addr(2 downto 0), we => we6, dout =>
  u_ram7: RAM8 port map (clk => clk, din => din, addr => addr(2 downto 0), we => we7, dout =>

  u_mux: Mux8Way16 port map (a => r0, b => r1, c => r2, d => r3, e => r4, f => r5, g => r6, h
end architecture;
```

1.1.28 RegFile

```
-- 16-Register File using RAM primitive
entity RegFile is
  port(
    clk   : in bit;
    we    : in bit;
    waddr : in bits(3 downto 0);
    wdata : in bits(15 downto 0);
    raddr1: in bits(3 downto 0);
    raddr2: in bits(3 downto 0);
    rdata1: out bits(15 downto 0);
    rdata2: out bits(15 downto 0)
  );
end entity;

architecture rtl of RegFile is
  component ram
    port(clk : in bit; we : in bit; addr : in bits(3 downto 0);
         din : in bits(15 downto 0); dout : out bits(15 downto 0));
  end component;
begin
  -- Use two RAM instances for dual read ports
  u_ram1: ram port map (clk => clk, we => we, addr => waddr, din => wdata, dout => rdata1);
  u_ram2: ram port map (clk => clk, we => we, addr => waddr, din => wdata, dout => rdata2);
  -- Note: Simplified - real implementation would need proper read port addressing
end architecture;
```

1.1.29 Decoder

```
-- Instruction Decoder
-- Decodes opcode into control signals
-- Opcodes: 0000=ALU, 0100=LOAD, 0101=STORE, 1000=BRANCH

entity Decoder is
  port(
    opcode    : in bits(3 downto 0);
    alu_op    : out bits(1 downto 0);
    reg_write : out bit;
    mem_read  : out bit;
    mem_write : out bit;
    branch    : out bit
  );
end entity;

architecture rtl of Decoder is
  component Inv
    port(a : in bit; y : out bit);
  end component;
  component And2
    port(a, b : in bit; y : out bit);
  end component;
  signal not_op3, not_op2, not_op1, not_op0 : bit;
  signal is_alu, is_load, is_store, is_branch : bit;
begin
```

```
  -- Invert opcode bits
  inv3: Inv port map (a => opcode(3), y => not_op3);
  inv2: Inv port map (a => opcode(2), y => not_op2);
  inv1: Inv port map (a => opcode(1), y => not_op1);
  inv0: Inv port map (a => opcode(0), y => not_op0);

  -- Decode: ALU = 0000
  alu_a: And2 port map (a => not_op3, b => not_op2, y => is_alu);

  -- Decode: LOAD = 0100
  ld_a: And2 port map (a => not_op3, b => opcode(2), y => is_load);

  -- Decode: STORE = 0101
  st_a: And2 port map (a => is_load, b => opcode(0), y => is_store);

  -- Decode: BRANCH = 1xxx
  br_a: Inv port map (a => not_op3, y => is_branch);

  -- Control signals - pass through lower opcode bits for ALU operation
  alu_op <= opcode(1 downto 0);
  reg_write <= is_alu;
  mem_read <= is_load;
  mem_write <= is_store;
  branch <= is_branch;
end architecture;
```

1.1.30 CondCheck

```
-- Condition Checker
-- Checks ALU flags against condition code
-- cond: 0000=EQ (zero), 0001=NE (!zero), 0010=LT (neg), 0011=GE (!neg)

entity CondCheck is
  port(
    cond : in bits(3 downto 0);
    zero : in bit;
    neg  : in bit;
    carry: in bit;
    ovf  : in bit;
    take : out bit
  );
end entity;

architecture rtl of CondCheck is
  component Inv
    port(a : in bit; y : out bit);
  end component;
  component Mux
    port(a, b : in bit; sel : in bit; y : out bit);
  end component;
  signal not_zero, not_neg : bit;
  signal eq_result, ne_result, lt_result, ge_result : bit;
  signal sel0, sel1 : bit;
begin
  -- Invert flags for NE and GE conditions
```

```
  inv_z: Inv port map (a => zero, y => not_zero);
  inv_n: Inv port map (a => neg, y => not_neg);

  -- Condition results
  eq_result <= zero;       -- EQ: zero=1
  ne_result <= not_zero;   -- NE: zero=0
  lt_result <= neg;        -- LT: neg=1
  ge_result <= not_neg;    -- GE: neg=0

  -- 4-way mux using cond(1:0)
  mux0: Mux port map (a => eq_result, b => ne_result, sel => cond(0), y => sel0);
  mux1: Mux port map (a => lt_result, b => ge_result, sel => cond(0), y => sel1);
  mux2: Mux port map (a => sel0, b => sel1, sel => cond(1), y => take);
end architecture;
```

1.1.31 Control

```
-- Control Unit
-- Generates all control signals

entity Control is
  port(
    clk       : in bit;
    opcode    : in bits(3 downto 0);
    cond      : in bits(3 downto 0);
    zero      : in bit;
    neg       : in bit;
    alu_op    : out bits(1 downto 0);
    reg_write : out bit;
    mem_read  : out bit;
    mem_write : out bit;
    pc_src    : out bit
  );
end entity;

architecture rtl of Control is
  component Decoder
    port(opcode : in bits(3 downto 0); alu_op : out bits(1 downto 0);
         reg_write, mem_read, mem_write, branch : out bit);
  end component;
  component CondCheck
    port(cond : in bits(3 downto 0); zero,neg,carry,ovf : in bit; take : out bit);
  end component;
  component And2
    port(a, b : in bit; y : out bit);
  end component;
  signal branch_sig, cond_take : bit;
begin
  -- Decoder generates base control signals
  u_dec: Decoder port map (
    opcode => opcode,
    alu_op => alu_op,
    reg_write => reg_write,
    mem_read => mem_read,
    mem_write => mem_write,
```

```
    branch => branch_sig
  );

  -- CondCheck evaluates branch condition
  u_cond: CondCheck port map (
    cond => cond,
    zero => zero,
    neg => neg,
    carry => '0',
    ovf => '0',
    take => cond_take
  );

  -- pc_src = branch AND condition_met
  u_and: And2 port map (a => branch_sig, b => cond_take, y => pc_src);
end architecture;
```

1.1.32 CPU

```
-- A32-Lite CPU
-- Simple 16-bit RISC processor
-- Instruction format: [15:12]=opcode, [11:8]=rd, [7:4]=rs1, [3:0]=rs2/imm

entity CPU is
  port(
    clk     : in bit;
    reset   : in bit;
    instr   : in bits(15 downto 0);
    mem_in  : in bits(15 downto 0);
    mem_out : out bits(15 downto 0);
    mem_addr: out bits(15 downto 0);
    mem_we  : out bit;
    pc_out  : out bits(15 downto 0)
  );
end entity;

architecture rtl of CPU is
  component RegFile
    port(clk,we : in bit; waddr,raddr1,raddr2 : in bits(3 downto 0);
         wdata : in bits(15 downto 0); rdata1,rdata2 : out bits(15 downto 0));
  end component;
  component ALU
    port(a,b : in bits(15 downto 0); op : in bits(1 downto 0);
         y : out bits(15 downto 0); zero,neg : out bit);
  end component;
  component PC
    port(clk : in bit; d : in bits(15 downto 0); inc,load,reset : in bit; q : out bits(15 dow
  end component;
  component Control
    port(clk : in bit; opcode,cond : in bits(3 downto 0); zero,neg : in bit;
         alu_op : out bits(1 downto 0); reg_write,mem_read,mem_write,pc_src : out bit);
  end component;
  component Mux16
    port(a,b : in bits(15 downto 0); sel : in bit; y : out bits(15 downto 0));
  end component;
```

```
  -- Instruction decode
  signal opcode, rd, rs1, rs2 : bits(3 downto 0);
  -- Control signals
  signal alu_op : bits(1 downto 0);
  signal reg_write, mem_rd, mem_wr, pc_src : bit;
  -- Datapath signals
  signal pc_val, branch_target : bits(15 downto 0);
  signal reg_data1, reg_data2, alu_result : bits(15 downto 0);
  signal write_data : bits(15 downto 0);
  signal zero_flag, neg_flag : bit;
  signal not_reset : bit;
  component Inv
    port(a : in bit; y : out bit);
  end component;
begin
  -- Instruction decode using slices
  opcode <= instr(15 downto 12);
  rd <= instr(11 downto 8);
  rs1 <= instr(7 downto 4);
  rs2 <= instr(3 downto 0);

  -- Control unit
  u_ctrl: Control port map (
    clk => clk, opcode => opcode, cond => rs2,
    zero => zero_flag, neg => neg_flag,
    alu_op => alu_op, reg_write => reg_write,
    mem_read => mem_rd, mem_write => mem_wr, pc_src => pc_src
  );

  -- Register file
  u_regs: RegFile port map (
    clk => clk, we => reg_write,
    waddr => rd, raddr1 => rs1, raddr2 => rs2,
    wdata => write_data, rdata1 => reg_data1, rdata2 => reg_data2
  );

  -- ALU
  u_alu: ALU port map (
    a => reg_data1, b => reg_data2, op => alu_op,
    y => alu_result, zero => zero_flag, neg => neg_flag
  );

  -- Write back mux (ALU result or memory)
  u_wb_mux: Mux16 port map (
    a => alu_result, b => mem_in, sel => mem_rd, y => write_data
  );

  -- Program counter
  inv_reset: Inv port map (a => reset, y => not_reset);
  branch_target <= reg_data1;
  u_pc: PC port map (
    clk => clk, d => branch_target,
    inc => not_reset, load => pc_src, reset => reset, q => pc_val
  );
```

```
  -- Outputs
  pc_out <= pc_val;
  mem_addr <= alu_result;
  mem_out <= reg_data2;
  mem_we <= mem_wr;
end architecture;
```

1.1.33 IF_ID_Reg

```
-- IF/ID Pipeline Register

entity IF_ID_Reg is
  port(
    clk : in bit;
    reset : in bit;
    stall : in bit;
    flush : in bit;
    if_instr : in bits(31 downto 0);
    if_pc_plus4 : in bits(31 downto 0);
    id_instr : out bits(31 downto 0);
    id_pc_plus4 : out bits(31 downto 0)
  );
end entity;

architecture rtl of IF_ID_Reg is
  signal instr_reg : bits(31 downto 0);
  signal pc_plus4_reg : bits(31 downto 0);
begin
  process(clk)
  begin
    if rising_edge(clk) then
      if (reset = '1') or (flush = '1') then
        instr_reg <= x"E0000000";  -- NOP
        pc_plus4_reg <= x"00000000";
      elsif stall = '0' then
        instr_reg <= if_instr;
        pc_plus4_reg <= if_pc_plus4;
      end if;
    end if;
  end process;

  id_instr <= instr_reg;
  id_pc_plus4 <= pc_plus4_reg;
end architecture;
```

1.1.34 HazardDetect

```
-- Hazard Detection Unit

entity HazardDetect is
  port(
    id_rn : in bits(3 downto 0);
    id_rm : in bits(3 downto 0);
```

```
    id_rn_used : in bit;
    id_rm_used : in bit;
    ex_rd : in bits(3 downto 0);
    ex_mem_read : in bit;
    stall : out bit
  );
end entity;

architecture rtl of HazardDetect is
  signal rn_hazard : bit;
  signal rm_hazard : bit;
begin
  -- Hazard if: load in EX AND register used in ID AND same register
  rn_hazard <= ex_mem_read and id_rn_used and (id_rn = ex_rd);
  rm_hazard <= ex_mem_read and id_rm_used and (id_rm = ex_rd);
  stall <= rn_hazard or rm_hazard;
end architecture;
```

1.1.35 ForwardUnit

```
-- Forwarding Unit

entity ForwardUnit is
  port(
    ex_rn : in bits(3 downto 0);
    ex_rm : in bits(3 downto 0);
    mem_rd : in bits(3 downto 0);
    mem_reg_write : in bit;
    wb_rd : in bits(3 downto 0);
    wb_reg_write : in bit;
    forward_a : out bits(1 downto 0);
    forward_b : out bits(1 downto 0)
  );
end entity;

architecture rtl of ForwardUnit is
  signal mem_fwd_a : bit;
  signal wb_fwd_a : bit;
  signal mem_fwd_b : bit;
  signal wb_fwd_b : bit;
begin
  -- Detect forwarding conditions
  mem_fwd_a <= mem_reg_write and (mem_rd = ex_rn);
  wb_fwd_a <= wb_reg_write and (wb_rd = ex_rn) and (not mem_fwd_a);
  mem_fwd_b <= mem_reg_write and (mem_rd = ex_rm);
  wb_fwd_b <= wb_reg_write and (wb_rd = ex_rm) and (not mem_fwd_b);

  -- Encode output: 00=none, 01=MEM, 10=WB
  forward_a <= wb_fwd_a & mem_fwd_a;
  forward_b <= wb_fwd_b & mem_fwd_b;
end architecture;
```

1.1.36 CPU_Pipeline

```
-- 5-Stage Pipelined CPU
-- See hdl_lib/05_cpu/CPU_Pipeline.hdl for full implementation
-- This exercise is a capstone project

entity CPU_Pipeline is
  port(
    clk : in bit;
    reset : in bit;
    instr_addr : out bits(31 downto 0);
    instr_data : in bits(31 downto 0);
    mem_addr : out bits(31 downto 0);
    mem_wdata : out bits(31 downto 0);
    mem_rdata : in bits(31 downto 0);
    mem_read : out bit;
    mem_write : out bit;
    halted : out bit
  );
end entity;

architecture rtl of CPU_Pipeline is
begin
  -- Implementation uses IF_ID_Reg, HazardDetect, ForwardUnit
  -- and additional pipeline registers EX_MEM, MEM_WB
  -- See hdl_lib/05_cpu/CPU_Pipeline.hdl for complete code
  instr_addr <= x"00000000";
  mem_addr <= x"00000000";
  mem_wdata <= x"00000000";
  mem_read <= '0';
  mem_write <= '0';
  halted <= '0';
end architecture;
```

1.2 B. Solutions Assembleur A32

1.2.1 Hello World

```
; Hello World - Solution

.text
.global _start
_start:
    MOV R0, #42
    HALT
```

1.2.2 Addition

```
; Addition - Solution

.text
.global _start
_start:
    MOV R0, #15
```

```
    ADD R0, R0, #27
    HALT
```

1.2.3 Soustraction

```
; Soustraction - Solution

.text
.global _start
_start:
    MOV R0, #100
    SUB R0, R0, #58
    HALT
```

1.2.4 Logique

```
; Logique - Solution

.text
.global _start
_start:
    MOV R1, #0xFF
    MOV R2, #0x0F
    AND R0, R1, R2
    HALT
```

1.2.5 Doubler

```
; Doubler - Solution

.text
.global _start
_start:
    MOV R0, #21
    ADD R0, R0, R0   ; R0 = R0 + R0 = 2 * R0
    HALT
```

1.2.6 Conditions

```
; Conditions - Solution

.text
.global _start
_start:
    MOV R1, #25
    MOV R2, #17

    CMP R1, R2
    B.GT .r1_bigger
    MOV R0, R2
```

```
    B .done

.r1_bigger:
    MOV R0, R1

.done:
    HALT
```

1.2.7 Valeur Absolue

```
; Valeur Absolue - Solution

.text
.global _start
_start:
    MOV R1, #0
    SUB R1, R1, #42  ; R1 = -42

    CMP R1, #0
    B.GE .positive
    ; Négatif: R0 = 0 - R1
    MOV R0, #0
    SUB R0, R0, R1
    B .done

.positive:
    MOV R0, R1

.done:
    HALT
```

1.2.8 Boucles

```
; Boucles - Solution

.text
.global _start
_start:
    MOV R0, #0      ; somme = 0
    MOV R1, #1      ; compteur = 1

.loop:
    ADD R0, R0, R1  ; somme += compteur
    ADD R1, R1, #1  ; compteur++
    CMP R1, #10
    B.LE .loop

    HALT
```

1.2.9 Multiplication

```
; Multiplication - Solution

.text
.global _start
_start:
    MOV R0, #0      ; résultat = 0
    MOV R1, #6      ; multiplicande
    MOV R2, #7      ; compteur (multiplicateur)

.loop:
    ADD R0, R0, R1  ; résultat += multiplicande
    SUB R2, R2, #1  ; compteur--
    CMP R2, #0
    B.GT .loop

    HALT
```

1.2.10 Fibonacci

```
; Fibonacci - Solution

.text
.global _start
_start:
    MOV R0, #1      ; F(1) = 1
    MOV R1, #1      ; F(2) = 1
    MOV R2, #2      ; compteur = 2

.loop:
    ADD R3, R0, R1  ; R3 = F(n-2) + F(n-1)
    MOV R0, R1      ; F(n-2) = ancien F(n-1)
    MOV R1, R3      ; F(n-1) = nouveau F(n)
    ADD R2, R2, #1  ; compteur++
    CMP R2, #10
    B.LT .loop

    MOV R0, R1      ; résultat dans R0
    HALT
```

1.2.11 Tableaux

```
; Tableaux - Solution

.data
data:
    .word 10
    .word 20
    .word 30
    .word 40
    .word 50

.text
```

```
.global _start
_start:
    MOV R0, #0          ; somme = 0
    LDR R1, =data       ; adresse du tableau
    MOV R2, #5          ; compteur = 5

.loop:
    LDR R3, [R1]        ; charge élément
    ADD R0, R0, R3      ; somme += élément
    ADD R1, R1, #4      ; adresse suivante
    SUB R2, R2, #1      ; compteur--
    CMP R2, #0
    B.GT .loop

    HALT
```

1.2.12 Maximum Tableau

```
; Maximum Tableau - Solution

.data
data:
    .word 12
    .word 45
    .word 7
    .word 89
    .word 23

.text
.global _start
_start:
    LDR R1, =data       ; adresse du tableau
    LDR R0, [R1]        ; max = premier élément
    ADD R1, R1, #4      ; passer au suivant
    MOV R2, #4          ; compteur = 4 (reste)

.loop:
    LDR R3, [R1]        ; charge élément
    CMP R3, R0          ; compare avec max
    B.LE .skip
    MOV R0, R3          ; nouveau max

.skip:
    ADD R1, R1, #4      ; adresse suivante
    SUB R2, R2, #1      ; compteur--
    CMP R2, #0
    B.GT .loop

    HALT
```

1.2.13 Mémoire

```
; Mémoire - Solution

.data
data:
    .word 0
    .word 0

.text
.global _start
_start:
    MOV R1, #10
    MOV R2, #20

    ; Sauvegarder
    LDR R4, =data
    STR R1, [R4]        ; data[0] = R1
    ADD R4, R4, #4
    STR R2, [R4]        ; data[1] = R2

    ; Effacer
    MOV R1, #0
    MOV R2, #0

    ; Recharger
    LDR R4, =data
    LDR R1, [R4]        ; R1 = data[0]
    ADD R4, R4, #4
    LDR R2, [R4]        ; R2 = data[1]

    ; Calculer
    ADD R0, R1, R2

    HALT
```

1.2.14 Structure Simple

```
; Structure Simple - Solution

.data
; Structure Point { int x; int y; }
point:
    .word 10    ; x = 10 (offset 0)
    .word 32    ; y = 32 (offset 4)

.text
.global _start
_start:
    ; Charger l'adresse de la structure
    LDR R1, =point

    ; Charger p.x (offset 0)
    LDR R2, [R1]        ; R2 = p.x = 10
```

```
    ; Charger p.y (offset 4)
    LDR R3, [R1, #4]    ; R3 = p.y = 32

    ; Calculer la somme
    ADD R0, R2, R3      ; R0 = 10 + 32 = 42

    HALT
```

1.2.15 Initialiser Structure

```
; Initialiser Structure - Solution

.data
; Structure Point (non initialisée)
point:
    .word 0     ; x (offset 0)
    .word 0     ; y (offset 4)

.text
.global _start
_start:
    LDR R4, =point  ; adresse de la structure

    ; Initialiser p.x = 20
    MOV R1, #20
    STR R1, [R4]        ; p.x = 20

    ; Initialiser p.y = 22
    MOV R2, #22
    STR R2, [R4, #4]    ; p.y = 22

    ; Relire et additionner
    LDR R1, [R4]        ; R1 = p.x
    LDR R2, [R4, #4]    ; R2 = p.y
    ADD R0, R1, R2      ; R0 = 42

