COMS10015 hand-out: exam-style revision questions

Part I: Mathematical preliminaries

▷ **Q1.** We studied representation of unsigned integers using a base-b positional number system. Which of the following literals

`A:` 10101
`B:` 11111
`C:` 11120
`D:` 12200
`E:` 12345

represents the unsigned decimal integer $123_{(10)}$ in base-3 (or ternary, digits in which are termed trits).

▷ **Q2.** Imagine that two signed, 8-bit integers x and y are represented using two's-complement and sign-magnitude respectively, and both of which have the decimal value $51_{(10)}$. If the most-significant bit of both x and y is set to 1, what are their new (decimal) values?

`A:` $-77_{(10)}$ and $179_{(10)}$
`B:` $-77_{(10)}$ and $-51_{(10)}$
`C:` $-51_{(10)}$ and $-77_{(10)}$
`D:` $179_{(10)}$ and $179_{(10)}$
`E:` $179_{(10)}$ and $-51_{(10)}$

▷ **Q3.** Imagine that two signed, 16-bit integers x and y are represented using two's-complement; their product $r = x \cdot y$ is a signed, 32-bit integer also represented using two's-complement. What is the largest (i.e., whose magnitude is greatest) negative value of r possible?

`A:` −0
`B:` −32768
`C:` −65535
`D:` −1073709056
`E:` −2147483648

▷ **Q4.** Imagine you write a C program that defines signed, 16-bit integer variables `x` and `y` (of type `short`) and then assigns them the decimal values $256_{(10)}$ and $4852_{(10)}$ respectively. If `x` and `y` are then cast into signed, 8-bit integers (of type `char`), which of the following

`A:` 0 and 12
`B:` 0 and −12
`C:` −1 and 256
`D:` −1 and −52
`E:` 0 and 52

identifies their decimal value? Or, put another way, which are the result of evaluating the two expressions `( char )( x )` and `( char )( y )`?

▷ **Q5.** Consider two signed, 8-bit integer variables `x` and `r` (of type `char`) used in a C program. If `x` has the decimal value $9_{(10)}$ and an assignment

```
r = ( ~x << 4 ) 0x97|
```

is executed, what is the decimal value of `r` afterwards?

`A:` $-9_{(10)}$
`B:` $-1_{(10)}$
`C:` $0_{(10)}$
`D:` $1_{(10)}$
`E:` $9_{(10)}$

▷ **Q6.** In general, some x is a fixed point of a function f if $f(x)$ equals x, i.e., if f maps x to itself. Consider the following function

```
int8_t abs( int8_t x ) {
  int8_t r;

  if( x >= 0 ) {
    r =  x;
  }
  else {
    r = -x;
  }

  return r;
}
```

implemented in C: `abs` was written in an attempt to compute the absolute value of `x`, a signed, 8-bit integer representing using two's-complement. How many of the $2^8 = 256$ possible values of `x` are fixed points of `abs`?

`A:` 0
`B:` 127
`C:` 128
`D:` 129
`E:` 256

▷ **Q7.** Imagine that within a given C function, you declare signed, 8-bit integer variables (i.e., variables whose type is `int8_t`) `x` and `r`. Assume C represents signed integers using two's-complement, and the right-shift operator yields arithmetic (rather than logical) shift: if `x` has the (decimal) value $-10_{(10)}$, what (decimal) value does `r` have after the assignment

```
r = ~( ( x >> 2 ) ^ 0xF4 )
```

is executed?

`A:` $-10_{(10)}$
`B:` $10_{(10)}$
`C:` $11_{(10)}$
`D:` $54_{(10)}$
`E:` $203_{(10)}$

▷ **Q8.** Consider two unsigned, 8-bit integer variables, `x` and `y`, as declared in some C function by using the type `uint8_t`. For how many assignments to these variables will the Hamming weight of their unsigned, 8-bit integer sum, i.e., `x + y`, be zero? Put another way, how many elements does the set

$$\{(\texttt{x}, \texttt{y}) \mid \text{HW}(\texttt{x + y}) = 0\}$$

have?