    HALT
```

1.2.16 Structure Rectangle

```
; Structure Rectangle - Solution

.data
; Structure Rectangle
rect:
    .word 5     ; x (offset 0)
    .word 10    ; y (offset 4)
    .word 6     ; width (offset 8)
    .word 7     ; height (offset 12)

.text
.global _start
_start:
```

```
    LDR R4, =rect

    ; Charger width et height
    LDR R1, [R4, #8]    ; R1 = width = 6
    LDR R2, [R4, #12]   ; R2 = height = 7

    ; Multiplier par additions: 6 * 7
    MOV R0, #0          ; résultat = 0
.mult:
    CMP R2, #0
    B.EQ .done
    ADD R0, R0, R1      ; résultat += width
    SUB R2, R2, #1      ; height--
    B .mult

.done:
    HALT
```

1.2.17 Tableau de Structures

```
; Tableau de Structures - Solution

.data
; Tableau de 3 Points (8 octets chacun)
points:
    ; points[0]: x=10, y=2
    .word 10
    .word 2
    ; points[1]: x=15, y=5
    .word 15
    .word 5
    ; points[2]: x=8, y=7
    .word 8
    .word 7

.text
.global _start
_start:
    LDR R1, =points     ; adresse du tableau
    MOV R0, #0          ; somme = 0
    MOV R2, #3          ; compteur = 3

.loop:
    CMP R2, #0
    B.EQ .done

    ; Charger x du Point courant
    LDR R3, [R1]        ; R3 = points[i].x
    ADD R0, R0, R3      ; somme += x

    ; Passer au Point suivant (8 octets)
    ADD R1, R1, #8

    ; Décrémenter compteur
    SUB R2, R2, #1
```

```
    B .loop

.done:
    HALT
```

1.2.18 Somme x+y Structures

```
; Somme x+y de Structures - Solution

.data
points:
    ; points[0]: x=5, y=3
    .word 5
    .word 3
    ; points[1]: x=10, y=4
    .word 10
    .word 4
    ; points[2]: x=12, y=8
    .word 12
    .word 8

.text
.global _start
_start:
    LDR R1, =points     ; adresse du tableau
    MOV R0, #0          ; somme totale = 0
    MOV R4, #3          ; compteur = 3

.loop:
    CMP R4, #0
    B.EQ .done

    ; Charger x et y du Point courant
    LDR R2, [R1]        ; R2 = x
    LDR R3, [R1, #4]    ; R3 = y

    ; Ajouter x+y à la somme
    ADD R0, R0, R2      ; somme += x
    ADD R0, R0, R3      ; somme += y

    ; Point suivant (8 octets)
    ADD R1, R1, #8
    SUB R4, R4, #1
    B .loop

.done:
    HALT
```

1.2.19 Fonctions

```
; Fonctions - Solution

.text
```

```
.global _start
_start:
    MOV R0, #21
    BL double
    HALT

double:
    ADD R0, R0, R0  ; R0 = R0 + R0 = 2 * R0
    MOV PC, LR      ; retour
```

1.2.20 Fonction Add3

```
; Fonction Add3 - Solution

.text
.global _start
_start:
    MOV R0, #10
    MOV R1, #15
    MOV R2, #17
    BL add3
    HALT

add3:
    ADD R0, R0, R1
    ADD R0, R0, R2
    MOV PC, LR
```

1.2.21 Écrire Caractère

```
; Écrire Caractère - Solution

.text
.global _start
_start:
    MOV R0, #65         ; 'A' = 65
    LDR R1, =0xFFFF0000 ; adresse PUTC
    STRB R0, [R1]       ; écrire le caractère
    HALT
```

1.2.22 Hello String

```
; Hello String - Solution

.text
.global _start
_start:
    LDR R1, =0xFFFF0000 ; adresse PUTC
    MOV R0, #0          ; compteur

    MOV R2, #72         ; 'H'
    STRB R2, [R1]
```

```
    ADD R0, R0, #1

    MOV R2, #105        ; 'i'
    STRB R2, [R1]
    ADD R0, R0, #1

    HALT
```

1.2.23 Print Loop

```
; Print Loop - Solution

.text
.global _start
_start:
    LDR R1, =0xFFFF0000 ; adresse PUTC
    MOV R0, #0          ; compteur
    MOV R2, #65         ; caractère courant = 'A'

.loop:
    STRB R2, [R1]       ; écrire caractère
    ADD R0, R0, #1      ; compteur++
    ADD R2, R2, #1      ; caractère suivant
    CMP R2, #69         ; 'E' = 69
    B.LT .loop

    HALT
```

1.2.24 Pixel

```
; Pixel - Solution

.text
.global _start
_start:
    LDR R1, =0x00400000 ; adresse écran
    MOV R0, #0x80       ; bit 7 = pixel 0
    STRB R0, [R1]       ; écrire le byte
    HALT
```

1.2.25 Ligne Horizontale

```
; Ligne Horizontale - Solution

.text
.global _start
_start:
    LDR R1, =0x00400000 ; adresse écran
    MOV R0, #0xFF       ; 8 pixels allumés
    STRB R0, [R1]       ; écrire le byte
    HALT
```

1.2.26 Ligne Verticale

```
; Ligne Verticale - Solution

.text
.global _start
_start:
    LDR R1, =0x00400000 ; adresse écran
    MOV R2, #0x80       ; bit du pixel (colonne 0)
    MOV R0, #0          ; compteur

.loop:
    STRB R2, [R1]       ; dessiner pixel
    ADD R1, R1, #40     ; ligne suivante (320/8 = 40)
    ADD R0, R0, #1      ; compteur++
    CMP R0, #8
    B.LT .loop

    HALT
```

1.2.27 Rectangle

```
; Rectangle - Solution

.text
.global _start
_start:
    LDR R1, =0x00400000 ; adresse écran
    MOV R2, #0xFF       ; ligne de 8 pixels
    MOV R0, #0          ; compteur de lignes

.loop:
    STRB R2, [R1]       ; dessiner ligne
    ADD R1, R1, #40     ; ligne suivante
    ADD R0, R0, #1      ; compteur++
    CMP R0, #8
    B.LT .loop

    HALT
```

1.2.28 Damier

```
; Damier - Solution

.text
.global _start
_start:
    LDR R1, =0x00400000 ; adresse écran
    MOV R2, #0xAA       ; motif courant
    MOV R0, #0          ; compteur de lignes
```

```
.loop:
    STRB R2, [R1]       ; colonne 0
    ADD R4, R1, #1
    EOR R5, R2, #0xFF   ; inverser pour colonne 1
    STRB R5, [R4]       ; colonne 1

    ADD R1, R1, #40     ; ligne suivante
    EOR R2, R2, #0xFF   ; alterner le motif
    ADD R0, R0, #1
    CMP R0, #8
    B.LT .loop

    HALT
```

1.2.29 Lire un Caractère

```
; Lire un Caractère - Solution

.text
.global _start
_start:
    LDR R1, =0x00402600 ; adresse KEYBOARD (temps réel)
    LDR R4, =0xFFFF0000 ; adresse PUTC

    ; Attendre une touche
.wait:
    LDR R2, [R1]        ; lire clavier
    CMP R2, #0
    B.EQ .wait          ; boucler si pas de touche

    ; Convertir ASCII → nombre
    SUB R0, R2, #0x30   ; R0 = chiffre (0-9)

    ; Afficher le chiffre tapé (écho)
    STR R2, [R4]        ; afficher le caractère

    HALT
```

1.2.30 Lire un Nombre à 2 Chiffres

```
; Lire un Nombre à 2 Chiffres - Solution

.text
.global _start
_start:
    LDR R5, =0x00402600 ; adresse KEYBOARD (temps réel)
    LDR R6, =0xFFFF0000 ; adresse PUTC

    ; Attendre premier chiffre
.wait1:
    LDR R1, [R5]
    CMP R1, #0
    B.EQ .wait1
```

```
    ; Écho du premier chiffre
    STR R1, [R6]

    ; Convertir en nombre
    SUB R1, R1, #0x30   ; R1 = premier chiffre

    ; Multiplier par 10: x → 2x → 4x → 5x → 10x
    ADD R2, R1, R1      ; R2 = 2x
    ADD R2, R2, R2      ; R2 = 4x
    ADD R2, R2, R1      ; R2 = 5x
    ADD R1, R2, R2      ; R1 = 10x

    ; Attendre relâche de la touche
.release1:
    LDR R2, [R5]
    CMP R2, #0
    B.NE .release1

    ; Attendre deuxième chiffre
.wait2:
    LDR R2, [R5]
    CMP R2, #0
    B.EQ .wait2

    ; Écho du deuxième chiffre
    STR R2, [R6]

    ; Convertir et ajouter
    SUB R2, R2, #0x30   ; R2 = deuxième chiffre
    ADD R0, R1, R2      ; R0 = nombre complet

    HALT
```

1.2.31 Deviner le Nombre

```
; Deviner le Nombre - Solution

.text
.global _start
_start:
    MOV R7, #55         ; nombre secret: '7' en ASCII (0x37)
    LDR R5, =0x00402600 ; KEYBOARD
    LDR R6, =0xFFFF0000 ; PUTC
    MOV R4, #0          ; dernière touche vue

.game_loop:
    ; Lire clavier
    LDR R0, [R5]

    ; Ignorer si pas de touche ou même touche que avant
    CMP R0, #0
    B.EQ .game_loop
    CMP R0, R4
    B.EQ .game_loop
```

```
    ; Nouvelle touche détectée
    MOV R4, R0          ; sauvegarder

    ; Écho de la touche
    STR R0, [R6]

    ; Comparer au secret
    CMP R0, R7
    B.EQ .win
    B.LT .too_small

    ; Trop grand -> afficher '-'
    MOV R1, #45         ; '-'
    STR R1, [R6]
    MOV R1, #10         ; newline
    STR R1, [R6]
    B .wait_release

.too_small:
    ; Trop petit -> afficher '+'
    MOV R1, #43         ; '+'
    STR R1, [R6]
    MOV R1, #10         ; newline
    STR R1, [R6]

.wait_release:
    ; Attendre relâche (R4 != 0, donc on attend que R0 change)
    LDR R0, [R5]
    CMP R0, R4
    B.EQ .wait_release
    MOV R4, #0          ; reset
    B .game_loop

.win:
    ; Gagné! Afficher '*'
    MOV R1, #42         ; '*'
    STR R1, [R6]
    MOV R1, #10
    STR R1, [R6]
    MOV R0, #7          ; résultat dans R0

    HALT
```

1.2.32 Dégradé (Dithering)

```
; Dégradé (Dithering) - Solution

.text
.global _start
_start:
    LDR R1, =0x00400000 ; adresse écran
    MOV R0, #0          ; compteur lignes

.line_loop:
```

```
    ; Écrire les 5 motifs de dégradé
    MOV R2, #0x00       ; noir
    STRB R2, [R1]

    MOV R2, #0x11       ; 12.5% blanc
    ADD R3, R1, #1
    STRB R2, [R3]

    MOV R2, #0x55       ; 50% blanc
    ADD R3, R1, #2
    STRB R2, [R3]

    MOV R2, #0xBB       ; 75% blanc
    ADD R3, R1, #3
    STRB R2, [R3]

    MOV R2, #0xFF       ; blanc
    ADD R3, R1, #4
    STRB R2, [R3]

    ; Ligne suivante
    ADD R1, R1, #40     ; 40 bytes par ligne
    ADD R0, R0, #1
    CMP R0, #240
    B.LT .line_loop

    HALT
```

1.2.33 Dégradé Plein Écran

```
; Dégradé Plein Écran - Solution
; 8 bandes avec motifs croissants

.text
.global _start
_start:
    LDR R1, =0x00400000 ; adresse écran
    MOV R0, #0          ; compteur bandes
    MOV R2, #0x00       ; motif initial (noir)

next_band:
    LDR R3, =1200       ; 30 lignes * 40 bytes

fill_band:
    STRB R2, [R1]
    ADD R1, R1, #1
    SUB R3, R3, #1
    CMP R3, #0
    B.GT fill_band

    ; Passer au motif suivant (approximation)
    ; 0x00 -> 0x11 -> 0x22 -> 0x33 -> ...
    ADD R2, R2, #0x11
    ADD R0, R0, #1
    CMP R0, #8
```

```
    B.LT next_band

    HALT
```

1.2.34 Recherche Dichotomique

```
; Recherche Dichotomique - Solution

.text
.global _start
_start:
    MOV R5, #42         ; nombre secret
    MOV R1, #0          ; low
    MOV R2, #100        ; high
    MOV R0, #0          ; compteur d'essais

.loop:
    ADD R0, R0, #1      ; compteur++

    ; mid = (low + high) / 2
    ADD R3, R1, R2
    MOV R4, R3
    ; Division par 2 avec shift (simulé par soustraction successive)
    MOV R3, #0
.div2:
    CMP R4, #2
    B.LT .div_done
    SUB R4, R4, #2
    ADD R3, R3, #1
    B .div2
.div_done:
    ; R3 = mid

    CMP R3, R5
    B.EQ .found
    B.LT .too_low

    ; mid > secret: high = mid - 1
    SUB R2, R3, #1
    B .loop

.too_low:
    ; mid < secret: low = mid + 1
    ADD R1, R3, #1
    B .loop

.found:
    HALT
```

1.3 C. Solutions C32

1.3.1 Variables

```c
// Variables - Solution

int main() {
    int x = 10;
    int y = 32;
    int result = x + y;
    return result;
}
```

1.3.2 Expressions

```c
// Expressions - Solution

int main() {
    int result = (5 + 3) * (10 - 4) / 2;
    return result;
}
```

1.3.3 Modulo

```c
// Modulo - Solution

int main() {
    int result = (100 % 7) + (45 % 8);
    return result;
}
```

1.3.4 Incrémentation

```c
// Incrémentation - Solution

int main() {
    int x = 5;

    x = x + 3;   // x = 8
    x = x * 2;   // x = 16
    x = x - 1;   // x = 15

    return x;
}
```

1.3.5 Conditions

```c
// Conditions - Solution

int main() {
    int a = 25;
```

```
    int b = 17;
    int max;

    if (a > b) {
        max = a;
    } else {
        max = b;
    }

    return max;
}
```

1.3.6 Else-If

```
// Else-If - Solution

int main() {
    int score = 75;
    int grade;

    if (score >= 90) {
        grade = 5;
    } else if (score >= 80) {
        grade = 4;
    } else if (score >= 70) {
        grade = 3;
    } else if (score >= 60) {
        grade = 2;
    } else {
        grade = 1;
    }

    return grade;
}
```

1.3.7 Opérateurs Logiques

```
// Opérateurs Logiques - Solution

int main() {
    int x = 15;
    int result;

    if (x >= 10 && x <= 20) {
        result = 1;
    } else {
        result = 0;
    }

    return result;
}
```

1.3.8 Maximum de 3

```c
// Maximum de 3 - Solution

int main() {
    int a = 15;
    int b = 42;
    int c = 27;
    int max;

    if (a >= b && a >= c) {
        max = a;
    } else if (b >= c) {
        max = b;
    } else {
        max = c;
    }

    return max;
}
```

1.3.9 Boucle For

```c
// Boucle For - Solution

int main() {
    int sum = 0;

    for (int i = 1; i <= 10; i = i + 1) {
        sum = sum + i;
    }

    return sum;
}
```

1.3.10 Boucle While

```c
// Boucle While - Solution

int main() {
    int n = 12345;
    int count = 0;

    while (n > 0) {
        count = count + 1;
        n = n / 10;
    }

    return count;
}
```

1.3.11 Boucles Imbriquées

```c
// Boucles Imbriquées - Solution

int main() {
    int sum = 0;

    for (int i = 1; i <= 3; i = i + 1) {
        for (int j = 1; j <= 4; j = j + 1) {
            sum = sum + i * j;
        }
    }

    return sum;
}
```

1.3.12 Multiplication

```c
// Multiplication - Solution

int main() {
    int a = 7;
    int b = 8;
    int result = 0;

    for (int i = 0; i < b; i = i + 1) {
        result = result + a;
    }

    return result;
}
```

1.3.13 Fonctions

```c
// Fonctions - Solution

int square(int n) {
    return n * n;
}

int main() {
    return square(7);
}
```

1.3.14 Paramètres Multiples

```c
// Paramètres Multiples - Solution

int add3(int a, int b, int c) {
    return a + b + c;
}
```

```c
int main() {
    return add3(10, 20, 12);
}
```

1.3.15 Valeur Absolue

```c
// Valeur Absolue - Solution

int abs(int x) {
    if (x < 0) {
        return -x;
    }
    return x;
}

int main() {
    return abs(-15) + abs(10);
}
```

1.3.16 Min et Max

```c
// Min et Max - Solution

int min(int a, int b) {
    if (a < b) {
        return a;
    }
    return b;
}

int max(int a, int b) {
    if (a > b) {
        return a;
    }
    return b;
}

int main() {
    return max(10, 25) - min(10, 25);
}
```

1.3.17 Tableaux

```c
// Tableaux - Solution

int main() {
    int arr[5];
    arr[0] = 3;
    arr[1] = 7;
    arr[2] = 2;
    arr[3] = 9;
    arr[4] = 5;
```

```
    int sum = 0;

    for (int i = 0; i < 5; i = i + 1) {
        sum = sum + arr[i];
    }

    return sum;
}
```

1.3.18 Maximum Tableau

```
// Maximum Tableau - Solution

int main() {
    int arr[6];
    arr[0] = 12;
    arr[1] = 45;
    arr[2] = 7;
    arr[3] = 23;
    arr[4] = 56;
    arr[5] = 34;

    int max = arr[0];

    for (int i = 1; i < 6; i = i + 1) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }

    return max;
}
```

1.3.19 Compter Éléments

```
// Compter Éléments - Solution

int main() {
    int arr[8];
    arr[0] = 3;
    arr[1] = 8;
    arr[2] = 2;
    arr[3] = 7;
    arr[4] = 4;
    arr[5] = 9;
    arr[6] = 6;
    arr[7] = 1;

    int count = 0;

    for (int i = 0; i < 8; i = i + 1) {
        if (arr[i] % 2 == 0) {
```

```
            count = count + 1;
        }
    }

    return count;
}
```

1.3.20 Pointeurs

```
// Pointeurs - Solution

int main() {
    int x = 10;
    int *p = &x;
    *p = 42;
    return x;
}
```

1.3.21 Swap

```
// Swap - Solution

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 10;
    int y = 20;

    swap(&x, &y);

    return x;
}
```

1.3.22 Pointeurs et Tableaux

```
// Pointeurs et Tableaux - Solution

int main() {
    int arr[4];
    arr[0] = 5;
    arr[1] = 10;
    arr[2] = 15;
    arr[3] = 20;

    int sum = 0;
    int *p = arr;

    for (int i = 0; i < 4; i = i + 1) {
```

```
        sum = sum + *(p + i);
    }

    return sum;
}
```

1.3.23 Opérations Binaires

```
// Opérations Binaires - Solution

int main() {
    int x = 10;
    int y = 12;

    int result = (x & y) | (x ^ y);
    return result;
}
```

1.3.24 Puissance de 2

```
// Puissance de 2 - Solution

int is_pow2(int n) {
    if (n <= 0) {
        return 0;
    }
    return (n & (n - 1)) == 0;
}

int main() {
    return is_pow2(16) + is_pow2(15) + is_pow2(32);
}
```

1.3.25 Factorielle

```
// Factorielle - Solution

int fact(int n) {
    if (n <= 1) {
        return 1;
    }
    return n * fact(n - 1);
}

int main() {
    return fact(5);
}
```

1.3.26 Fibonacci

```c
// Fibonacci - Solution

int fib(int n) {
    if (n <= 0) {
        return 0;
    }
    if (n == 1) {
        return 1;
    }
    return fib(n - 1) + fib(n - 2);
}

int main() {
    return fib(10);
}
```

1.3.27 Somme Récursive

```c
// Somme Récursive - Solution

int sum(int n) {
    if (n <= 0) {
        return 0;
    }
    return n + sum(n - 1);
}

int main() {
    return sum(10);
}
```

1.3.28 PGCD (Euclide)

```c
// PGCD (Euclide) - Solution

int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}

int main() {
    return gcd(48, 18);
}
```

1.3.29 Puissance

```c
// Puissance - Solution
```

```c
int power(int x, int n) {
    if (n == 0) {
        return 1;
    }
    if (n % 2 == 0) {
        return power(x * x, n / 2);
    }
    return x * power(x, n - 1);
}

int main() {
    return power(2, 10);
}
```

1.3.30 Test Primalité

```c
// Test Primalité - Solution

int is_prime(int n) {
    if (n < 2) {
        return 0;
    }
    for (int i = 2; i * i <= n; i = i + 1) {
        if (n % i == 0) {
            return 0;
        }
    }
    return 1;
}

int main() {
    int count = 0;
    for (int n = 2; n <= 20; n = n + 1) {
        if (is_prime(n)) {
            count = count + 1;
        }
    }
    return count;
}
```

1.3.31 Tri à Bulles

```c
// Tri à Bulles - Solution

int main() {
    int arr[5];
    arr[0] = 64;
    arr[1] = 34;
    arr[2] = 25;
    arr[3] = 12;
    arr[4] = 22;

    for (int i = 0; i < 5; i = i + 1) {
```

```
        for (int j = 0; j < 4 - i; j = j + 1) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }

    return arr[0];
}
```

1.3.32 Recherche Binaire

```
// Recherche Binaire - Solution

int binary_search(int *arr, int size, int target) {
    int left = 0;
    int right = size - 1;

    while (left <= right) {
        int mid = (left + right) / 2;
        if (arr[mid] == target) {
            return mid;
        }
        if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return -1;
}

int main() {
    int arr[10];
    arr[0] = 2;
    arr[1] = 5;
    arr[2] = 8;
    arr[3] = 12;
    arr[4] = 16;
    arr[5] = 23;
    arr[6] = 38;
    arr[7] = 56;
    arr[8] = 72;
    arr[9] = 91;

    return binary_search(arr, 10, 23);
}
```

1.3.33 Inverser Tableau

```c
// Inverser Tableau - Solution

int main() {
    int arr[5];
    arr[0] = 1;
    arr[1] = 2;
    arr[2] = 3;
    arr[3] = 4;
    arr[4] = 5;

    int left = 0;
    int right = 4;
    while (left < right) {
        int temp = arr[left];
        arr[left] = arr[right];
        arr[right] = temp;
        left = left + 1;
        right = right - 1;
    }

    int sum = 0;
    for (int i = 0; i < 5; i = i + 1) {
        sum = sum + arr[i] * (i + 1);
    }
    return sum;
}
```

1.3.34 Somme des Chiffres

```c
// Somme des Chiffres - Solution

int digit_sum(int n) {
    int sum = 0;
    while (n > 0) {
        sum = sum + (n % 10);
        n = n / 10;
    }
    return sum;
}

int main() {
    return digit_sum(12345);
}
```

1.3.35 Nombre Palindrome

```c
// Nombre Palindrome - Solution

int is_palindrome(int n) {
    int original = n;
    int reversed = 0;
```

```
    while (n > 0) {
        reversed = reversed * 10 + (n % 10);
        n = n / 10;
    }

    if (reversed == original) {
        return 1;
    }
    return 0;
}

int main() {
    return is_palindrome(12321) + is_palindrome(1221) + is_palindrome(123);
}
```

1.3.36 Définition Struct

```
// Définition Struct - Solution

struct Point { int x; int y; };

int main() {
    struct Point p;
    p.x = 17;
    p.y = 25;
    return p.x + p.y;
}
```

1.3.37 Pointeur Struct

```
// Pointeur Struct - Solution

struct Point { int x; int y; };

int main() {
    struct Point p;
    p.x = 10;
    p.y = 32;

    struct Point *ptr = &p;
    return ptr->x + ptr->y;
}
```

1.3.38 Struct et Fonctions

```
// Struct et Fonctions - Solution

struct Point { int x; int y; };

int distance_sq(struct Point *p) {
    return p->x * p->x + p->y * p->y;
}
```

```c
int main() {
    struct Point p;
    p.x = 3;
    p.y = 4;
    return distance_sq(&p);
}
```

1.3.39 Structs Imbriquées

```c
// Structs Imbriquées - Solution

struct Point { int x; int y; };
struct Rectangle { struct Point corner; int width; int height; };

int main() {
    struct Rectangle r;
    r.corner.x = 0;
    r.corner.y = 0;
    r.width = 6;
    r.height = 7;

    return r.width * r.height;
}
```

1.3.40 Tableau de Structs

```c
// Tableau de Structs - Solution

struct Point { int x; int y; };

int main() {
    struct Point points[3];

    points[0].x = 10;
    points[0].y = 2;
    points[1].x = 15;
    points[1].y = 5;
    points[2].x = 8;
    points[2].y = 2;

    int sum = 0;
    for (int i = 0; i < 3; i = i + 1) {
        sum = sum + points[i].x;
    }
    return sum;
}
```

1.3.41 Sizeof Struct

```c
// Sizeof Struct - Solution
```

```c
struct S1 { int a; };
struct S2 { int x; int y; char c; };

int main() {
    return sizeof(struct S1) + sizeof(struct S2);
}
```

1.3.42 Écrire un Caractère

```c
// Écrire un Caractère - Solution

void putchar(int c) {
    int *port = (int*)0xFFFF0000;
    *port = c;
}

int main() {
    putchar(65);  // Affiche 'A'
    return 65;
}
```

1.3.43 Afficher une Chaîne

```c
// Afficher une Chaîne - Solution

void putchar(int c) {
    int *port = (int*)0xFFFF0000;
    *port = c;
}

int print(char *s) {
    int count = 0;
    while (*s) {
        putchar(*s);
        s = s + 1;
        count = count + 1;
    }
    return count;
}

int main() {
    return print("HI");
}
```

1.3.44 Afficher un Nombre

```c
// Afficher un Nombre - Solution

void putchar(int c) {
    int *port = (int*)0xFFFF0000;
    *port = c;
}
```

```c
void print_int(int n) {
    char buf[12];
    int i = 0;

    if (n == 0) {
        putchar(48);
        return;
    }

    while (n > 0) {
        buf[i] = 48 + (n % 10);
        n = n / 10;
        i = i + 1;
    }

    while (i > 0) {
        i = i - 1;
        putchar(buf[i]);
    }
}

int main() {
    print_int(42);
    return 42;
}
```

1.3.45 Dessiner un Pixel

```c
// Dessiner un Pixel - Solution

int main() {
    char *screen = (char*)0x00400000;
    *screen = 0x80;  // 0b10000000 - allume le pixel 0
    return 128;
}
```

1.3.46 Ligne Horizontale

```c
// Ligne Horizontale - Solution

int main() {
    char *screen = (char*)0x00400000;

    screen[0] = 0xFF;  // 8 premiers pixels
    screen[1] = 0xFF;  // 8 pixels suivants

    return 16;
}
```

1.3.47 Dessiner un Rectangle

```
// Dessiner un Rectangle - Solution

int main() {
    char *screen = (char*)0x00400000;

    for (int y = 0; y < 8; y = y + 1) {
        screen[y * 40] = 0xFF;  // 40 bytes par ligne
    }

    return 64;
}
```

1.3.48 Crible d

```
// Crible d'Ératosthène - Solution

int main() {
    int is_prime[51];
    int i;
    int j;
    int count;

    for (i = 0; i <= 50; i = i + 1) {
        is_prime[i] = 1;
    }
    is_prime[0] = 0;
    is_prime[1] = 0;

    for (i = 2; i * i <= 50; i = i + 1) {
        if (is_prime[i]) {
            for (j = i * i; j <= 50; j = j + i) {
                is_prime[j] = 0;
            }
        }
    }

    count = 0;
    for (i = 2; i <= 50; i = i + 1) {
        if (is_prime[i]) {
            count = count + 1;
        }
    }
    return count;
}
```

1.3.49 Suite de Collatz

```
// Suite de Collatz - Solution

int collatz_length(int n) {
    int count;
```

```
    count = 1;
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
        count = count + 1;
    }
    return count;
}

int main() {
    return collatz_length(27);  // attendu: 112
}
```

1.3.50 Projet Final

```
// Projet Final - Solution

int sum_divisors(int n) {
    int sum = 0;
    for (int i = 1; i < n; i = i + 1) {
        if (n % i == 0) {
            sum = sum + i;
        }
    }
    return sum;
}

int main() {
    return sum_divisors(28);
}
```

1.4 D. Solutions Construction du Compilateur

1.4.1 1.1 Reconnaître un Chiffre

```
int is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main() {
    int score = 0;
    if (is_digit('0') == 1) score = score + 1;
    if (is_digit('5') == 1) score = score + 1;
    if (is_digit('9') == 1) score = score + 1;
    if (is_digit('a') == 0) score = score + 1;
    if (is_digit(' ') == 0) score = score + 1;
    return score;
}
```

1.4.2 1.2 Lire un Nombre

```
int is_digit(char c) { return c >= '0' && c <= '9'; }

int parse_number(char* s, int* pos) {
    int result = 0;
    while (is_digit(s[*pos])) {
        result = result * 10 + (s[*pos] - '0');
        *pos = *pos + 1;
    }
    return result;
}

int main() {
    int score = 0;
    int p;

    p = 0;
    if (parse_number("42", &p) == 42 && p == 2) score = score + 1;

    p = 0;
    if (parse_number("123+45", &p) == 123 && p == 3) score = score + 1;

    p = 4;
    if (parse_number("123+45", &p) == 45 && p == 6) score = score + 1;

    p = 0;
    if (parse_number("7", &p) == 7 && p == 1) score = score + 1;

    return score;
}
```

1.4.3 1.3 Identifier les Tokens

```
int is_digit(char c) { return c >= '0' && c <= '9'; }

int next_token(char* s, int* pos) {
    while (s[*pos] == ' ') *pos = *pos + 1;

    char c = s[*pos];
    if (c == 0) return 0;

    if (is_digit(c)) {
        while (is_digit(s[*pos])) *pos = *pos + 1;
        return 1;
    }

    *pos = *pos + 1;
    if (c == '+') return 2;
    if (c == '-') return 3;
    if (c == '*') return 4;
    if (c == '/') return 5;
    if (c == '(') return 6;
    if (c == ')') return 7;
```

```
    return 0;
}

int main() {
    int score = 0;
    int p;

    p = 0;
    if (next_token("42", &p) == 1) score = score + 1;

    p = 0;
    if (next_token("+", &p) == 2) score = score + 1;

    p = 0;
    if (next_token("3 + 5", &p) == 1) score = score + 1;
    if (next_token("3 + 5", &p) == 2) score = score + 1;
    if (next_token("3 + 5", &p) == 1) score = score + 1;
    if (next_token("3 + 5", &p) == 0) score = score + 1;

    return score;
}
```

1.4.4 2.1 Évaluer a + b

```
int is_digit(char c) { return c >= '0' && c <= '9'; }

int parse_num(char* s, int* p) {
    while (s[*p] == ' ') *p = *p + 1;
    int v = 0;
    while (is_digit(s[*p])) {
        v = v * 10 + (s[*p] - '0');
        *p = *p + 1;
    }
    return v;
}

int eval_simple(char* s) {
    int pos = 0;
    int a = parse_num(s, &pos);

    while (s[pos] == ' ') pos = pos + 1;
    char op = s[pos];
    pos = pos + 1;

    int b = parse_num(s, &pos);

    if (op == '+') return a + b;
    if (op == '-') return a - b;
    if (op == '*') return a * b;
    if (op == '/') return a / b;
    return 0;
}

int main() {
    int score = 0;
```

```
    if (eval_simple("3 + 5") == 8) score = score + 1;
    if (eval_simple("10 - 4") == 6) score = score + 1;
    if (eval_simple("6 * 7") == 42) score = score + 1;
    if (eval_simple("20 / 4") == 5) score = score + 1;
    return score;
}
```

1.4.5 2.2 Évaluer a + b + c

```
int is_digit(char c) { return c >= '0' && c <= '9'; }

int parse_num(char* s, int* p) {
    while (s[*p] == ' ') *p = *p + 1;
    int v = 0;
    while (is_digit(s[*p])) { v = v * 10 + (s[*p] - '0'); *p = *p + 1; }
    return v;
}

int is_op(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}

int eval_chain(char* s) {
    int pos = 0;
    int result = parse_num(s, &pos);

    while (1) {
        while (s[pos] == ' ') pos = pos + 1;
        char op = s[pos];
        if (!is_op(op)) break;
        pos = pos + 1;
        int b = parse_num(s, &pos);

        if (op == '+') result = result + b;
        else if (op == '-') result = result - b;
        else if (op == '*') result = result * b;
        else if (op == '/') result = result / b;
    }

    return result;
}

int main() {
    int score = 0;
    if (eval_chain("5") == 5) score = score + 1;
    if (eval_chain("3 + 5") == 8) score = score + 1;
    if (eval_chain("1 + 2 + 3") == 6) score = score + 1;
    if (eval_chain("10 - 2 - 3") == 5) score = score + 1;
    if (eval_chain("2 * 3 * 4") == 24) score = score + 1;
    return score;
}
```

1.4.6 2.3 Respecter la Précédence

```
int is_digit(char c) { return c >= '0' && c <= '9'; }
char* input;
int pos;

void skip() { while (input[pos] == ' ') pos = pos + 1; }

int parse_num() {
    skip();
    int v = 0;
    while (is_digit(input[pos])) { v = v * 10 + (input[pos] - '0'); pos = pos + 1; }
    return v;
}

int parse_factor() {
    return parse_num();
}

int parse_term() {
    int left = parse_factor();
    while (1) {
        skip();
        char op = input[pos];
        if (op != '*' && op != '/') break;
        pos = pos + 1;
        int right = parse_factor();
        if (op == '*') left = left * right;
        else left = left / right;
    }
    return left;
}

int parse_expr() {
    int left = parse_term();
    while (1) {
        skip();
        char op = input[pos];
        if (op != '+' && op != '-') break;
        pos = pos + 1;
        int right = parse_term();
        if (op == '+') left = left + right;
        else left = left - right;
    }
    return left;
}

int eval(char* s) {
    input = s;
    pos = 0;
    return parse_expr();
}

int main() {
    int score = 0;
    if (eval("42") == 42) score = score + 1;
    if (eval("3 + 5") == 8) score = score + 1;
```

```
    if (eval("3 * 4") == 12) score = score + 1;
    if (eval("2 + 3 * 4") == 14) score = score + 1;
    if (eval("2 * 3 + 4") == 10) score = score + 1;
    if (eval("10 - 2 * 3") == 4) score = score + 1;
    return score;
}
```

1.4.7 2.4 Gérer les Parenthèses

```
int is_digit(char c) { return c >= '0' && c <= '9'; }
char* input;
int pos;

void skip() { while (input[pos] == ' ') pos = pos + 1; }

int parse_expr();

int parse_num() {
    skip();
    int v = 0;
    while (is_digit(input[pos])) { v = v * 10 + (input[pos] - '0'); pos = pos + 1; }
    return v;
}

int parse_factor() {
    skip();
    if (input[pos] == '(') {
        pos = pos + 1;
        int v = parse_expr();
        skip();
        pos = pos + 1;  // ')'
        return v;
    }
    return parse_num();
}

int parse_term() {
    int left = parse_factor();
    while (1) {
        skip();
        char op = input[pos];
        if (op != '*' && op != '/') break;
        pos = pos + 1;
        int right = parse_factor();
        if (op == '*') left = left * right;
        else left = left / right;
    }
    return left;
}

int parse_expr() {
    int left = parse_term();
    while (1) {
        skip();
        char op = input[pos];
```

```
        if (op != '+' && op != '-') break;
        pos = pos + 1;
        int right = parse_term();
        if (op == '+') left = left + right;
        else left = left - right;
    }
    return left;
}

int eval(char* s) { input = s; pos = 0; return parse_expr(); }

int main() {
    int score = 0;
    if (eval("(5)") == 5) score = score + 1;
    if (eval("(2 + 3)") == 5) score = score + 1;
    if (eval("(2 + 3) * 4") == 20) score = score + 1;
    if (eval("2 * (3 + 4)") == 14) score = score + 1;
    if (eval("(1 + 2) * (3 + 4)") == 21) score = score + 1;
    if (eval("((2))") == 2) score = score + 1;
    return score;
}
```

1.4.8 3.1 Émettre MOV

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }

void append(char* buf, char* s) {
    int i = strlen(buf);
    int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; }
    buf[i] = 0;
}

void append_num(char* buf, int n) {
    char tmp[12];
    int i = 0;
    if (n == 0) { tmp[i] = '0'; i = i + 1; }
    else {
        int rev[12]; int r = 0;
        while (n > 0) { rev[r] = n % 10; r = r + 1; n = n / 10; }
        while (r > 0) { r = r - 1; tmp[i] = '0' + rev[r]; i = i + 1; }
    }
    tmp[i] = 0;
    append(buf, tmp);
}

void emit_mov(char* buf, int reg, int val) {
    append(buf, "MOV R");
    append_num(buf, reg);
    append(buf, ", #");
    append_num(buf, val);
    append(buf, "\\n");
}

int check(char* a, char* b) {
```

```
    int i = 0;
    while (a[i] && b[i]) { if (a[i] != b[i]) return 0; i = i + 1; }
    return a[i] == b[i];
}

int main() {
    char buf[100];
    int score = 0;

    buf[0] = 0;
    emit_mov(buf, 0, 42);
    if (check(buf, "MOV R0, #42\\n")) score = score + 1;

    buf[0] = 0;
    emit_mov(buf, 1, 5);
    if (check(buf, "MOV R1, #5\\n")) score = score + 1;

    buf[0] = 0;
    emit_mov(buf, 0, 0);
    if (check(buf, "MOV R0, #0\\n")) score = score + 1;

    return score;
}
```

1.4.9 3.2 Émettre ADD/SUB/MUL

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; }
    buf[i] = 0;
}
void append_num(char* buf, int n) {
    if (n == 0) { append(buf, "0"); return; }
    char tmp[12]; int i = 11; tmp[11] = 0;
    while (n > 0) { i = i - 1; tmp[i] = '0' + (n % 10); n = n / 10; }
    append(buf, tmp + i);
}

void emit_op(char* buf, char op, int rd, int rn, int rm) {
    if (op == '+') append(buf, "ADD R");
    else if (op == '-') append(buf, "SUB R");
    else if (op == '*') append(buf, "MUL R");
    append_num(buf, rd);
    append(buf, ", R");
    append_num(buf, rn);
    append(buf, ", R");
    append_num(buf, rm);
    append(buf, "\\n");
}

int check(char* a, char* b) {
    int i = 0;
    while (a[i] && b[i]) { if (a[i] != b[i]) return 0; i = i + 1; }
    return a[i] == b[i];
```

```
}

int main() {
    char buf[100];
    int score = 0;

    buf[0] = 0;
    emit_op(buf, '+', 0, 1, 2);
    if (check(buf, "ADD R0, R1, R2\\n")) score = score + 1;

    buf[0] = 0;
    emit_op(buf, '-', 0, 0, 1);
    if (check(buf, "SUB R0, R0, R1\\n")) score = score + 1;

    buf[0] = 0;
    emit_op(buf, '*', 2, 3, 4);
    if (check(buf, "MUL R2, R3, R4\\n")) score = score + 1;

    return score;
}
```

1.4.10 3.3 Émettre PUSH/POP

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; }
    buf[i] = 0;
}
void append_num(char* buf, int n) {
    if (n == 0) { append(buf, "0"); return; }
    char tmp[12]; int i = 11; tmp[11] = 0;
    while (n > 0) { i = i - 1; tmp[i] = '0' + (n % 10); n = n / 10; }
    append(buf, tmp + i);
}

void emit_push(char* buf, int reg) {
    append(buf, "STR R");
    append_num(buf, reg);
    append(buf, ", [SP, #-4]!\\n");
}

void emit_pop(char* buf, int reg) {
    append(buf, "LDR R");
    append_num(buf, reg);
    append(buf, ", [SP], #4\\n");
}

int check(char* a, char* b) {
    int i = 0;
    while (a[i] && b[i]) { if (a[i] != b[i]) return 0; i = i + 1; }
    return a[i] == b[i];
}

int main() {
```

```
    char buf[100];
    int score = 0;

    buf[0] = 0;
    emit_push(buf, 0);
    if (check(buf, "STR R0, [SP, #-4]!\\n")) score = score + 1;

    buf[0] = 0;
    emit_pop(buf, 1);
    if (check(buf, "LDR R1, [SP], #4\\n")) score = score + 1;

    buf[0] = 0;
    emit_push(buf, 0);
    emit_pop(buf, 1);
    if (check(buf, "STR R0, [SP, #-4]!\\nLDR R1, [SP], #4\\n")) score = score + 1;

    return score;
}
```