`A:` 0
`B:` 1
`C:` 255
`D:` 256
`E:` 65536

▷ **Q9.** Consider a literal $\hat{x} = 10$, which represents a value x using a base-b positional number system. Based on this information alone, which of the following values

`A:` $x = 1$
`B:` $x = 2$
`C:` $x = b$
`D:` $x = 10$
`E:` $x = 16$

is correct?

▷ **Q10.** Assuming an n-bit x and use of two's-complement representation for signed integers, which of the following identities

A: $x \wedge \neg x \equiv 0_{(10)}$
B: $x \vee \neg x \equiv -1_{(10)}$
C: $x \oplus \neg x \equiv -1_{(10)}$
D: $x + \neg x \equiv -1_{(10)}$
E: $x - \neg x \equiv -1_{(10)}$

is **not** correct?

▷ **Q11.** Classify each of the following statements as either **true** or **false**, then explain why using at most a few sentences.

a If $\mathrm{HW}(x)$ and $\mathrm{HD}(x, y)$ denote Hamming weight (of x) and Hamming distance (between x and y), one can define $\mathrm{HD}(x, y) = \mathrm{HW}(x \oplus y)$.

b If an n-bit integer x is represented using two's-complement, setting $x = -1$ implies setting each bit in the representation of x to 1.

▷ **Q12.** a For the sets $A = \{1, 2, 3\}$, $B = \{3, 4, 5\}$ and $\mathcal{U} = \{1, 2, 3, 4, 5, 6, 7, 8\}$, compute the following:

i $|A|$.
ii $A \cup B$.
iii $A \cap B$.
iv $A - B$.
v $\overline{A}$.
vi $\{x \mid 2 \cdot x \in \mathcal{U}\}$.

b For each of the following decimal integers, write down the 8-bit binary representation in sign-magnitude **and** two's-complement:

i $+0$.
ii -0.
iii $+72$.
iv -34.
v -8.
vi 240.

▷ **Q13.** For some 32-bit integer x, explain what is meant by the Hamming weight of x; write a short C function to compute the Hamming weight of a given 32-bit input.

▷ **Q14.** From the following list

A: $(x \wedge y) \oplus z$
B: $(\neg x \vee y) \oplus z$
C: $(x \vee \neg y) \oplus z$
D: $\neg(x \vee y) \oplus z$
E: $\neg\neg(x \vee y) \oplus z$

identify **each** Boolean expression that evaluates to 1 given the assignment $x = 0$, $y = 0$ and $z = 1$.

▷ **Q15.** **One** of the following equivalences

A: $(x \wedge y) \wedge z \equiv x \wedge (y \wedge z)$
B: $x \vee 1 \equiv x$
C: $x \vee \neg x \equiv 1$
D: $\neg(x \vee y) \equiv \neg x \wedge \neg y$
E: $\neg\neg x \equiv x$

is incorrect: identify which.

▷ **Q16.** The Boolean expression

$$(x \vee (z \vee y)) \wedge \neg(\neg y \wedge \neg z)$$

is equivalent to which of the following alternatives?

A: $y \vee z$
B: $((x \vee z) \vee y)) \wedge (x \vee z)$
C: $(x \wedge y) \vee (x \wedge z)$
D: $(x \vee y) \wedge \neg(x \vee z)$
E: $(x \wedge z) \vee (x \wedge y)$

▷ **Q17.** The Boolean expression

$$(x \vee y) \vee (x \wedge z)$$

is equivalent to which of the following alternatives?

A: $(x \vee y) \wedge (x \vee z)$
B: $(x \vee y) \wedge z$
C: $(x \vee y) \wedge (x \wedge z)$
D: $x \vee y$
E: $(x \wedge y) \vee x$

▷ **Q18.** A given set of Boolean operators may be termed functionally complete (or universal): this means *any* Boolean function can be expressed using a Boolean expression involving elements of the set alone. For example, because we know the NAND operator is functionally complete, we can also term the sets $\{ \overline{\wedge} \}$ and $\{\wedge, \neg\}$ functionally complete. Noting that $\not\equiv$ and $\not\Rightarrow$ denote the inverse of equivalence and implication respectively (i.e., not equivalent, and does not imply), which of the following sets

A: $\{\oplus, \vee\}$
B: $\{\Rightarrow, \not\equiv\}$
C: $\{\Rightarrow, \not\Rightarrow\}$
D: all of the above
E: none of the above

is/are functionally complete?