1.4.11 4.1 Compiler une Constante

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; }
    buf[i] = 0;
}
void append_num(char* buf, int n) {
    if (n == 0) { append(buf, "0"); return; }
    char tmp[12]; int i = 11; tmp[11] = 0;
    while (n > 0) { i = i - 1; tmp[i] = '0' + (n % 10); n = n / 10; }
    append(buf, tmp + i);
}
void emit_mov(char* buf, int reg, int val) {
    append(buf, "MOV R"); append_num(buf, reg);
    append(buf, ", #"); append_num(buf, val); append(buf, "\\n");
}

int is_digit(char c) { return c >= '0' && c <= '9'; }
char* input;
int pos;

int parse_num() {
    int v = 0;
    while (is_digit(input[pos])) { v = v * 10 + (input[pos] - '0'); pos = pos + 1; }
    return v;
}

void codegen_const(char* buf) {
    int n = parse_num();
    emit_mov(buf, 0, n);
}

int check(char* a, char* b) {
    int i = 0;
```

```
    while (a[i] && b[i]) { if (a[i] != b[i]) return 0; i = i + 1; }
    return a[i] == b[i];
}

int main() {
    char buf[100];
    int score = 0;

    buf[0] = 0; input = "42"; pos = 0;
    codegen_const(buf);
    if (check(buf, "MOV R0, #42\\n")) score = score + 1;

    buf[0] = 0; input = "5"; pos = 0;
    codegen_const(buf);
    if (check(buf, "MOV R0, #5\\n")) score = score + 1;

    buf[0] = 0; input = "123"; pos = 0;
    codegen_const(buf);
    if (check(buf, "MOV R0, #123\\n")) score = score + 1;

    return score;
}
```

1.4.12 4.2 Compiler a + b

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; } buf[i] = 0;
}
void append_num(char* buf, int n) {
    if (n == 0) { append(buf, "0"); return; }
    char tmp[12]; int i = 11; tmp[11] = 0;
    while (n > 0) { i = i - 1; tmp[i] = '0' + (n % 10); n = n / 10; }
    append(buf, tmp + i);
}
void emit_mov(char* buf, int reg, int val) {
    append(buf, "MOV R"); append_num(buf, reg);
    append(buf, ", #"); append_num(buf, val); append(buf, "\\n");
}
void emit_push(char* buf, int reg) {
    append(buf, "STR R"); append_num(buf, reg); append(buf, ", [SP, #-4]!\\n");
}
void emit_pop(char* buf, int reg) {
    append(buf, "LDR R"); append_num(buf, reg); append(buf, ", [SP], #4\\n");
}
void emit_op(char* buf, char op, int rd, int rn, int rm) {
    if (op == '+') append(buf, "ADD R");
    else if (op == '-') append(buf, "SUB R");
    else if (op == '*') append(buf, "MUL R");
    append_num(buf, rd); append(buf, ", R"); append_num(buf, rn);
    append(buf, ", R"); append_num(buf, rm); append(buf, "\\n");
}

int is_digit(char c) { return c >= '0' && c <= '9'; }
```

```
char* input;
int pos;
void skip() { while (input[pos] == ' ') pos = pos + 1; }
int parse_num() {
    skip();
    int v = 0;
    while (is_digit(input[pos])) { v = v * 10 + (input[pos] - '0'); pos = pos + 1; }
    return v;
}

void codegen_binop(char* buf) {
    int a = parse_num();
    emit_mov(buf, 0, a);
    emit_push(buf, 0);

    skip();
    char op = input[pos];
    pos = pos + 1;

    int b = parse_num();
    emit_mov(buf, 0, b);
    emit_pop(buf, 1);
    emit_op(buf, op, 0, 1, 0);
}

int contains(char* buf, char* sub) {
    int i = 0;
    while (buf[i]) {
        int j = 0;
        while (sub[j] && buf[i+j] == sub[j]) j = j + 1;
        if (sub[j] == 0) return 1;
        i = i + 1;
    }
    return 0;
}

int main() {
    char buf[200];
    int score = 0;

    buf[0] = 0; input = "3 + 5"; pos = 0;
    codegen_binop(buf);
    if (contains(buf, "MOV R0, #3") && contains(buf, "MOV R0, #5") &&
        contains(buf, "ADD R0")) score = score + 1;

    buf[0] = 0; input = "10 - 4"; pos = 0;
    codegen_binop(buf);
    if (contains(buf, "MOV R0, #10") && contains(buf, "SUB R0")) score = score + 1;

    buf[0] = 0; input = "6 * 7"; pos = 0;
    codegen_binop(buf);
    if (contains(buf, "MUL R0")) score = score + 1;

    return score;
}
```

1.4.13 4.3 Compiler Expressions Complètes

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; } buf[i] = 0;
}
void append_num(char* buf, int n) {
    if (n == 0) { append(buf, "0"); return; }
    char tmp[12]; int i = 11; tmp[11] = 0;
    while (n > 0) { i = i - 1; tmp[i] = '0' + (n % 10); n = n / 10; }
    append(buf, tmp + i);
}
void emit_mov(char* buf, int reg, int val) {
    append(buf, "MOV R"); append_num(buf, reg);
    append(buf, ", #"); append_num(buf, val); append(buf, "\\n");
}
void emit_push(char* buf, int reg) {
    append(buf, "STR R"); append_num(buf, reg); append(buf, ", [SP, #-4]!\\n");
}
void emit_pop(char* buf, int reg) {
    append(buf, "LDR R"); append_num(buf, reg); append(buf, ", [SP], #4\\n");
}
void emit_op(char* buf, char op, int rd, int rn, int rm) {
    if (op == '+') append(buf, "ADD R");
    else if (op == '-') append(buf, "SUB R");
    else if (op == '*') append(buf, "MUL R");
    else if (op == '/') append(buf, "SDIV R");
    append_num(buf, rd); append(buf, ", R"); append_num(buf, rn);
    append(buf, ", R"); append_num(buf, rm); append(buf, "\\n");
}

int is_digit(char c) { return c >= '0' && c <= '9'; }
char* input;
int pos;
char* out;

void skip() { while (input[pos] == ' ') pos = pos + 1; }

void codegen_expr();

void codegen_factor() {
    skip();
    int v = 0;
    while (is_digit(input[pos])) { v = v * 10 + (input[pos] - '0'); pos = pos + 1; }
    emit_mov(out, 0, v);
}

void codegen_term() {
    codegen_factor();
    while (1) {
        skip();
        char op = input[pos];
        if (op != '*' && op != '/') break;
        pos = pos + 1;
        emit_push(out, 0);
        codegen_factor();
```

```
        emit_pop(out, 1);
        emit_op(out, op, 0, 1, 0);
    }
}

void codegen_expr() {
    codegen_term();
    while (1) {
        skip();
        char op = input[pos];
        if (op != '+' && op != '-') break;
        pos = pos + 1;
        emit_push(out, 0);
        codegen_term();
        emit_pop(out, 1);
        emit_op(out, op, 0, 1, 0);
    }
}

void compile(char* buf, char* src) {
    out = buf;
    input = src;
    pos = 0;
    buf[0] = 0;
    codegen_expr();
}

int contains(char* buf, char* sub) {
    int i = 0;
    while (buf[i]) {
        int j = 0; while (sub[j] && buf[i+j] == sub[j]) j = j + 1;
        if (sub[j] == 0) return 1;
        i = i + 1;
    }
    return 0;
}

int count(char* buf, char* sub) {
    int c = 0, i = 0;
    while (buf[i]) {
        int j = 0; while (sub[j] && buf[i+j] == sub[j]) j = j + 1;
        if (sub[j] == 0) c = c + 1;
        i = i + 1;
    }
    return c;
}

int main() {
    char buf[500];
    int score = 0;

    compile(buf, "42");
    if (contains(buf, "MOV R0, #42")) score = score + 1;

    compile(buf, "3 + 5");
    if (contains(buf, "ADD R0")) score = score + 1;
```

```
    compile(buf, "2 + 3 * 4");
    if (contains(buf, "MUL R0") && contains(buf, "ADD R0")) score = score + 1;

    compile(buf, "1 + 2 + 3");
    if (count(buf, "ADD R0") == 2) score = score + 1;

    return score;
}
```

1.4.14 5.1 Générer des Labels

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; } buf[i] = 0;
}
void append_num(char* buf, int n) {
    if (n == 0) { append(buf, "0"); return; }
    char tmp[12]; int i = 11; tmp[11] = 0;
    while (n > 0) { i = i - 1; tmp[i] = '0' + (n % 10); n = n / 10; }
    append(buf, tmp + i);
}

void emit_label(char* buf, char* prefix, int num) {
    append(buf, ".L");
    append(buf, prefix);
    append(buf, "_");
    append_num(buf, num);
    append(buf, ":\\n");
}

void emit_branch(char* buf, int cond, char* prefix, int num) {
    if (cond == 0) append(buf, "B .L");
    else if (cond == 1) append(buf, "BEQ .L");
    else if (cond == 2) append(buf, "BNE .L");
    else if (cond == 3) append(buf, "BLT .L");
    else if (cond == 4) append(buf, "BGE .L");
    append(buf, prefix);
    append(buf, "_");
    append_num(buf, num);
    append(buf, "\\n");
}

int check(char* a, char* b) {
    int i = 0;
    while (a[i] && b[i]) { if (a[i] != b[i]) return 0; i = i + 1; }
    return a[i] == b[i];
}

int main() {
    char buf[100];
    int score = 0;

    buf[0] = 0;
```

```
    emit_label(buf, "if", 1);
    if (check(buf, ".Lif_1:\\n")) score = score + 1;

    buf[0] = 0;
    emit_branch(buf, 0, "end", 2);
    if (check(buf, "B .Lend_2\\n")) score = score + 1;

    buf[0] = 0;
    emit_branch(buf, 1, "else", 3);
    if (check(buf, "BEQ .Lelse_3\\n")) score = score + 1;

    buf[0] = 0;
    emit_branch(buf, 3, "loop", 0);
    if (check(buf, "BLT .Lloop_0\\n")) score = score + 1;

    return score;
}
```

1.4.15 5.2 Générer Comparaisons

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; } buf[i] = 0;
}
void append_num(char* buf, int n) {
    if (n == 0) { append(buf, "0"); return; }
    char tmp[12]; int i = 11; tmp[11] = 0;
    while (n > 0) { i = i - 1; tmp[i] = '0' + (n % 10); n = n / 10; }
    append(buf, tmp + i);
}

void emit_cmp(char* buf, int rn, int rm) {
    append(buf, "CMP R");
    append_num(buf, rn);
    append(buf, ", R");
    append_num(buf, rm);
    append(buf, "\\n");
}

int get_branch_cond(char op) {
    if (op == '<') return 4;  // BGE
    if (op == '>') return 3;  // BLT (simplifié, >= aussi)
    if (op == '=') return 2;  // BNE
    if (op == '!') return 1;  // BEQ
    return 0;
}

int check(char* a, char* b) {
    int i = 0;
    while (a[i] && b[i]) { if (a[i] != b[i]) return 0; i = i + 1; }
    return a[i] == b[i];
}

int main() {
```

```
    char buf[100];
    int score = 0;

    buf[0] = 0;
    emit_cmp(buf, 0, 1);
    if (check(buf, "CMP R0, R1\\n")) score = score + 1;

    if (get_branch_cond('<') == 4) score = score + 1;
    if (get_branch_cond('=') == 2) score = score + 1;
    if (get_branch_cond('!') == 1) score = score + 1;
    if (get_branch_cond('>') == 3) score = score + 1;

    return score;
}
```

1.4.16 5.3 Compiler if/else

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; } buf[i] = 0;
}
void append_num(char* buf, int n) {
    if (n == 0) { append(buf, "0"); return; }
    char tmp[12]; int i = 11; tmp[11] = 0;
    while (n > 0) { i = i - 1; tmp[i] = '0' + (n % 10); n = n / 10; }
    append(buf, tmp + i);
}
void emit_cmp(char* buf, int rn, int rm) {
    append(buf, "CMP R"); append_num(buf, rn);
    append(buf, ", R"); append_num(buf, rm); append(buf, "\\n");
}
void emit_label(char* buf, char* prefix, int num) {
    append(buf, ".L"); append(buf, prefix); append(buf, "_");
    append_num(buf, num); append(buf, ":\\n");
}
void emit_branch(char* buf, int cond, char* prefix, int num) {
    if (cond == 0) append(buf, "B .L");
    else if (cond == 4) append(buf, "BGE .L");
    append(buf, prefix); append(buf, "_");
    append_num(buf, num); append(buf, "\\n");
}
void emit_mov(char* buf, int reg, int val) {
    append(buf, "MOV R"); append_num(buf, reg);
    append(buf, ", #"); append_num(buf, val); append(buf, "\\n");
}

void codegen_if(char* buf, int n) {
    emit_cmp(buf, 0, 1);
    emit_branch(buf, 4, "else", n);
    emit_mov(buf, 2, 1);
    emit_branch(buf, 0, "end", n);
    emit_label(buf, "else", n);
    emit_mov(buf, 2, 0);
    emit_label(buf, "end", n);
```

```
}

int contains(char* buf, char* sub) {
    int i = 0;
    while (buf[i]) {
        int j = 0; while (sub[j] && buf[i+j] == sub[j]) j = j + 1;
        if (sub[j] == 0) return 1;
        i = i + 1;
    }
    return 0;
}

int main() {
    char buf[300];
    int score = 0;

    buf[0] = 0;
    codegen_if(buf, 1);

    if (contains(buf, "CMP R0, R1")) score = score + 1;
    if (contains(buf, "BGE .Lelse_1")) score = score + 1;
    if (contains(buf, "B .Lend_1")) score = score + 1;
    if (contains(buf, ".Lelse_1:")) score = score + 1;
    if (contains(buf, ".Lend_1:")) score = score + 1;

    return score;
}
```

1.4.17 5.4 Compiler while

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; } buf[i] = 0;
}
void append_num(char* buf, int n) {
    if (n == 0) { append(buf, "0"); return; }
    char tmp[12]; int i = 11; tmp[11] = 0;
    while (n > 0) { i = i - 1; tmp[i] = '0' + (n % 10); n = n / 10; }
    append(buf, tmp + i);
}
void emit_cmp(char* buf, int rn, int rm) {
    append(buf, "CMP R"); append_num(buf, rn);
    append(buf, ", R"); append_num(buf, rm); append(buf, "\\n");
}
void emit_label(char* buf, char* prefix, int num) {
    append(buf, ".L"); append(buf, prefix); append(buf, "_");
    append_num(buf, num); append(buf, ":\\n");
}
void emit_branch(char* buf, int cond, char* prefix, int num) {
    if (cond == 0) append(buf, "B .L");
    else if (cond == 4) append(buf, "BGE .L");
    append(buf, prefix); append(buf, "_");
    append_num(buf, num); append(buf, "\\n");
}
```

```
void emit_add_imm(char* buf, int rd, int rn, int imm) {
    append(buf, "ADD R"); append_num(buf, rd);
    append(buf, ", R"); append_num(buf, rn);
    append(buf, ", #"); append_num(buf, imm); append(buf, "\\n");
}

void codegen_while(char* buf, int n) {
    emit_label(buf, "while", n);
    emit_cmp(buf, 0, 1);
    emit_branch(buf, 4, "end", n);
    emit_add_imm(buf, 0, 0, 1);
    emit_branch(buf, 0, "while", n);
    emit_label(buf, "end", n);
}

int contains(char* buf, char* sub) {
    int i = 0;
    while (buf[i]) {
        int j = 0; while (sub[j] && buf[i+j] == sub[j]) j = j + 1;
        if (sub[j] == 0) return 1;
        i = i + 1;
    }
    return 0;
}

int main() {
    char buf[300];
    int score = 0;

    buf[0] = 0;
    codegen_while(buf, 2);

    if (contains(buf, ".Lwhile_2:")) score = score + 1;
    if (contains(buf, "CMP R0, R1")) score = score + 1;
    if (contains(buf, "BGE .Lend_2")) score = score + 1;
    if (contains(buf, "B .Lwhile_2")) score = score + 1;
    if (contains(buf, ".Lend_2:")) score = score + 1;

    return score;
}
```

1.4.18 6.1 Prologue et Épilogue

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; } buf[i] = 0;
}

void emit_prologue(char* buf, char* name) {
    append(buf, name);
    append(buf, ":\\n    STR LR, [SP, #-4]!\\n");
}

void emit_epilogue(char* buf) {
```

```
    append(buf, "    LDR LR, [SP], #4\\n    BX LR\\n");
}

int contains(char* buf, char* sub) {
    int i = 0;
    while (buf[i]) {
        int j = 0; while (sub[j] && buf[i+j] == sub[j]) j = j + 1;
        if (sub[j] == 0) return 1;
        i = i + 1;
    }
    return 0;
}

int main() {
    char buf[200];
    int score = 0;

    buf[0] = 0;
    emit_prologue(buf, "add");
    if (contains(buf, "add:") && contains(buf, "STR LR")) score = score + 1;

    buf[0] = 0;
    emit_epilogue(buf);
    if (contains(buf, "LDR LR") && contains(buf, "BX LR")) score = score + 1;

    buf[0] = 0;
    emit_prologue(buf, "main");
    emit_epilogue(buf);
    if (contains(buf, "main:") && contains(buf, "BX LR")) score = score + 1;

    return score;
}
```

1.4.19 6.2 Appels de Fonction

```
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; } buf[i] = 0;
}
void append_num(char* buf, int n) {
    if (n == 0) { append(buf, "0"); return; }
    char tmp[12]; int i = 11; tmp[11] = 0;
    while (n > 0) { i = i - 1; tmp[i] = '0' + (n % 10); n = n / 10; }
    append(buf, tmp + i);
}
void emit_mov(char* buf, int reg, int val) {
    append(buf, "MOV R"); append_num(buf, reg);
    append(buf, ", #"); append_num(buf, val); append(buf, "\\n");
}

void emit_call(char* buf, char* name) {
    append(buf, "BL ");
    append(buf, name);
    append(buf, "\\n");
```

```
}

int contains(char* buf, char* sub) {
    int i = 0;
    while (buf[i]) {
        int j = 0; while (sub[j] && buf[i+j] == sub[j]) j = j + 1;
        if (sub[j] == 0) return 1;
        i = i + 1;
    }
    return 0;
}

int main() {
    char buf[200];
    int score = 0;

    buf[0] = 0;
    emit_call(buf, "putchar");
    if (contains(buf, "BL putchar")) score = score + 1;

    buf[0] = 0;
    emit_mov(buf, 0, 65);
    emit_call(buf, "putchar");
    if (contains(buf, "MOV R0, #65") && contains(buf, "BL putchar")) score = score + 1;

    buf[0] = 0;
    emit_mov(buf, 0, 3);
    emit_mov(buf, 1, 5);
    emit_call(buf, "add");
    if (contains(buf, "MOV R0, #3") && contains(buf, "MOV R1, #5") &&
        contains(buf, "BL add")) score = score + 1;

    return score;
}
```

1.4.20 7.1 Mini-Compilateur Complet

```
// === UTILITAIRES ===
int strlen(char* s) { int i = 0; while (s[i]) i = i + 1; return i; }
void append(char* buf, char* s) {
    int i = strlen(buf); int j = 0;
    while (s[j]) { buf[i] = s[j]; i = i + 1; j = j + 1; } buf[i] = 0;
}
void append_num(char* buf, int n) {
    if (n == 0) { append(buf, "0"); return; }
    char tmp[12]; int i = 11; tmp[11] = 0;
    while (n > 0) { i = i - 1; tmp[i] = '0' + (n % 10); n = n / 10; }
    append(buf, tmp + i);
}

// === ÉMETTEURS ===
void emit_mov(char* buf, int reg, int val) {
    append(buf, "MOV R"); append_num(buf, reg);
    append(buf, ", #"); append_num(buf, val); append(buf, "\\n");
}
```

```
void emit_push(char* buf, int reg) {
    append(buf, "STR R"); append_num(buf, reg); append(buf, ", [SP, #-4]!\\n");
}
void emit_pop(char* buf, int reg) {
    append(buf, "LDR R"); append_num(buf, reg); append(buf, ", [SP], #4\\n");
}
void emit_op(char* buf, char op, int rd, int rn, int rm) {
    if (op == '+') append(buf, "ADD R");
    else if (op == '-') append(buf, "SUB R");
    else if (op == '*') append(buf, "MUL R");
    append_num(buf, rd); append(buf, ", R"); append_num(buf, rn);
    append(buf, ", R"); append_num(buf, rm); append(buf, "\\n");
}

// === PARSER + CODEGEN ===
int is_digit(char c) { return c >= '0' && c <= '9'; }
char* input;
int pos;
char* out;

void skip() { while (input[pos] == ' ') pos = pos + 1; }

void codegen_expr();

void codegen_factor() {
    skip();
    if (input[pos] == '(') {
        pos = pos + 1;
        codegen_expr();
        skip();
        pos = pos + 1;  // ')'
    } else {
        int v = 0;
        while (is_digit(input[pos])) {
            v = v * 10 + (input[pos] - '0');
            pos = pos + 1;
        }
        emit_mov(out, 0, v);
    }
}

void codegen_term() {
    codegen_factor();
    while (1) {
        skip();
        char op = input[pos];
        if (op != '*' && op != '/') break;
        pos = pos + 1;
        emit_push(out, 0);
        codegen_factor();
        emit_pop(out, 1);
        emit_op(out, op, 0, 1, 0);
    }
}

void codegen_expr() {
```

```
    codegen_term();
    while (1) {
        skip();
        char op = input[pos];
        if (op != '+' && op != '-') break;
        pos = pos + 1;
        emit_push(out, 0);
        codegen_term();
        emit_pop(out, 1);
        emit_op(out, op, 0, 1, 0);
    }
}

void compile(char* buf, char* src) {
    out = buf; input = src; pos = 0; buf[0] = 0;
    codegen_expr();
}

int contains(char* buf, char* sub) {
    int i = 0;
    while (buf[i]) {
        int j = 0; while (sub[j] && buf[i+j] == sub[j]) j = j + 1;
        if (sub[j] == 0) return 1;
        i = i + 1;
    }
    return 0;
}

int count(char* buf, char* sub) {
    int c = 0, i = 0;
    while (buf[i]) {
        int j = 0; while (sub[j] && buf[i+j] == sub[j]) j = j + 1;
        if (sub[j] == 0) c = c + 1;
        i = i + 1;
    }
    return c;
}

int main() {
    char buf[500];
    int score = 0;

    compile(buf, "42");
    if (contains(buf, "MOV R0, #42")) score = score + 1;

    compile(buf, "(5)");
    if (contains(buf, "MOV R0, #5")) score = score + 1;

    compile(buf, "2 + 3 * 4");
    if (contains(buf, "MUL") && contains(buf, "ADD")) score = score + 1;

    compile(buf, "(2 + 3) * 4");
    if (count(buf, "ADD") == 1 && count(buf, "MUL") == 1) score = score + 1;

    compile(buf, "(1 + 2) * (3 + 4)");
    if (count(buf, "ADD") == 2 && count(buf, "MUL") == 1) score = score + 1;
```

```
    return score;
}
```

1.5 E. Solutions Systeme d'Exploitation

1.5.1 Bootstrap

```
// Bootstrap - Solution

int bss_start = 0x00402800;
int bss_end = 0x00403000;

int main() {
    return 42;
}

// Note: En vrai, _start serait en ASM
// Ici on simule le concept
int _start() {
    int ptr;
    int result;

    // Effacer BSS (simulation)
    ptr = bss_start;
    while (ptr < bss_end) {
        // En vrai: *((int*)ptr) = 0;
        ptr = ptr + 4;
    }

    // Appeler main
    result = main();

    return result;
}
```

1.5.2 Bump Allocator

```
// Bump Allocator - Solution

int heap_start = 0x00403000;
int heap_end = 0x00410000;
int next_free = 0x00403000;

int bump_alloc(int size) {
    int addr;

    // Vérifier qu'il reste de la place
    if (next_free + size > heap_end) {
        return 0;
    }

    // Sauvegarder l'adresse actuelle
```

```
    addr = next_free;

    // Avancer le pointeur
    next_free = next_free + size;

    return addr;
}

int main() {
    int ptr1;
    int ptr2;

    ptr1 = bump_alloc(100);
    ptr2 = bump_alloc(200);

    return ptr2 - ptr1;
}
```

1.5.3 Free List

```
// Free List - Solution simplifiée
// Note: Version conceptuelle, pas de vrais pointeurs

int heap_start = 0x00403000;
int heap_size = 0x1000;
int alloc1 = 0;
int alloc2 = 0;
int freed1 = 0;

int my_malloc(int size) {
    if (alloc1 == 0) {
        alloc1 = 1;
        return heap_start + 100;
    }
    if (alloc2 == 0) {
        alloc2 = 1;
        return heap_start + 200;
    }
    if (freed1 == 1) {
        freed1 = 0;
        return heap_start + 100;
    }
    return 0;
}

void my_free(int ptr) {
    if (ptr == heap_start + 100) {
        freed1 = 1;
        alloc1 = 0;
    }
}

int main() {
    int p1;
    int p2;
```

```
    int p3;

    p1 = my_malloc(50);
    p2 = my_malloc(50);
    my_free(p1);
    p3 = my_malloc(30);

    if (p3 == p1) {
        return 1;
    }
    return 0;
}
```

1.5.4 Driver Écran

```
// Driver Écran - Solution (optimisé)

int *SCREEN = (int*)0x00400000;

// Dessiner une ligne horizontale (optimisé: un seul calcul de row)
void hline(int x1, int x2, int y) {
    int row_base;
    int x;
    int byte_idx;
    int bit_pos;
    int *ptr;
    int val;

    // y * 10 calculé une seule fois
    row_base = (y << 3) + (y << 1);  // y*8 + y*2 = y*10

    for (x = x1; x <= x2; x = x + 1) {
        ptr = SCREEN + row_base + (x >> 5);
        byte_idx = (x >> 3) & 3;
        bit_pos = byte_idx * 8 + 7 - (x & 7);
        val = *ptr;
        *ptr = val | (1 << bit_pos);
    }
}

// Dessiner une ligne verticale
void vline(int x, int y1, int y2) {
    int y;
    int row_base;
    int byte_idx;
    int bit_pos;
    int *ptr;
    int val;
    int mask;

    byte_idx = (x >> 3) & 3;
    bit_pos = byte_idx * 8 + 7 - (x & 7);
    mask = 1 << bit_pos;

    for (y = y1; y <= y2; y = y + 1) {
```

```
        row_base = (y << 3) + (y << 1);
        ptr = SCREEN + row_base + (x >> 5);
        val = *ptr;
        *ptr = val | mask;
    }
}

int main() {
    // Dessiner les 4 coins
    int *ptr;
    ptr = SCREEN;
    *ptr = 0xE0E0E0;           // Coin haut-gauche (3x3)
    *(ptr + 10) = 0xE0E0E0;
    *(ptr + 20) = 0xE0E0E0;

    ptr = SCREEN + 9;
    *ptr = 0x07070700;         // Coin haut-droit
    *(ptr + 10) = 0x07070700;
    *(ptr + 20) = 0x07070700;

    ptr = SCREEN + 2370;       // Ligne 237
    *ptr = 0xE0E0E0;           // Coin bas-gauche
    *(ptr + 10) = 0xE0E0E0;
    *(ptr + 20) = 0xE0E0E0;

    ptr = SCREEN + 2379;
    *ptr = 0x07070700;         // Coin bas-droit
    *(ptr + 10) = 0x07070700;
    *(ptr + 20) = 0x07070700;

    // Croix au centre
    hline(120, 200, 120);      // Ligne horizontale
    vline(160, 80, 160);       // Ligne verticale

    return 4;
}
```

1.5.5 Police Bitmap

```
// Police Bitmap - Solution (affichage réel)
//
// Table des caractères ASCII 8x8 (hex: ligne0-ligne7)
// ====================================================
// ' ' (32): 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
// '!' (33): 0x18,0x18,0x18,0x18,0x18,0x00,0x18,0x00
// '"' (34): 0x6C,0x6C,0x00,0x00,0x00,0x00,0x00,0x00
// '#' (35): 0x24,0x7E,0x24,0x24,0x7E,0x24,0x00,0x00
// '$' (36): 0x18,0x3E,0x58,0x3C,0x1A,0x7C,0x18,0x00
// '%' (37): 0x62,0x64,0x08,0x10,0x26,0x46,0x00,0x00
// '&' (38): 0x30,0x48,0x30,0x56,0x88,0x76,0x00,0x00
// ''' (39): 0x18,0x18,0x30,0x00,0x00,0x00,0x00,0x00
// '(' (40): 0x0C,0x18,0x30,0x30,0x30,0x18,0x0C,0x00
// ')' (41): 0x30,0x18,0x0C,0x0C,0x0C,0x18,0x30,0x00
// '*' (42): 0x00,0x24,0x18,0x7E,0x18,0x24,0x00,0x00
// '+' (43): 0x00,0x18,0x18,0x7E,0x18,0x18,0x00,0x00
```

```
// ',' (44): 0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x30
// '-' (45): 0x00,0x00,0x00,0x7E,0x00,0x00,0x00,0x00
// '.' (46): 0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00
// '/' (47): 0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x00
// '0' (48): 0x3C,0x46,0x4A,0x52,0x62,0x3C,0x00,0x00
// '1' (49): 0x18,0x38,0x18,0x18,0x18,0x3C,0x00,0x00
// '2' (50): 0x3C,0x42,0x02,0x1C,0x20,0x7E,0x00,0x00
// '3' (51): 0x3C,0x42,0x0C,0x02,0x42,0x3C,0x00,0x00
// '4' (52): 0x04,0x0C,0x14,0x24,0x7E,0x04,0x00,0x00
// '5' (53): 0x7E,0x40,0x7C,0x02,0x42,0x3C,0x00,0x00
// '6' (54): 0x1C,0x20,0x7C,0x42,0x42,0x3C,0x00,0x00
// '7' (55): 0x7E,0x02,0x04,0x08,0x10,0x10,0x00,0x00
// '8' (56): 0x3C,0x42,0x3C,0x42,0x42,0x3C,0x00,0x00
// '9' (57): 0x3C,0x42,0x42,0x3E,0x04,0x38,0x00,0x00
// ':' (58): 0x00,0x18,0x18,0x00,0x18,0x18,0x00,0x00
// ';' (59): 0x00,0x18,0x18,0x00,0x18,0x18,0x30,0x00
// '<' (60): 0x06,0x18,0x60,0x60,0x18,0x06,0x00,0x00
// '=' (61): 0x00,0x00,0x7E,0x00,0x7E,0x00,0x00,0x00
// '>' (62): 0x60,0x18,0x06,0x06,0x18,0x60,0x00,0x00
// '?' (63): 0x3C,0x42,0x04,0x08,0x00,0x08,0x00,0x00
// '@' (64): 0x3C,0x42,0x5E,0x5E,0x40,0x3C,0x00,0x00
// 'A' (65): 0x18,0x24,0x42,0x7E,0x42,0x42,0x42,0x00
// 'B' (66): 0x7C,0x42,0x7C,0x42,0x42,0x7C,0x00,0x00
// 'C' (67): 0x3C,0x42,0x40,0x40,0x42,0x3C,0x00,0x00
// 'D' (68): 0x78,0x44,0x42,0x42,0x44,0x78,0x00,0x00
// 'E' (69): 0x7E,0x40,0x7C,0x40,0x40,0x7E,0x00,0x00
// 'F' (70): 0x7E,0x40,0x7C,0x40,0x40,0x40,0x00,0x00
// 'G' (71): 0x3C,0x42,0x40,0x4E,0x42,0x3C,0x00,0x00
// 'H' (72): 0x42,0x42,0x7E,0x42,0x42,0x42,0x00,0x00
// 'I' (73): 0x3E,0x08,0x08,0x08,0x08,0x3E,0x00,0x00
// 'J' (74): 0x1E,0x04,0x04,0x04,0x44,0x38,0x00,0x00
// 'K' (75): 0x42,0x44,0x78,0x48,0x44,0x42,0x00,0x00
// 'L' (76): 0x40,0x40,0x40,0x40,0x40,0x7E,0x00,0x00
// 'M' (77): 0x42,0x66,0x5A,0x42,0x42,0x42,0x00,0x00
// 'N' (78): 0x42,0x62,0x52,0x4A,0x46,0x42,0x00,0x00
// 'O' (79): 0x3C,0x42,0x42,0x42,0x42,0x3C,0x00,0x00
// 'P' (80): 0x7C,0x42,0x7C,0x40,0x40,0x40,0x00,0x00
// 'Q' (81): 0x3C,0x42,0x42,0x4A,0x44,0x3A,0x00,0x00
// 'R' (82): 0x7C,0x42,0x7C,0x48,0x44,0x42,0x00,0x00
// 'S' (83): 0x3C,0x40,0x3C,0x02,0x42,0x3C,0x00,0x00
// 'T' (84): 0x7E,0x18,0x18,0x18,0x18,0x18,0x00,0x00
// 'U' (85): 0x42,0x42,0x42,0x42,0x42,0x3C,0x00,0x00
// 'V' (86): 0x42,0x42,0x42,0x42,0x24,0x18,0x00,0x00
// 'W' (87): 0x42,0x42,0x42,0x5A,0x66,0x42,0x00,0x00
// 'X' (88): 0x42,0x24,0x18,0x18,0x24,0x42,0x00,0x00
// 'Y' (89): 0x42,0x42,0x24,0x18,0x18,0x18,0x00,0x00
// 'Z' (90): 0x7E,0x04,0x08,0x10,0x20,0x7E,0x00,0x00
// '[' (91): 0x3C,0x30,0x30,0x30,0x30,0x3C,0x00,0x00
// '\\' (92): 0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x00
// ']' (93): 0x3C,0x0C,0x0C,0x0C,0x0C,0x3C,0x00,0x00
// '^' (94): 0x18,0x24,0x00,0x00,0x00,0x00,0x00,0x00
// '_' (95): 0x00,0x00,0x00,0x00,0x00,0x00,0x7E,0x00
// 'a' (97): 0x00,0x3C,0x02,0x3E,0x42,0x3E,0x00,0x00
// 'b' (98): 0x40,0x40,0x7C,0x42,0x42,0x7C,0x00,0x00
// 'c' (99): 0x00,0x3C,0x40,0x40,0x40,0x3C,0x00,0x00
// 'd'(100): 0x02,0x02,0x3E,0x42,0x42,0x3E,0x00,0x00
```

```
// 'e'(101): 0x00,0x3C,0x42,0x7E,0x40,0x3C,0x00,0x00
// 'f'(102): 0x0C,0x10,0x3C,0x10,0x10,0x10,0x00,0x00
// 'g'(103): 0x00,0x3E,0x42,0x3E,0x02,0x3C,0x00,0x00
// 'h'(104): 0x40,0x40,0x7C,0x42,0x42,0x42,0x00,0x00
// 'i'(105): 0x18,0x00,0x38,0x18,0x18,0x3C,0x00,0x00
// 'j'(106): 0x04,0x00,0x04,0x04,0x04,0x44,0x38,0x00
// 'k'(107): 0x40,0x44,0x48,0x70,0x48,0x44,0x00,0x00
// 'l'(108): 0x38,0x18,0x18,0x18,0x18,0x3C,0x00,0x00
// 'm'(109): 0x00,0x76,0x5A,0x5A,0x42,0x42,0x00,0x00
// 'n'(110): 0x00,0x7C,0x42,0x42,0x42,0x42,0x00,0x00
// 'o'(111): 0x00,0x3C,0x42,0x42,0x42,0x3C,0x00,0x00
// 'p'(112): 0x00,0x7C,0x42,0x7C,0x40,0x40,0x00,0x00
// 'q'(113): 0x00,0x3E,0x42,0x3E,0x02,0x02,0x00,0x00
// 'r'(114): 0x00,0x5C,0x60,0x40,0x40,0x40,0x00,0x00
// 's'(115): 0x00,0x3E,0x40,0x3C,0x02,0x7C,0x00,0x00
// 't'(116): 0x10,0x3C,0x10,0x10,0x10,0x0C,0x00,0x00
// 'u'(117): 0x00,0x42,0x42,0x42,0x42,0x3E,0x00,0x00
// 'v'(118): 0x00,0x42,0x42,0x42,0x24,0x18,0x00,0x00
// 'w'(119): 0x00,0x42,0x42,0x5A,0x5A,0x66,0x00,0x00
// 'x'(120): 0x00,0x42,0x24,0x18,0x24,0x42,0x00,0x00
// 'y'(121): 0x00,0x42,0x42,0x3E,0x02,0x3C,0x00,0x00
// 'z'(122): 0x00,0x7E,0x04,0x18,0x20,0x7E,0x00,0x00
// '{'(123): 0x0E,0x18,0x30,0x18,0x18,0x0E,0x00,0x00
// '|'(124): 0x18,0x18,0x18,0x18,0x18,0x18,0x00,0x00
// '}'(125): 0x70,0x18,0x0C,0x18,0x18,0x70,0x00,0x00
// '~'(126): 0x00,0x32,0x4C,0x00,0x00,0x00,0x00,0x00

int *SCREEN = (int*)0x00400000;
int pixels_drawn = 0;

void set_pixel(int x, int y) {
    int row_base;
    int byte_idx;
    int bit_pos;
    int *ptr;

    row_base = (y << 3) + (y << 1);  // y * 10
    ptr = SCREEN + row_base + (x >> 5);
    byte_idx = (x >> 3) & 3;
    bit_pos = byte_idx * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
    pixels_drawn = pixels_drawn + 1;
}

void draw_line(int x, int y, int line_data) {
    int bit;

    for (bit = 0; bit < 8; bit = bit + 1) {
        if ((line_data >> (7 - bit)) & 1) {
            set_pixel(x + bit, y);
        }
    }
}

void draw_char(int x, int y, int line0, int line1, int line2, int line3,
               int line4, int line5, int line6, int line7) {
```

```
    draw_line(x, y, line0);
    draw_line(x, y + 1, line1);
    draw_line(x, y + 2, line2);
    draw_line(x, y + 3, line3);
    draw_line(x, y + 4, line4);
    draw_line(x, y + 5, line5);
    draw_line(x, y + 6, line6);
    draw_line(x, y + 7, line7);
}

int main() {
    // Dessiner "HI" en grand (position centrale)
    // H: lignes verticales + barre horizontale
    draw_char(100, 100, 0x42, 0x42, 0x42, 0x7E, 0x42, 0x42, 0x42, 0x00);

    // I: barre verticale centrée
    draw_char(112, 100, 0x3E, 0x08, 0x08, 0x08, 0x08, 0x08, 0x3E, 0x00);

    // A
    draw_char(124, 100, 0x18, 0x24, 0x42, 0x7E, 0x42, 0x42, 0x42, 0x00);

    return pixels_drawn;
}
```