▷ **Q19.** How many n-input, 1-output Boolean functions are there?

A: 1
B: n
C: 2^n
D: 2^{2^n}
E: $2^{2^{2^n}}$

▷ **Q20.** Consider the equivalence

$$(y \wedge \neg x) \vee (x \wedge \neg y) \;\equiv\; (x \vee y) \wedge \neg(x \wedge y),$$

the LHS of which can be manipulated into the RHS by applying the following sequence of Boolean axioms:

$$\text{identity} \rightsquigarrow \text{inverse} \rightsquigarrow \text{distribution} \rightsquigarrow \text{commutativity} \rightsquigarrow \text{distribution} \rightsquigarrow \text{commutativity} \rightsquigarrow \text{X}$$

The final axiom is missing, i.e., replaced with X: which of the following options for X yields a valid derivation?

A: Absorption
B: Idempotency
C: Implication
D: Null
E: de Morgan

▷ **Q21.** One of the following equivalences

A: $(x \wedge y) \wedge z \;\equiv\; x \wedge (y \wedge z)$

B: $x \Rightarrow y \equiv \neg x \vee y$
C: $x \wedge (x \vee y) \equiv y$
D: $\neg(x \vee y) \equiv \neg x \wedge \neg y$
E: $x \vee 0 \equiv x$

is incorrect: identify which.

▷ **Q22.** Identify which of the following Boolean expressions

A: $x \wedge (x \vee \neg x)$
B: $(x \vee \neg x) \wedge x$
C: x
D: $\neg x$
E: $\neg((\neg x \vee \neg x) \wedge (\neg x \vee x))$

is **not** equivalent to

$$x \wedge x \vee x \wedge \neg x.$$

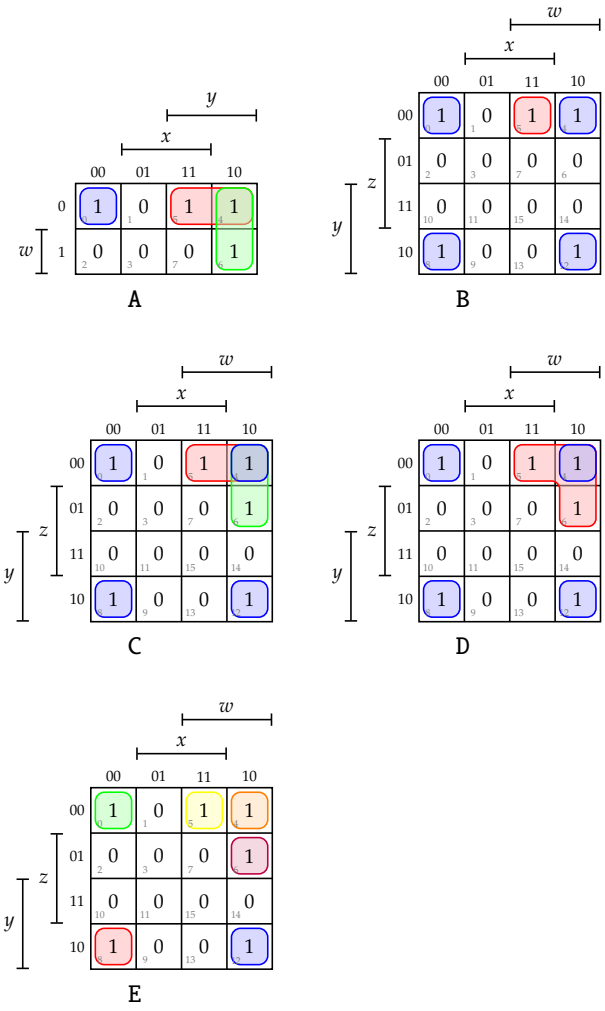


Figure 1: *A set of* 5 *different Karnaugh maps, captioned with an associated option.*

▷ **Q23.** Consider the following truth table, which describes a Boolean function f:

w	x	y	z	$f(w,x,y,z)$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Which of the Karnaugh maps shown in Figure 1 will yield the most efficient (in terms of the number of operators involved), correct Boolean expression for f?