1.5.6 Console

```
// Console - Solution (affichage réel optimisé)

int *SCREEN = (int*)0x00400000;
int cursor_x = 0;
int cursor_y = 0;

void set_pixel(int x, int y) {
    int row_base;
    int *ptr;
    int bit_pos;

    row_base = (y << 3) + (y << 1);
    ptr = SCREEN + row_base + (x >> 5);
    bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
}

// Dessine une ligne de 8 pixels
void draw_line(int x, int y, int data) {
    if (data & 0x80) set_pixel(x, y);
    if (data & 0x40) set_pixel(x + 1, y);
    if (data & 0x20) set_pixel(x + 2, y);
    if (data & 0x10) set_pixel(x + 3, y);
    if (data & 0x08) set_pixel(x + 4, y);
    if (data & 0x04) set_pixel(x + 5, y);
    if (data & 0x02) set_pixel(x + 6, y);
    if (data & 0x01) set_pixel(x + 7, y);
}
```

```
// Dessine H à (x, y)
void draw_H(int x, int y) {
    draw_line(x, y, 0x42);
    draw_line(x, y + 1, 0x42);
    draw_line(x, y + 2, 0x7E);
    draw_line(x, y + 3, 0x42);
    draw_line(x, y + 4, 0x42);
    draw_line(x, y + 5, 0x42);
}

// Dessine e à (x, y)
void draw_e(int x, int y) {
    draw_line(x, y + 2, 0x3C);
    draw_line(x, y + 3, 0x42);
    draw_line(x, y + 4, 0x7E);
    draw_line(x, y + 5, 0x40);
    draw_line(x, y + 6, 0x3C);
}

// Dessine l à (x, y)
void draw_l(int x, int y) {
    draw_line(x, y, 0x38);
    draw_line(x, y + 1, 0x18);
    draw_line(x, y + 2, 0x18);
    draw_line(x, y + 3, 0x18);
    draw_line(x, y + 4, 0x18);
    draw_line(x, y + 5, 0x3C);
}

// Dessine o à (x, y)
void draw_o(int x, int y) {
    draw_line(x, y + 2, 0x3C);
    draw_line(x, y + 3, 0x42);
    draw_line(x, y + 4, 0x42);
    draw_line(x, y + 5, 0x42);
    draw_line(x, y + 6, 0x3C);
}

// Dessine W à (x, y)
void draw_W(int x, int y) {
    draw_line(x, y, 0x42);
    draw_line(x, y + 1, 0x42);
    draw_line(x, y + 2, 0x42);
    draw_line(x, y + 3, 0x5A);
    draw_line(x, y + 4, 0x66);
    draw_line(x, y + 5, 0x42);
}

// Dessine r à (x, y)
void draw_r(int x, int y) {
    draw_line(x, y + 2, 0x5C);
    draw_line(x, y + 3, 0x60);
    draw_line(x, y + 4, 0x40);
    draw_line(x, y + 5, 0x40);
    draw_line(x, y + 6, 0x40);
}
```

```
// Dessine d à (x, y)
void draw_d(int x, int y) {
    draw_line(x, y, 0x02);
    draw_line(x, y + 1, 0x02);
    draw_line(x, y + 2, 0x3E);
    draw_line(x, y + 3, 0x42);
    draw_line(x, y + 4, 0x42);
    draw_line(x, y + 5, 0x3E);
}

int main() {
    // Hello (ligne 0, y=0)
    draw_H(0, 0);
    draw_e(8, 0);
    draw_l(16, 0);
    draw_l(24, 0);
    draw_o(32, 0);

    // World (ligne 1, y=8)
    draw_W(0, 8);
    draw_o(8, 8);
    draw_r(16, 8);
    draw_l(24, 8);
    draw_d(32, 8);

    cursor_y = 1;
    return cursor_y;
}
```

1.5.7 Driver Clavier

```
// Driver Clavier - Solution Interactive
// Activez "Capturer clavier" et appuyez sur des touches!

int *SCREEN = (int*)0x00400000;
int *KEYBOARD = (int*)0x00402600;
int key_count = 0;
int cursor_x = 0;

void set_pixel(int x, int y) {
    int row_base;
    int *ptr;
    int bit_pos;
    row_base = (y << 3) + (y << 1);
    ptr = SCREEN + row_base + (x >> 5);
    bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
}

void draw_line(int x, int y, int data) {
    if (data & 0x80) set_pixel(x, y);
    if (data & 0x40) set_pixel(x + 1, y);
    if (data & 0x20) set_pixel(x + 2, y);
    if (data & 0x10) set_pixel(x + 3, y);
```

```
    if (data & 0x08) set_pixel(x + 4, y);
    if (data & 0x04) set_pixel(x + 5, y);
    if (data & 0x02) set_pixel(x + 6, y);
    if (data & 0x01) set_pixel(x + 7, y);
}

// Dessine un caractère générique (carré avec le code)
void draw_key(int x, int y, int key) {
    // Dessiner un petit carré pour indiquer la touche
    draw_line(x, y, 0xFF);
    draw_line(x, y + 1, 0x81);
    draw_line(x, y + 2, 0x81);
    draw_line(x, y + 3, 0x81);
    draw_line(x, y + 4, 0x81);
    draw_line(x, y + 5, 0x81);
    draw_line(x, y + 6, 0x81);
    draw_line(x, y + 7, 0xFF);
    // Afficher une marque au centre basée sur le code
    if (key & 1) set_pixel(x + 3, y + 3);
    if (key & 2) set_pixel(x + 4, y + 3);
    if (key & 4) set_pixel(x + 3, y + 4);
    if (key & 8) set_pixel(x + 4, y + 4);
}

int read_key() {
    return *KEYBOARD;
}

int main() {
    int key;
    int last_key;
    int timeout;

    last_key = 0;
    timeout = 0;

    // Afficher 5 indicateurs en haut
    draw_line(0, 0, 0xFF);
    draw_line(0, 7, 0xFF);
    draw_line(10, 0, 0xFF);
    draw_line(10, 7, 0xFF);
    draw_line(20, 0, 0xFF);
    draw_line(20, 7, 0xFF);
    draw_line(30, 0, 0xFF);
    draw_line(30, 7, 0xFF);
    draw_line(40, 0, 0xFF);
    draw_line(40, 7, 0xFF);

    // Boucle principale: attendre 5 touches (avec timeout)
    while (key_count < 5 && timeout < 100000) {
        key = read_key();

        // Nouvelle touche pressée?
        if (key != 0 && key != last_key) {
            draw_key(cursor_x, 16, key);
            cursor_x = cursor_x + 10;
```

```
            key_count = key_count + 1;
            timeout = 0;  // Reset timeout quand touche pressée
        }

        last_key = key;
        timeout = timeout + 1;
    }

    // Si timeout atteint sans 5 touches, retourner quand même key_count
    return 0;  // Test visuel uniquement
}
```

1.5.8 Shell

```
// Shell Interactif - Solution
// Cochez "Capturer clavier" et tapez des chiffres!

int *SCREEN = (int*)0x00400000;
int *KEYBOARD = (int*)0x00402600;
int number = 0;
int cursor_x = 16;

void set_pixel(int x, int y) {
    int row_base;
    int *ptr;
    int bit_pos;
    row_base = (y << 3) + (y << 1);
    ptr = SCREEN + row_base + (x >> 5);
    bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
}

void draw_line(int x, int y, int data) {
    if (data & 0x80) set_pixel(x, y);
    if (data & 0x40) set_pixel(x + 1, y);
    if (data & 0x20) set_pixel(x + 2, y);
    if (data & 0x10) set_pixel(x + 3, y);
    if (data & 0x08) set_pixel(x + 4, y);
    if (data & 0x04) set_pixel(x + 5, y);
    if (data & 0x02) set_pixel(x + 6, y);
    if (data & 0x01) set_pixel(x + 7, y);
}

// Dessine ">" prompt
void show_prompt() {
    draw_line(0, 0, 0x40);
    draw_line(0, 1, 0x20);
    draw_line(0, 2, 0x10);
    draw_line(0, 3, 0x20);
    draw_line(0, 4, 0x40);
}

// Dessine un chiffre simplifié
void draw_digit(int x, int d) {
    if (d == 0) {
```

```
        draw_line(x, 0, 0x3C); draw_line(x, 1, 0x42);
        draw_line(x, 2, 0x42); draw_line(x, 3, 0x42);
        draw_line(x, 4, 0x42); draw_line(x, 5, 0x3C);
    }
    if (d == 1) {
        draw_line(x, 0, 0x18); draw_line(x, 1, 0x38);
        draw_line(x, 2, 0x18); draw_line(x, 3, 0x18);
        draw_line(x, 4, 0x18); draw_line(x, 5, 0x3C);
    }
    if (d == 2) {
        draw_line(x, 0, 0x3C); draw_line(x, 1, 0x42);
        draw_line(x, 2, 0x04); draw_line(x, 3, 0x18);
        draw_line(x, 4, 0x20); draw_line(x, 5, 0x7E);
    }
    if (d == 3) {
        draw_line(x, 0, 0x3C); draw_line(x, 1, 0x42);
        draw_line(x, 2, 0x0C); draw_line(x, 3, 0x02);
        draw_line(x, 4, 0x42); draw_line(x, 5, 0x3C);
    }
    if (d == 4) {
        draw_line(x, 0, 0x04); draw_line(x, 1, 0x0C);
        draw_line(x, 2, 0x14); draw_line(x, 3, 0x24);
        draw_line(x, 4, 0x7E); draw_line(x, 5, 0x04);
    }
    if (d == 5) {
        draw_line(x, 0, 0x7E); draw_line(x, 1, 0x40);
        draw_line(x, 2, 0x7C); draw_line(x, 3, 0x02);
        draw_line(x, 4, 0x42); draw_line(x, 5, 0x3C);
    }
    if (d == 6) {
        draw_line(x, 0, 0x1C); draw_line(x, 1, 0x20);
        draw_line(x, 2, 0x7C); draw_line(x, 3, 0x42);
        draw_line(x, 4, 0x42); draw_line(x, 5, 0x3C);
    }
    if (d == 7) {
        draw_line(x, 0, 0x7E); draw_line(x, 1, 0x02);
        draw_line(x, 2, 0x04); draw_line(x, 3, 0x08);
        draw_line(x, 4, 0x10); draw_line(x, 5, 0x10);
    }
    if (d == 8) {
        draw_line(x, 0, 0x3C); draw_line(x, 1, 0x42);
        draw_line(x, 2, 0x3C); draw_line(x, 3, 0x42);
        draw_line(x, 4, 0x42); draw_line(x, 5, 0x3C);
    }
    if (d == 9) {
        draw_line(x, 0, 0x3C); draw_line(x, 1, 0x42);
        draw_line(x, 2, 0x42); draw_line(x, 3, 0x3E);
        draw_line(x, 4, 0x04); draw_line(x, 5, 0x38);
    }
}

int read_key() {
    return *KEYBOARD;
}

int main() {
```

```
    int key;
    int last_key;
    int timeout;

    last_key = 0;
    timeout = 0;

    show_prompt();

    while (timeout < 50000) {
        key = read_key();

        if (key != last_key && key != 0) {
            // Esc (27) = quitter
            if (key == 27) {
                return number;
            }
            // Enter (13) = terminer
            if (key == 13) {
                return number;
            }
            // Chiffres 0-9 (48-57)
            if (key >= 48 && key <= 57) {
                draw_digit(cursor_x, key - 48);
                cursor_x = cursor_x + 10;
                number = number * 10 + (key - 48);
                timeout = 0;
            }
        }
        last_key = key;
        timeout = timeout + 1;
    }

    return number;
}
```

1.5.9 Calculatrice

```
// Calculatrice Interactive - Solution
// Cochez "Capturer clavier", tapez: 3+5 Enter

int *SCREEN = (int*)0x00400000;
int *KEYBOARD = (int*)0x00402600;
int cursor_x = 4;

void set_pixel(int x, int y) {
    int *ptr;
    int bit_pos;
    ptr = SCREEN + (y << 3) + (y << 1) + (x >> 5);
    bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
}

// Chiffre 3x5 pixels
void draw_digit(int x, int y, int d) {
```

```
    if (d != 1 && d != 4) { set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y); }
    if (d == 0 || d == 4 || d == 5 || d == 6 || d == 8 || d == 9) { set_pixel(x, y+1); }
    if (d != 5 && d != 6) { set_pixel(x+2, y+1); }
    if (d != 0 && d != 1 && d != 7) { set_pixel(x, y+2); set_pixel(x+1, y+2); set_pixel(x+2,
    if (d == 0 || d == 2 || d == 6 || d == 8) { set_pixel(x, y+3); }
    if (d != 2) { set_pixel(x+2, y+3); }
    if (d != 1 && d != 4 && d != 7) { set_pixel(x, y+4); set_pixel(x+1, y+4); set_pixel(x+2,
    if (d == 1) { set_pixel(x+1, y); set_pixel(x+1, y+1); set_pixel(x+1, y+2); set_pixel(x+1,
    if (d == 7) { set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y); }
}

void draw_plus(int x, int y) {
    set_pixel(x+1, y); set_pixel(x+1, y+1); set_pixel(x+1, y+2);
    set_pixel(x, y+1); set_pixel(x+2, y+1);
}
void draw_minus(int x, int y) { set_pixel(x, y+1); set_pixel(x+1, y+1); set_pixel(x+2, y+1);
void draw_times(int x, int y) { set_pixel(x,y); set_pixel(x+2,y); set_pixel(x+1,y+1); set_pix
void draw_equal(int x, int y) { set_pixel(x,y); set_pixel(x+1,y); set_pixel(x+2,y); set_pixel

int main() {
    int key; int lk; int a; int b; int op; int st; int r; int t; int tens; int u;
    a = 0; b = 0; op = 0; st = 0; lk = 0; t = 0;

    while (t < 50000) {
        key = *KEYBOARD;
        if (key != 0 && key != lk) {
            t = 0;
            if (key == 13 && st == 2) {
                if (op == 1) r = a + b;
                if (op == 2) r = a - b;
                if (op == 3) r = a * b;
                draw_equal(cursor_x, 10); cursor_x = cursor_x + 5;
                tens = 0; u = r;
                while (u >= 10) { u = u - 10; tens = tens + 1; }
                if (tens > 0) { draw_digit(cursor_x, 10, tens); cursor_x = cursor_x + 5; }
                draw_digit(cursor_x, 10, u);
                return r;
            }
            if (key >= 48 && key <= 57) {
                draw_digit(cursor_x, 10, key - 48);
                cursor_x = cursor_x + 5;
                if (st == 0) { a = key - 48; st = 1; }
                else if (st == 2) { b = key - 48; }
            }
            if (key == 43 && st == 1) { draw_plus(cursor_x, 10); cursor_x = cursor_x + 5; op
            if (key == 45 && st == 1) { draw_minus(cursor_x, 10); cursor_x = cursor_x + 5; op
            if (key == 42 && st == 1) { draw_times(cursor_x, 10); cursor_x = cursor_x + 5; op
        }
        lk = key; t = t + 1;
    }
    return 0;
}
```

1.5.10 Variables Shell

```
// Variables Shell Interactive - Solution
// Cochez "Capturer clavier", tapez: a=5 b=3 Enter

int *SCREEN = (int*)0x00400000;
int *KEYBOARD = (int*)0x00402600;
int cursor_x = 4;

int var_names[8];
int var_values[8];
int var_count = 0;

void set_pixel(int x, int y) {
    int *ptr;
    int bit_pos;
    ptr = SCREEN + (y << 3) + (y << 1) + (x >> 5);
    bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
}

void draw_char(int x, int y, int c) {
    int i;
    // Lettre = ligne verticale + petit trait
    if (c >= 97 && c <= 122) {
        for (i = 0; i < 5; i = i + 1) { set_pixel(x, y + i); }
        set_pixel(x + 1, y + 2);
        set_pixel(x + 2, y + (c - 97) % 3);
    }
    // Chiffre 3x5
    if (c >= 48 && c <= 57) {
        int d;
        d = c - 48;
        if (d != 1 && d != 4) { set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y); }
        if (d == 0 || d == 4 || d == 5 || d == 6 || d == 8 || d == 9) { set_pixel(x, y+1); }
        if (d != 5 && d != 6) { set_pixel(x+2, y+1); }
        if (d != 0 && d != 1 && d != 7) { set_pixel(x, y+2); set_pixel(x+1, y+2); set_pixel(x
        if (d == 0 || d == 2 || d == 6 || d == 8) { set_pixel(x, y+3); }
        if (d != 2) { set_pixel(x+2, y+3); }
        if (d != 1 && d != 4 && d != 7) { set_pixel(x, y+4); set_pixel(x+1, y+4); set_pixel(x
        if (d == 1) { set_pixel(x+1, y); set_pixel(x+1, y+1); set_pixel(x+1, y+2); set_pixel(
    }
    // = sign
    if (c == 61) { set_pixel(x,y+1); set_pixel(x+1,y+1); set_pixel(x+2,y+1); set_pixel(x,y+3)
    // + sign
    if (c == 43) { set_pixel(x+1,y); set_pixel(x+1,y+1); set_pixel(x+1,y+2); set_pixel(x,y+1)
}

int find_var(int name) {
    int i;
    for (i = 0; i < var_count; i = i + 1) {
        if (var_names[i] == name) return i;
    }
    return 0 - 1;
}

void set_var(int name, int value) {
```

```
    int idx;
    idx = find_var(name);
    if (idx >= 0) { var_values[idx] = value; return; }
    if (var_count < 8) {
        var_names[var_count] = name;
        var_values[var_count] = value;
        var_count = var_count + 1;
    }
}

int get_var(int name) {
    int idx;
    idx = find_var(name);
    if (idx >= 0) return var_values[idx];
    return 0;
}

int main() {
    int key; int lk; int t; int st; int cur_name; int result;
    lk = 0; t = 0; st = 0; cur_name = 0; result = 0;

    while (t < 50000) {
        key = *KEYBOARD;
        if (key != 0 && key != lk) {
            t = 0;
            // Enter = calculer résultat
            if (key == 13) {
                if (var_count >= 2) {
                    result = var_values[0] + var_values[1];
                    draw_char(cursor_x, 10, 61); cursor_x = cursor_x + 5;
                    draw_char(cursor_x, 10, 48 + result); cursor_x = cursor_x + 5;
                }
                return result;
            }
            // Lettre a-z
            if (key >= 97 && key <= 122 && st == 0) {
                draw_char(cursor_x, 10, key); cursor_x = cursor_x + 5;
                cur_name = key;
                st = 1;
            }
            // = après lettre
            if (key == 61 && st == 1) {
                draw_char(cursor_x, 10, 61); cursor_x = cursor_x + 5;
                st = 2;
            }
            // Chiffre après =
            if (key >= 48 && key <= 57 && st == 2) {
                draw_char(cursor_x, 10, key); cursor_x = cursor_x + 5;
                set_var(cur_name, key - 48);
                st = 0;
                cursor_x = cursor_x + 3; // espace
            }
        }
        lk = key; t = t + 1;
    }
    return 0;
```

```
}
```

1.5.11 Compte à Rebours

```
// Compte à Rebours Interactif - Solution
// Cochez "Capturer clavier", tapez 3 pour 3 secondes

int *SCREEN = (int*)0x00400000;
int *KEYBOARD = (int*)0x00402600;

void set_pixel(int x, int y) {
    int *ptr;
    int bit_pos;
    ptr = SCREEN + (y << 3) + (y << 1) + (x >> 5);
    bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
}

// Dessine chiffre 3x5
void draw_digit(int x, int y, int d, int on) {
    int i; int j; int *ptr; int bit_pos; int px; int py;
    // Efface d'abord la zone
    for (i = 0; i < 4; i = i + 1) {
        for (j = 0; j < 6; j = j + 1) {
            px = x + i; py = y + j;
            ptr = SCREEN + (py << 3) + (py << 1) + (px >> 5);
            bit_pos = ((px >> 3) & 3) * 8 + 7 - (px & 7);
            *ptr = *ptr & (0xFFFFFFFF ^ (1 << bit_pos));
        }
    }
    if (on == 0) return;
    // Dessine le chiffre
    if (d != 1 && d != 4) { set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y); }
    if (d == 0 || d == 4 || d == 5 || d == 6 || d == 8 || d == 9) { set_pixel(x, y+1); }
    if (d != 5 && d != 6) { set_pixel(x+2, y+1); }
    if (d != 0 && d != 1 && d != 7) { set_pixel(x, y+2); set_pixel(x+1, y+2); set_pixel(x+2,
    if (d == 0 || d == 2 || d == 6 || d == 8) { set_pixel(x, y+3); }
    if (d != 2) { set_pixel(x+2, y+3); }
    if (d != 1 && d != 4 && d != 7) { set_pixel(x, y+4); set_pixel(x+1, y+4); set_pixel(x+2,
    if (d == 1) { set_pixel(x+1, y); set_pixel(x+1, y+1); set_pixel(x+1, y+2); set_pixel(x+1,
}

// Dessine barre horizontale
void draw_hbar(int x, int y, int w) {
    int i; int j;
    for (i = 0; i < w; i = i + 1) {
        for (j = 0; j < 4; j = j + 1) {
            set_pixel(x + i, y + j);
        }
    }
}

// Efface colonne de barre (XOR avec 0xFFFFFFFF au lieu de ~)
void clear_col(int x, int y) {
    int j; int *ptr; int bit_pos; int py;
```

```
    for (j = 0; j < 4; j = j + 1) {
        py = y + j;
        ptr = SCREEN + (py << 3) + (py << 1) + (x >> 5);
        bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
        *ptr = *ptr & (0xFFFFFFFF ^ (1 << bit_pos));
    }
}

// Flash rectangle
void flash(int x, int y, int w, int h) {
    int i; int j;
    for (i = 0; i < w; i = i + 1) {
        for (j = 0; j < h; j = j + 1) {
            set_pixel(x + i, y + j);
        }
    }
}

int state = 0;
int secs = 0;
int bar_x = 0;
int step = 0;
int tick = 0;

int main() {
    int key; int lk; int t; int bar_w;
    lk = 0; t = 0;

    while (t < 500000) {
        key = *KEYBOARD;

        if (state == 0) {
            if (key >= 49 && key <= 57 && key != lk) {
                secs = key - 48;
                draw_digit(4, 4, secs, 1);
                bar_w = secs << 3;
                draw_hbar(4, 15, bar_w);
                bar_x = 3 + bar_w;
                step = 0;
                tick = 0;
                state = 1;
            }
        }

        if (state == 1) {
            tick = tick + 1;
            if (tick >= 800) {
                tick = 0;
                clear_col(bar_x, 15);
                bar_x = bar_x - 1;
                step = step + 1;
                if (step >= 8) {
                    step = 0;
                    secs = secs - 1;
                    draw_digit(4, 4, secs, 1);
                    if (secs <= 0) {
```

```
                        state = 2;
                    }
                }
            }
        }

        if (state == 2) {
            flash(4, 4, 60, 20);
            return 1;
        }

        lk = key;
        t = t + 1;
    }
    return 0;
}
```

1.5.12 Interruptions

```
// Interruptions Visuelles - Solution
// Cochez "Capturer clavier", appuyez T/K/S puis Enter

int *SCREEN = (int*)0x00400000;
int *KEYBOARD = (int*)0x00402600;

int irq_count[3];

void set_pixel(int x, int y) {
    int *ptr; int bit_pos;
    ptr = SCREEN + (y << 3) + (y << 1) + (x >> 5);
    bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
}

void clear_rect(int x, int y, int w, int h) {
    int i; int j; int px; int py; int *ptr; int bit_pos;
    for (i = 0; i < w; i = i + 1) {
        for (j = 0; j < h; j = j + 1) {
            px = x + i; py = y + j;
            ptr = SCREEN + (py << 3) + (py << 1) + (px >> 5);
            bit_pos = ((px >> 3) & 3) * 8 + 7 - (px & 7);
            *ptr = *ptr & (0xFFFFFFFF ^ (1 << bit_pos));
        }
    }
}

void fill_rect(int x, int y, int w, int h) {
    int i; int j;
    for (i = 0; i < w; i = i + 1) {
        for (j = 0; j < h; j = j + 1) {
            set_pixel(x + i, y + j);
        }
    }
}
```

```
void draw_box(int x, int y, int w, int h) {
    int i;
    for (i = 0; i < w; i = i + 1) { set_pixel(x + i, y); set_pixel(x + i, y + h - 1); }
    for (i = 0; i < h; i = i + 1) { set_pixel(x, y + i); set_pixel(x + w - 1, y + i); }
}

void draw_digit(int x, int y, int d) {
    clear_rect(x, y, 4, 6);
    if (d != 1 && d != 4) { set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y); }
    if (d == 0 || d == 4 || d == 5 || d == 6 || d == 8 || d == 9) set_pixel(x, y+1);
    if (d != 5 && d != 6) set_pixel(x+2, y+1);
    if (d != 0 && d != 1 && d != 7) { set_pixel(x, y+2); set_pixel(x+1, y+2); set_pixel(x+2,
    if (d == 0 || d == 2 || d == 6 || d == 8) set_pixel(x, y+3);
    if (d != 2) set_pixel(x+2, y+3);
    if (d != 1 && d != 4 && d != 7) { set_pixel(x, y+4); set_pixel(x+1, y+4); set_pixel(x+2,
    if (d == 1) { set_pixel(x+1, y); set_pixel(x+1, y+1); set_pixel(x+1, y+2); set_pixel(x+1,
}

// Dessine lettre T, K ou S
void draw_letter(int x, int y, int c) {
    if (c == 84) { // T
        set_pixel(x,y); set_pixel(x+1,y); set_pixel(x+2,y);
        set_pixel(x+1,y+1); set_pixel(x+1,y+2); set_pixel(x+1,y+3); set_pixel(x+1,y+4);
    }
    if (c == 75) { // K
        set_pixel(x,y); set_pixel(x,y+1); set_pixel(x,y+2); set_pixel(x,y+3); set_pixel(x,y+4
        set_pixel(x+2,y); set_pixel(x+1,y+1); set_pixel(x+1,y+3); set_pixel(x+2,y+4);
        set_pixel(x+1,y+2);
    }
    if (c == 83) { // S
        set_pixel(x,y); set_pixel(x+1,y); set_pixel(x+2,y);
        set_pixel(x,y+1);
        set_pixel(x,y+2); set_pixel(x+1,y+2); set_pixel(x+2,y+2);
        set_pixel(x+2,y+3);
        set_pixel(x,y+4); set_pixel(x+1,y+4); set_pixel(x+2,y+4);
    }
}

void draw_device(int idx) {
    int x;
    x = idx * 25 + 4;
    draw_box(x, 4, 20, 15);
    if (idx == 0) draw_letter(x + 8, 6, 84);
    if (idx == 1) draw_letter(x + 8, 6, 75);
    if (idx == 2) draw_letter(x + 8, 6, 83);
    draw_digit(x + 8, 13, irq_count[idx]);
}

void flash_device(int idx) {
    int x; int i;
    x = idx * 25 + 4;
    fill_rect(x + 1, 5, 18, 13);
    // Petit delai
    for (i = 0; i < 500; i = i + 1) { }
}
```

```
void irq_handler(int type) {
    if (type >= 0 && type < 3) {
        irq_count[type] = irq_count[type] + 1;
        flash_device(type);
        draw_device(type);
    }
}

int main() {
    int key; int lk; int t; int total;
    lk = 0; t = 0;
    irq_count[0] = 0; irq_count[1] = 0; irq_count[2] = 0;

    draw_device(0);
    draw_device(1);
    draw_device(2);

    while (t < 100000) {
        key = *KEYBOARD;
        if (key != 0 && key != lk) {
            t = 0;
            if (key == 13) {
                total = irq_count[0] + irq_count[1] + irq_count[2];
                return total;
            }
            if (key == 116 || key == 84) irq_handler(0);
            if (key == 107 || key == 75) irq_handler(1);
            if (key == 115 || key == 83) irq_handler(2);
        }
        lk = key;
        t = t + 1;
    }
    return 0;
}
```

1.5.13 Coroutines

```
// Coroutines Visuelles - Solution
// Cochez "Capturer clavier", appuyez Espace pour chaque step

int *SCREEN = (int*)0x00400000;
int *KEYBOARD = (int*)0x00402600;

int current_task = 0;
int task_a_val = 0;
int task_b_val = 0;
int steps = 0;

void set_pixel(int x, int y) {
    int *ptr; int bit_pos;
    ptr = SCREEN + (y << 3) + (y << 1) + (x >> 5);
    bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
}
```

```
void clear_rect(int x, int y, int w, int h) {
    int i; int j; int px; int py; int *ptr; int bit_pos;
    for (i = 0; i < w; i = i + 1) {
        for (j = 0; j < h; j = j + 1) {
            px = x + i; py = y + j;
            ptr = SCREEN + (py << 3) + (py << 1) + (px >> 5);
            bit_pos = ((px >> 3) & 3) * 8 + 7 - (px & 7);
            *ptr = *ptr & (0xFFFFFFFF ^ (1 << bit_pos));
        }
    }
}

void fill_rect(int x, int y, int w, int h) {
    int i; int j;
    for (i = 0; i < w; i = i + 1) {
        for (j = 0; j < h; j = j + 1) {
            set_pixel(x + i, y + j);
        }
    }
}

void draw_box(int x, int y, int w, int h) {
    int i;
    for (i = 0; i < w; i = i + 1) { set_pixel(x + i, y); set_pixel(x + i, y + h - 1); }
    for (i = 0; i < h; i = i + 1) { set_pixel(x, y + i); set_pixel(x + w - 1, y + i); }
}

void draw_digit(int x, int y, int d) {
    clear_rect(x, y, 4, 6);
    if (d != 1 && d != 4) { set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y); }
    if (d == 0 || d == 4 || d == 5 || d == 6 || d == 8 || d == 9) set_pixel(x, y+1);
    if (d != 5 && d != 6) set_pixel(x+2, y+1);
    if (d != 0 && d != 1 && d != 7) { set_pixel(x, y+2); set_pixel(x+1, y+2); set_pixel(x+2,
    if (d == 0 || d == 2 || d == 6 || d == 8) set_pixel(x, y+3);
    if (d != 2) set_pixel(x+2, y+3);
    if (d != 1 && d != 4 && d != 7) { set_pixel(x, y+4); set_pixel(x+1, y+4); set_pixel(x+2,
    if (d == 1) { set_pixel(x+1, y); set_pixel(x+1, y+1); set_pixel(x+1, y+2); set_pixel(x+1,
}

void draw_letter_A(int x, int y) {
    set_pixel(x+1, y);
    set_pixel(x, y+1); set_pixel(x+2, y+1);
    set_pixel(x, y+2); set_pixel(x+1, y+2); set_pixel(x+2, y+2);
    set_pixel(x, y+3); set_pixel(x+2, y+3);
    set_pixel(x, y+4); set_pixel(x+2, y+4);
}

void draw_letter_B(int x, int y) {
    set_pixel(x, y); set_pixel(x+1, y);
    set_pixel(x, y+1); set_pixel(x+2, y+1);
    set_pixel(x, y+2); set_pixel(x+1, y+2);
    set_pixel(x, y+3); set_pixel(x+2, y+3);
    set_pixel(x, y+4); set_pixel(x+1, y+4);
}

void draw_arrow(int x, int y) {
```

```
    set_pixel(x, y+2);
    set_pixel(x+1, y+1); set_pixel(x+1, y+2); set_pixel(x+1, y+3);
    set_pixel(x+2, y); set_pixel(x+2, y+2); set_pixel(x+2, y+4);
    set_pixel(x+3, y+1); set_pixel(x+3, y+2); set_pixel(x+3, y+3);
    set_pixel(x+4, y+2);
}

void draw_task(int idx, int active) {
    int x;
    x = idx * 35 + 4;
    clear_rect(x, 4, 30, 20);
    if (active) {
        fill_rect(x, 4, 30, 20);
        // Dessiner en inverse (effacer les pixels pour la lettre/chiffre)
        clear_rect(x + 12, 7, 6, 6);
        clear_rect(x + 12, 15, 5, 6);
    } else {
        draw_box(x, 4, 30, 20);
    }
    if (idx == 0) {
        if (active) clear_rect(x + 13, 7, 4, 5);
        else draw_letter_A(x + 13, 7);
        draw_digit(x + 13, 15, task_a_val);
    } else {
        if (active) clear_rect(x + 13, 7, 4, 5);
        else draw_letter_B(x + 13, 7);
        draw_digit(x + 13, 15, task_b_val);
    }
}

void draw_all() {
    draw_task(0, current_task == 0);
    draw_task(1, current_task == 1);
    // Fleche entre les taches
    if (current_task == 0) {
        clear_rect(32, 12, 8, 6);
        draw_arrow(32, 12);
    } else {
        clear_rect(32, 12, 8, 6);
        // Fleche inversee
        set_pixel(36, 14);
        set_pixel(35, 13); set_pixel(35, 14); set_pixel(35, 15);
        set_pixel(34, 12); set_pixel(34, 14); set_pixel(34, 16);
        set_pixel(33, 13); set_pixel(33, 14); set_pixel(33, 15);
        set_pixel(32, 14);
    }
}

int step() {
    if (current_task == 0) {
        task_a_val = task_a_val + 1;
        if (task_a_val > 3) return 0;
        current_task = 1;
    } else {
        task_b_val = task_b_val + 1;
        if (task_b_val > 3) return 0;
```

```
        current_task = 0;
    }
    steps = steps + 1;
    return 1;
}

int main() {
    int key; int lk; int t; int running;
    lk = 0; t = 0; running = 1;

    draw_all();

    while (t < 100000 && running) {
        key = *KEYBOARD;
        if (key != 0 && key != lk) {
            t = 0;
            if (key == 32) {
                running = step();
                draw_all();
            }
            if (key == 13) {
                return steps;
            }
        }
        lk = key;
        t = t + 1;
    }
    // Flash final
    fill_rect(0, 0, 80, 30);
    return steps;
}
```

1.5.14 Scheduler

```
// Scheduler Round-Robin Visuel - Solution
// Cochez "Capturer clavier", Espace pour chaque tick

int *SCREEN = (int*)0x00400000;
int *KEYBOARD = (int*)0x00402600;

int proc_time[3];
int proc_state[3];
int current_proc = 0;
int quantum_left = 2;
int switches = 0;
int ticks = 0;

void set_pixel(int x, int y) {
    int *ptr; int bit_pos;
    ptr = SCREEN + (y << 3) + (y << 1) + (x >> 5);
    bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
}

void clear_rect(int x, int y, int w, int h) {
```

```
    int i; int j; int px; int py; int *ptr; int bit_pos;
    for (i = 0; i < w; i = i + 1) {
        for (j = 0; j < h; j = j + 1) {
            px = x + i; py = y + j;
            ptr = SCREEN + (py << 3) + (py << 1) + (px >> 5);
            bit_pos = ((px >> 3) & 3) * 8 + 7 - (px & 7);
            *ptr = *ptr & (0xFFFFFFFF ^ (1 << bit_pos));
        }
    }
}

void fill_rect(int x, int y, int w, int h) {
    int i; int j;
    for (i = 0; i < w; i = i + 1) {
        for (j = 0; j < h; j = j + 1) {
            set_pixel(x + i, y + j);
        }
    }
}

void draw_box(int x, int y, int w, int h) {
    int i;
    for (i = 0; i < w; i = i + 1) { set_pixel(x + i, y); set_pixel(x + i, y + h - 1); }
    for (i = 0; i < h; i = i + 1) { set_pixel(x, y + i); set_pixel(x + w - 1, y + i); }
}

void draw_digit(int x, int y, int d) {
    clear_rect(x, y, 4, 6);
    if (d != 1 && d != 4) { set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y); }
    if (d == 0 || d == 4 || d == 5 || d == 6 || d == 8 || d == 9) set_pixel(x, y+1);
    if (d != 5 && d != 6) set_pixel(x+2, y+1);
    if (d != 0 && d != 1 && d != 7) { set_pixel(x, y+2); set_pixel(x+1, y+2); set_pixel(x+2,
    if (d == 0 || d == 2 || d == 6 || d == 8) set_pixel(x, y+3);
    if (d != 2) set_pixel(x+2, y+3);
    if (d != 1 && d != 4 && d != 7) { set_pixel(x, y+4); set_pixel(x+1, y+4); set_pixel(x+2,
    if (d == 1) { set_pixel(x+1, y); set_pixel(x+1, y+1); set_pixel(x+1, y+2); set_pixel(x+1,
}

void draw_P(int x, int y) {
    set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y);
    set_pixel(x, y+1); set_pixel(x+2, y+1);
    set_pixel(x, y+2); set_pixel(x+1, y+2);
    set_pixel(x, y+3); set_pixel(x, y+4);
}

void draw_proc(int idx, int active) {
    int x; int y; int t; int i;
    x = idx * 32 + 4;
    y = 4;

    clear_rect(x, y, 28, 24);

    if (active) {
        fill_rect(x, y, 28, 24);
        clear_rect(x + 2, y + 2, 24, 20);
    } else {
```

```
        draw_box(x, y, 28, 24);
    }

    // P et numero
    draw_P(x + 4, y + 4);
    draw_digit(x + 10, y + 4, idx);

    // Barre de temps restant
    t = proc_time[idx];
    if (t > 0) {
        for (i = 0; i < t; i = i + 1) {
            fill_rect(x + 4 + i * 5, y + 14, 4, 6);
        }
    }

    // X si termine
    if (proc_state[idx] == 2) {
        set_pixel(x + 8, y + 14); set_pixel(x + 12, y + 14);
        set_pixel(x + 9, y + 15); set_pixel(x + 11, y + 15);
        set_pixel(x + 10, y + 16);
        set_pixel(x + 9, y + 17); set_pixel(x + 11, y + 17);
        set_pixel(x + 8, y + 18); set_pixel(x + 12, y + 18);
    }
}

void draw_all() {
    int i;
    for (i = 0; i < 3; i = i + 1) {
        draw_proc(i, i == current_proc && proc_state[i] != 2);
    }

    // Quantum: Q=N
    clear_rect(4, 32, 20, 6);
    // Q
    set_pixel(5, 32); set_pixel(6, 32); set_pixel(7, 32);
    set_pixel(4, 33); set_pixel(8, 33);
    set_pixel(4, 34); set_pixel(8, 34);
    set_pixel(4, 35); set_pixel(6, 35); set_pixel(8, 35);
    set_pixel(5, 36); set_pixel(6, 36); set_pixel(8, 36);
    draw_digit(12, 32, quantum_left);

    // Switches: SW=N
    clear_rect(30, 32, 30, 6);
    // S
    set_pixel(31, 32); set_pixel(32, 32); set_pixel(33, 32);
    set_pixel(30, 33);
    set_pixel(31, 34); set_pixel(32, 34);
    set_pixel(33, 35);
    set_pixel(30, 36); set_pixel(31, 36); set_pixel(32, 36);
    // W
    set_pixel(36, 32); set_pixel(40, 32);
    set_pixel(36, 33); set_pixel(40, 33);
    set_pixel(36, 34); set_pixel(38, 34); set_pixel(40, 34);
    set_pixel(36, 35); set_pixel(38, 35); set_pixel(40, 35);
    set_pixel(37, 36); set_pixel(39, 36);
    draw_digit(44, 32, switches);
```

```
    // Ticks: T=N
    clear_rect(60, 32, 20, 6);
    // T
    set_pixel(60, 32); set_pixel(61, 32); set_pixel(62, 32);
    set_pixel(61, 33); set_pixel(61, 34); set_pixel(61, 35); set_pixel(61, 36);
    draw_digit(66, 32, ticks);
}

int find_next(int from) {
    int i; int next;
    for (i = 1; i <= 3; i = i + 1) {
        next = (from + i) % 3;
        if (proc_state[next] == 0 && proc_time[next] > 0) return next;
    }
    return from;
}

int tick() {
    int next; int all_done;

    // Verifier si tous termines
    all_done = 1;
    if (proc_state[0] != 2) all_done = 0;
    if (proc_state[1] != 2) all_done = 0;
    if (proc_state[2] != 2) all_done = 0;
    if (all_done) return 0;

    ticks = ticks + 1;

    // Executer processus courant
    if (proc_time[current_proc] > 0) {
        proc_time[current_proc] = proc_time[current_proc] - 1;
        quantum_left = quantum_left - 1;

        if (proc_time[current_proc] == 0) {
            proc_state[current_proc] = 2;
            quantum_left = 0;
        }
    }

    // Context switch si quantum epuise
    if (quantum_left <= 0) {
        next = find_next(current_proc);
        if (next != current_proc) {
            current_proc = next;
            switches = switches + 1;
        }
        quantum_left = 2;
    }

    return 1;
}

int main() {
    int key; int lk; int t; int running;
```

```
    proc_time[0] = 3; proc_time[1] = 2; proc_time[2] = 4;
    proc_state[0] = 0; proc_state[1] = 0; proc_state[2] = 0;

    lk = 0; t = 0; running = 1;
    draw_all();

    while (t < 100000) {
        key = *KEYBOARD;
        if (key != 0 && key != lk) {
            t = 0;
            if (key == 32) {
                running = tick();
                draw_all();
            }
            if (key == 13) {
                return switches;
            }
        }
        lk = key;
        t = t + 1;
    }

    return switches;
}
```