▷ **Q24.** Identify which of the following Boolean expressions

A: $x \wedge y$
B: $x \wedge z$
C: $y \wedge z$
D: $x \wedge y \wedge z$
E: 1

is equivalent to

$$x \wedge y \vee x \wedge y \wedge z.$$

▷ **Q25.** Consider a Boolean function f with $n = 1$ input x. How many such functions are **not** idempotent, i.e., how many f exist such that $\forall x \in \{0,1\}, f(f(x)) = f(x)$ does **not** hold?

A: 0
B: 1
C: 2
D: 3
E: 4

▷ **Q26.** Consider a Boolean function f with $n = 2$ inputs x and y. How many such functions are symmetric, i.e., how many f exist such that $\forall x, y \in \{0,1\}, f(x,y) = f(y,x)$ holds?

A: 0
B: 1
C: 2
D: 8
E: 16

▷ **Q27.** Which of the following Boolean expressions

A: $x \wedge \neg x \vee y \wedge (1 \vee x)$
B: $0 \vee x \wedge y \vee y$
C: $x \wedge y$
D: y
E: $\neg((\neg x \vee (x \wedge \neg y)) \wedge \neg y)$

is **not** equivalent to

$$x \wedge (\neg x \vee y) \vee y.$$

▷ **Q28.** Which of the following Boolean expressions

A: $\neg x$
B: x
C: $\neg y$
D: y
E: 0

is equivalent to

$$(x \vee y) \wedge (x \vee \neg y).$$

▷ **Q29.** a Write out a truth table for the Boolean function

$$f(a, b, c) = (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge \neg b \wedge c),$$

then decide how many

i input combinations, and
ii outputs where $f(a, b, c) = 1$

exist in it.

b Consider the Boolean function

$$f(a, b, c, d) = \neg a \wedge b \wedge \neg c \wedge d.$$

Which of the following assignments

i $a = 0, b = 0, c = 0$ and $d = 1$,
ii $a = 0, b = 1, c = 0$ and $d = 1$,
iii $a = 1, b = 1, c = 1$ and $d = 1$,
iv $a = 0, b = 0, c = 1$ and $d = 0$.

produces the output $f(a, b, c, d) = 1$?

c Which of the following Boolean expressions

i $(a \vee b \vee d) \wedge (\neg c \vee d)$,
ii $(a \wedge b \wedge d) \vee (\neg c \wedge d)$,
iii $(a \vee b \vee d) \vee (\neg c \vee d)$.

is in Sum-of-Products (SoP) standard form?

d Identify **each** equivalence that is correct:

i $a \vee 1 \equiv a$.
ii $a \oplus 1 \equiv \neg a$.
iii $a \wedge 1 \equiv a$.
iv $\neg(a \wedge b) \equiv \neg a \vee \neg b$.

e Identify **each** equivalence that is correct:

i $\neg\neg a \equiv a$.
ii $\neg(a \wedge b) \equiv \neg a \vee \neg b$.
iii $\neg a \wedge b \equiv a \wedge \neg b$.
iv $\neg a \equiv a \oplus a$.

▷ **Q30.** a The OR form of the null axiom is $x \vee 1 \equiv 1$. Which of the following options

i $x \wedge 1 \equiv 1$,
ii $x \wedge 0 \equiv 0$,
iii $x \vee 0 \equiv 0$,

iv $x \wedge x \equiv x$,

is the dual of this axiom?

b Given the Boolean equation

$$f = \neg a \wedge \neg b \vee \neg c \vee \neg d \vee \neg e,$$

which of the following

i $\neg f = a \vee b \vee c \vee d \vee e$,
ii $\neg f = a \wedge b \wedge c \wedge d \wedge e$,
iii $\neg f = a \wedge b \wedge (c \vee d \vee e)$,
iv $\neg f = a \wedge b \vee \neg c \vee \neg d \vee \neg e$,
v $\neg f = (a \vee b) \wedge c \wedge d \wedge e$

is correct?

c If we write the de Morgan axiom in English, which of the following

i NOR is equivalent to AND if each input to AND is complemented,
ii NAND is equivalent to OR if each input to OR is complemented,
iii AND is equivalent to NOR if each input to NOR is complemented, or
iv NOR is equivalent to NAND if each input to NAND is complemented.

describes the correct equivalence?