1.5.15 Projet 1: Mini-OS Shell

```
// Mini-OS Shell - Solution
// Cochez "Capturer clavier"

int *SCREEN = (int*)0x00400000;
int *KEYBOARD = (int*)0x00402600;

void set_pixel(int x, int y) {
    int *ptr; int bit_pos;
    ptr = SCREEN + (y << 3) + (y << 1) + (x >> 5);
    bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
}

void clear_screen() {
    int i; int *ptr;
    ptr = SCREEN;
    for (i = 0; i < 2400; i = i + 1) {
        *ptr = 0;
        ptr = ptr + 1;
    }
}

void draw_digit(int x, int y, int d) {
    if (d != 1 && d != 4) { set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y); }
    if (d == 0 || d == 4 || d == 5 || d == 6 || d == 8 || d == 9) set_pixel(x, y+1);
    if (d != 5 && d != 6) set_pixel(x+2, y+1);
    if (d != 0 && d != 1 && d != 7) { set_pixel(x, y+2); set_pixel(x+1, y+2); set_pixel(x+2,
```

```
    if (d == 0 || d == 2 || d == 6 || d == 8) set_pixel(x, y+3);
    if (d != 2) set_pixel(x+2, y+3);
    if (d != 1 && d != 4 && d != 7) { set_pixel(x, y+4); set_pixel(x+1, y+4); set_pixel(x+2,
    if (d == 1) { set_pixel(x+1, y); set_pixel(x+1, y+1); set_pixel(x+1, y+2); set_pixel(x+1,
}

void draw_plus(int x, int y) {
    set_pixel(x+1, y); set_pixel(x, y+1); set_pixel(x+1, y+1); set_pixel(x+2, y+1); set_pixel
}

void draw_equal(int x, int y) {
    set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y);
    set_pixel(x, y+2); set_pixel(x+1, y+2); set_pixel(x+2, y+2);
}

void draw_H(int x, int y) {
    set_pixel(x, y); set_pixel(x, y+1); set_pixel(x, y+2); set_pixel(x, y+3); set_pixel(x, y+
    set_pixel(x+1, y+2);
    set_pixel(x+2, y); set_pixel(x+2, y+1); set_pixel(x+2, y+2); set_pixel(x+2, y+3); set_pix
}

void draw_I(int x, int y) {
    set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y);
    set_pixel(x+1, y+1); set_pixel(x+1, y+2); set_pixel(x+1, y+3);
    set_pixel(x, y+4); set_pixel(x+1, y+4); set_pixel(x+2, y+4);
}

void draw_box(int x, int y, int w, int h) {
    int i;
    for (i = 0; i < w; i = i + 1) { set_pixel(x + i, y); set_pixel(x + i, y + h - 1); }
    for (i = 0; i < h; i = i + 1) { set_pixel(x, y + i); set_pixel(x + w - 1, y + i); }
}

void draw_menu() {
    // Titre: MENU
    // M
    set_pixel(4, 4); set_pixel(4, 5); set_pixel(4, 6); set_pixel(4, 7); set_pixel(4, 8);
    set_pixel(5, 5); set_pixel(6, 6); set_pixel(7, 5);
    set_pixel(8, 4); set_pixel(8, 5); set_pixel(8, 6); set_pixel(8, 7); set_pixel(8, 8);
    // E
    set_pixel(11, 4); set_pixel(12, 4); set_pixel(13, 4);
    set_pixel(11, 5); set_pixel(11, 6); set_pixel(12, 6); set_pixel(11, 7);
    set_pixel(11, 8); set_pixel(12, 8); set_pixel(13, 8);
    // N
    set_pixel(16, 4); set_pixel(16, 5); set_pixel(16, 6); set_pixel(16, 7); set_pixel(16, 8);
    set_pixel(17, 5); set_pixel(18, 6); set_pixel(19, 7);
    set_pixel(20, 4); set_pixel(20, 5); set_pixel(20, 6); set_pixel(20, 7); set_pixel(20, 8);
    // U
    set_pixel(23, 4); set_pixel(23, 5); set_pixel(23, 6); set_pixel(23, 7);
    set_pixel(24, 8); set_pixel(25, 8);
    set_pixel(26, 4); set_pixel(26, 5); set_pixel(26, 6); set_pixel(26, 7);

    // Option 1: CALC
    draw_box(4, 14, 30, 12);
    draw_digit(8, 17, 1);
    // C
```

```
    set_pixel(15, 17); set_pixel(16, 17); set_pixel(17, 17);
    set_pixel(14, 18); set_pixel(14, 19); set_pixel(14, 20);
    set_pixel(15, 21); set_pixel(16, 21); set_pixel(17, 21);

    // Option 2: COUNT
    draw_box(4, 28, 30, 12);
    draw_digit(8, 31, 2);
    // #
    set_pixel(15, 31); set_pixel(17, 31);
    set_pixel(14, 32); set_pixel(15, 32); set_pixel(16, 32); set_pixel(17, 32); set_pixel(18,
    set_pixel(15, 33); set_pixel(17, 33);
    set_pixel(14, 34); set_pixel(15, 34); set_pixel(16, 34); set_pixel(17, 34); set_pixel(18,
    set_pixel(15, 35); set_pixel(17, 35);

    // Option 3: MSG
    draw_box(4, 42, 30, 12);
    draw_digit(8, 45, 3);
    draw_H(15, 45);
    draw_I(20, 45);

    // Option 0: QUIT
    draw_box(4, 56, 30, 12);
    draw_digit(8, 59, 0);
    // X
    set_pixel(15, 59); set_pixel(19, 59);
    set_pixel(16, 60); set_pixel(18, 60);
    set_pixel(17, 61);
    set_pixel(16, 62); set_pixel(18, 62);
    set_pixel(15, 63); set_pixel(19, 63);
}

void delay() {
    int i;
    for (i = 0; i < 50000; i = i + 1) { }
}

void wait_key() {
    int k;
    while (1) {
        k = *KEYBOARD;
        if (k != 0) return;
    }
}

void app_calc() {
    clear_screen();
    // Titre
    // C
    set_pixel(5, 4); set_pixel(6, 4); set_pixel(7, 4);
    set_pixel(4, 5); set_pixel(4, 6); set_pixel(4, 7);
    set_pixel(5, 8); set_pixel(6, 8); set_pixel(7, 8);
    // A
    set_pixel(11, 4); set_pixel(10, 5); set_pixel(12, 5);
    set_pixel(10, 6); set_pixel(11, 6); set_pixel(12, 6);
    set_pixel(10, 7); set_pixel(12, 7); set_pixel(10, 8); set_pixel(12, 8);
    // L
```

```
    set_pixel(15, 4); set_pixel(15, 5); set_pixel(15, 6); set_pixel(15, 7);
    set_pixel(15, 8); set_pixel(16, 8); set_pixel(17, 8);
    // C
    set_pixel(21, 4); set_pixel(22, 4); set_pixel(23, 4);
    set_pixel(20, 5); set_pixel(20, 6); set_pixel(20, 7);
    set_pixel(21, 8); set_pixel(22, 8); set_pixel(23, 8);

    // 3 + 5 = 8
    draw_digit(10, 20, 3);
    draw_plus(16, 20);
    draw_digit(22, 20, 5);
    draw_equal(28, 20);
    draw_digit(34, 20, 8);

    wait_key();
}

void app_count() {
    int i;
    clear_screen();
    // Titre: COUNT
    // #
    set_pixel(5, 4); set_pixel(7, 4);
    set_pixel(4, 5); set_pixel(5, 5); set_pixel(6, 5); set_pixel(7, 5); set_pixel(8, 5);
    set_pixel(5, 6); set_pixel(7, 6);
    set_pixel(4, 7); set_pixel(5, 7); set_pixel(6, 7); set_pixel(7, 7); set_pixel(8, 7);
    set_pixel(5, 8); set_pixel(7, 8);

    for (i = 0; i < 6; i = i + 1) {
        draw_digit(10 + i * 6, 20, i);
        delay();
    }

    wait_key();
}

void app_msg() {
    clear_screen();
    // Titre: MSG
    // M
    set_pixel(4, 4); set_pixel(4, 5); set_pixel(4, 6); set_pixel(4, 7); set_pixel(4, 8);
    set_pixel(5, 5); set_pixel(6, 6); set_pixel(7, 5);
    set_pixel(8, 4); set_pixel(8, 5); set_pixel(8, 6); set_pixel(8, 7); set_pixel(8, 8);
    // S
    set_pixel(12, 4); set_pixel(13, 4); set_pixel(11, 5);
    set_pixel(12, 6); set_pixel(13, 7);
    set_pixel(11, 8); set_pixel(12, 8);
    // G
    set_pixel(17, 4); set_pixel(18, 4); set_pixel(19, 4);
    set_pixel(16, 5); set_pixel(16, 6); set_pixel(18, 6); set_pixel(19, 6);
    set_pixel(16, 7); set_pixel(19, 7);
    set_pixel(17, 8); set_pixel(18, 8); set_pixel(19, 8);

    // Grand HI
    // H
    set_pixel(10, 18); set_pixel(10, 19); set_pixel(10, 20); set_pixel(10, 21); set_pixel(10,
```

```
    set_pixel(10, 23); set_pixel(10, 24); set_pixel(10, 25); set_pixel(10, 26); set_pixel(10,
    set_pixel(11, 22); set_pixel(12, 22); set_pixel(13, 22);
    set_pixel(14, 18); set_pixel(14, 19); set_pixel(14, 20); set_pixel(14, 21); set_pixel(14,
    set_pixel(14, 23); set_pixel(14, 24); set_pixel(14, 25); set_pixel(14, 26); set_pixel(14,
    // I
    set_pixel(18, 18); set_pixel(19, 18); set_pixel(20, 18); set_pixel(21, 18); set_pixel(22,
    set_pixel(20, 19); set_pixel(20, 20); set_pixel(20, 21); set_pixel(20, 22);
    set_pixel(20, 23); set_pixel(20, 24); set_pixel(20, 25); set_pixel(20, 26);
    set_pixel(18, 27); set_pixel(19, 27); set_pixel(20, 27); set_pixel(21, 27); set_pixel(22,

    wait_key();
}

int main() {
    int key; int lk; int running;
    running = 1;
    lk = 0;

    while (running) {
        clear_screen();
        draw_menu();

        while (1) {
            key = *KEYBOARD;
            if (key != 0 && key != lk) {
                if (key == 49) { app_calc(); break; }
                if (key == 50) { app_count(); break; }
                if (key == 51) { app_msg(); break; }
                if (key == 48) { running = 0; break; }
            }
            lk = key;
        }
    }

    clear_screen();
    return 0;
}
```

1.5.16 Projet 2: Task Manager

```
// Gestionnaire de Tâches - Solution
// Cochez "Capturer clavier"

int *SCREEN = (int*)0x00400000;
int *KEYBOARD = (int*)0x00402600;

int proc_work[4];
int proc_state[4];
int proc_done[4];
int current = 0;
int quantum_left = 2;
int switches = 0;
int ticks = 0;
int auto_run = 0;
```

```
void set_pixel(int x, int y) {
    int *ptr; int bit_pos;
    ptr = SCREEN + (y << 3) + (y << 1) + (x >> 5);
    bit_pos = ((x >> 3) & 3) * 8 + 7 - (x & 7);
    *ptr = *ptr | (1 << bit_pos);
}

void clear_rect(int x, int y, int w, int h) {
    int i; int j; int px; int py; int *ptr; int bit_pos;
    for (i = 0; i < w; i = i + 1) {
        for (j = 0; j < h; j = j + 1) {
            px = x + i; py = y + j;
            ptr = SCREEN + (py << 3) + (py << 1) + (px >> 5);
            bit_pos = ((px >> 3) & 3) * 8 + 7 - (px & 7);
            *ptr = *ptr & (0xFFFFFFFF ^ (1 << bit_pos));
        }
    }
}

void fill_rect(int x, int y, int w, int h) {
    int i; int j;
    for (i = 0; i < w; i = i + 1) {
        for (j = 0; j < h; j = j + 1) {
            set_pixel(x + i, y + j);
        }
    }
}

void draw_box(int x, int y, int w, int h) {
    int i;
    for (i = 0; i < w; i = i + 1) { set_pixel(x + i, y); set_pixel(x + i, y + h - 1); }
    for (i = 0; i < h; i = i + 1) { set_pixel(x, y + i); set_pixel(x + w - 1, y + i); }
}

void draw_digit(int x, int y, int d) {
    clear_rect(x, y, 4, 6);
    if (d != 1 && d != 4) { set_pixel(x, y); set_pixel(x+1, y); set_pixel(x+2, y); }
    if (d == 0 || d == 4 || d == 5 || d == 6 || d == 8 || d == 9) set_pixel(x, y+1);
    if (d != 5 && d != 6) set_pixel(x+2, y+1);
    if (d != 0 && d != 1 && d != 7) { set_pixel(x, y+2); set_pixel(x+1, y+2); set_pixel(x+2,
    if (d == 0 || d == 2 || d == 6 || d == 8) set_pixel(x, y+3);
    if (d != 2) set_pixel(x+2, y+3);
    if (d != 1 && d != 4 && d != 7) { set_pixel(x, y+4); set_pixel(x+1, y+4); set_pixel(x+2,
    if (d == 1) { set_pixel(x+1, y); set_pixel(x+1, y+1); set_pixel(x+1, y+2); set_pixel(x+1,
}

void draw_proc(int idx) {
    int x; int w; int i; int active;
    x = idx * 40 + 4;

    clear_rect(x, 4, 36, 55);

    active = (idx == current && proc_state[idx] == 1);

    // Cadre - double si running
    draw_box(x, 4, 36, 55);
```

```
if (active) {
    draw_box(x + 2, 6, 32, 51);
}

// P et numero (grand)
fill_rect(x + 8, 10, 2, 9);
fill_rect(x + 10, 10, 4, 2);
fill_rect(x + 14, 10, 2, 5);
fill_rect(x + 10, 14, 4, 2);

draw_digit(x + 20, 12, idx);

// Etat: R=Ready *=Run B=Block X=Done
clear_rect(x + 10, 24, 16, 10);
if (proc_state[idx] == 0) {
    // R
    fill_rect(x + 12, 25, 2, 8);
    fill_rect(x + 14, 25, 4, 2);
    fill_rect(x + 18, 25, 2, 4);
    fill_rect(x + 14, 28, 4, 2);
    fill_rect(x + 16, 30, 2, 3);
}
if (proc_state[idx] == 1) {
    // * etoile
    fill_rect(x + 14, 26, 2, 6);
    fill_rect(x + 12, 28, 6, 2);
}
if (proc_state[idx] == 2) {
    // B
    fill_rect(x + 12, 25, 2, 8);
    fill_rect(x + 14, 25, 4, 2);
    fill_rect(x + 14, 28, 4, 2);
    fill_rect(x + 14, 31, 4, 2);
    fill_rect(x + 18, 26, 2, 2);
    fill_rect(x + 18, 29, 2, 2);
}
if (proc_state[idx] == 3) {
    // X
    fill_rect(x + 12, 25, 2, 2); fill_rect(x + 18, 25, 2, 2);
    fill_rect(x + 14, 27, 2, 2); fill_rect(x + 16, 27, 2, 2);
    fill_rect(x + 14, 29, 4, 2);
    fill_rect(x + 12, 31, 2, 2); fill_rect(x + 18, 31, 2, 2);
}

// Barre travail restant
w = proc_work[idx];
for (i = 0; i < 4; i = i + 1) {
    if (i < w) {
        fill_rect(x + 6 + i * 7, 38, 5, 6);
    } else {
        draw_box(x + 6 + i * 7, 38, 5, 6);
    }
}

// Compteur
draw_digit(x + 15, 50, proc_done[idx] % 10);
```

```
}

void draw_info() {
    clear_rect(4, 62, 156, 8);
    // Q:
    fill_rect(6, 63, 2, 5); fill_rect(8, 63, 3, 2); fill_rect(11, 63, 2, 5);
    fill_rect(8, 66, 3, 2); fill_rect(10, 67, 3, 2);
    draw_digit(16, 63, quantum_left);
    // S:
    fill_rect(30, 63, 5, 2); fill_rect(28, 65, 2, 2);
    fill_rect(30, 66, 3, 2); fill_rect(33, 67, 2, 2);
    fill_rect(28, 68, 5, 2);
    draw_digit(38, 63, switches % 10);
    // T:
    fill_rect(52, 63, 7, 2); fill_rect(54, 65, 3, 4);
    draw_digit(62, 63, ticks % 10);
}

int find_next(int from) {
    int i; int next;
    for (i = 1; i <= 4; i = i + 1) {
        next = (from + i) % 4;
        if (proc_state[next] == 0 && proc_work[next] > 0) return next;
    }
    return from;
}

int all_done() {
    int i;
    for (i = 0; i < 4; i = i + 1) {
        if (proc_state[i] != 3 && proc_state[i] != 2) {
            if (proc_work[i] > 0) return 0;
        }
    }
    return 1;
}

void do_tick() {
    int next;
    if (all_done()) return;
    ticks = ticks + 1;
    if (proc_state[current] == 0) proc_state[current] = 1;
    if (proc_state[current] == 1 && proc_work[current] > 0) {
        proc_work[current] = proc_work[current] - 1;
        proc_done[current] = proc_done[current] + 1;
        quantum_left = quantum_left - 1;
        if (proc_work[current] == 0) {
            proc_state[current] = 3;
            quantum_left = 0;
        }
    }
    if (quantum_left <= 0 || proc_state[current] == 2 || proc_state[current] == 3) {
        if (proc_state[current] == 1) proc_state[current] = 0;
        next = find_next(current);
        if (next != current) {
            current = next;
```

```
            switches = switches + 1;
        }
        quantum_left = 2;
    }
}

void toggle_block(int idx) {
    if (proc_state[idx] == 0 || proc_state[idx] == 1) {
        proc_state[idx] = 2;
        if (idx == current) quantum_left = 0;
    } else if (proc_state[idx] == 2) {
        proc_state[idx] = 0;
    }
}

void draw_all() {
    int i;
    for (i = 0; i < 4; i = i + 1) draw_proc(i);
    draw_info();
}

int main() {
    int key; int lk; int delay;
    proc_work[0] = 3; proc_work[1] = 2; proc_work[2] = 4; proc_work[3] = 3;
    proc_state[0] = 0; proc_state[1] = 0; proc_state[2] = 0; proc_state[3] = 0;
    proc_done[0] = 0; proc_done[1] = 0; proc_done[2] = 0; proc_done[3] = 0;
    lk = 0; delay = 0;
    draw_all();
    while (1) {
        key = *KEYBOARD;
        if (key != 0 && key != lk) {
            if (key == 32) { do_tick(); draw_all(); }
            if (key == 65) { auto_run = 1 - auto_run; }
            if (key == 48) { toggle_block(0); draw_all(); }
            if (key == 49) { toggle_block(1); draw_all(); }
            if (key == 50) { toggle_block(2); draw_all(); }
            if (key == 51) { toggle_block(3); draw_all(); }
            if (key == 13) return switches;
        }
        lk = key;
        if (auto_run) {
            delay = delay + 1;
            if (delay > 3000) { delay = 0; do_tick(); draw_all(); }
        }
    }
    return switches;
}
```