▷ **Q31.** a Identify which **one** of these Boolean expressions

i $c \vee d \vee e$
ii $\neg c \wedge \neg d \wedge \neg e$
iii $\neg a \wedge \neg b$
iv $\neg a \wedge \neg b \wedge \neg c \wedge \neg d \wedge \neg e$

is the correct result of simplifying

$$(\neg(a \vee b) \wedge \neg(c \vee d \vee e)) \vee \neg(a \vee b).$$

b If you simplify the Boolean expression

$$(a \vee b \vee c) \wedge \neg(d \vee e) \vee (a \vee b \vee c) \wedge (d \vee e)$$

into a form that contains the fewest operators possible, which of the following options

i $a \vee b \vee c$,
ii $\neg a \wedge \neg b \wedge \neg c$,
iii $d \vee e$,
iv $\neg d \wedge \neg e$,
v none of the above

do you end up with and why?

c If you simplify the Boolean expression

$$a \wedge c \vee c \wedge (\neg a \vee a \wedge b)$$

into a form that contains the fewest operators possible, which of the following options

i $(b \wedge c) \vee c$,
ii $c \vee (a \wedge b \wedge c)$,
iii $a \wedge c$,

iv $a \vee (b \wedge c)$,
v none of the above

do you end up with and why?

d Consider the Boolean expression

$$a \wedge b \vee a \wedge b \wedge c \vee a \wedge b \wedge c \wedge d \vee a \wedge b \wedge c \wedge d \wedge e \vee a \wedge b \wedge c \wedge d \wedge e \wedge f.$$

Which of the following simplifications

i $a \wedge b \wedge c \wedge d \wedge e \wedge f$,
ii $a \wedge b \vee c \wedge d \vee e \wedge f$,
iii $a \vee b \vee c \vee d \vee e \vee f$,
iv $a \wedge b$,
v $c \wedge d$,
vi $e \wedge f$,
vii $a \vee b \wedge (c \vee d \wedge (e \vee f))$
viii $((a \vee b) \wedge c) \vee d \wedge e \vee f$

is correct?

e Given the options

i 1,
ii 2,
iii 3,
iv 4,

decide which is the least number of operator required to compute the same result as

$$f(a, b, c) = (a \wedge b) \vee a \wedge (a \vee c) \vee b \wedge (a \vee c).$$

Show how you arrived at your decision.

f Prove that

$$(\neg x \wedge y) \vee (\neg y \wedge x) \vee (\neg x \wedge \neg y) \equiv \neg x \vee \neg y.$$

g Prove that

$$(x \wedge y) \vee (y \wedge z \wedge (y \vee z)) \equiv y \wedge (x \vee z).$$

h Simplify the Boolean expression

$$\neg(a \vee b) \wedge \neg(\neg a \vee \neg b)$$

into a form which contains the fewest operators possible.

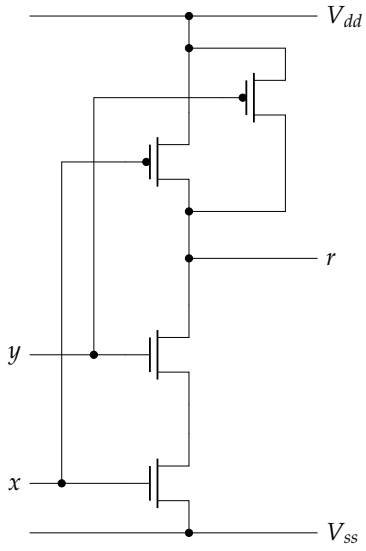
Part II: Basics of digital logic: general

▷ **Q32.** From the following list

`A:` has N-type semiconductor terminals and P-type body
`B:` has P-type semiconductor terminals and N-type body
`C:` is paired with another N-MOSFET to form a CMOS cell
`D:` has a threshold voltage above which the transistor is deemed active

identify **each** statement that correctly describes an N-MOSFET.

▷ **Q33.** Consider the following implementation of a 2-input NAND gate:

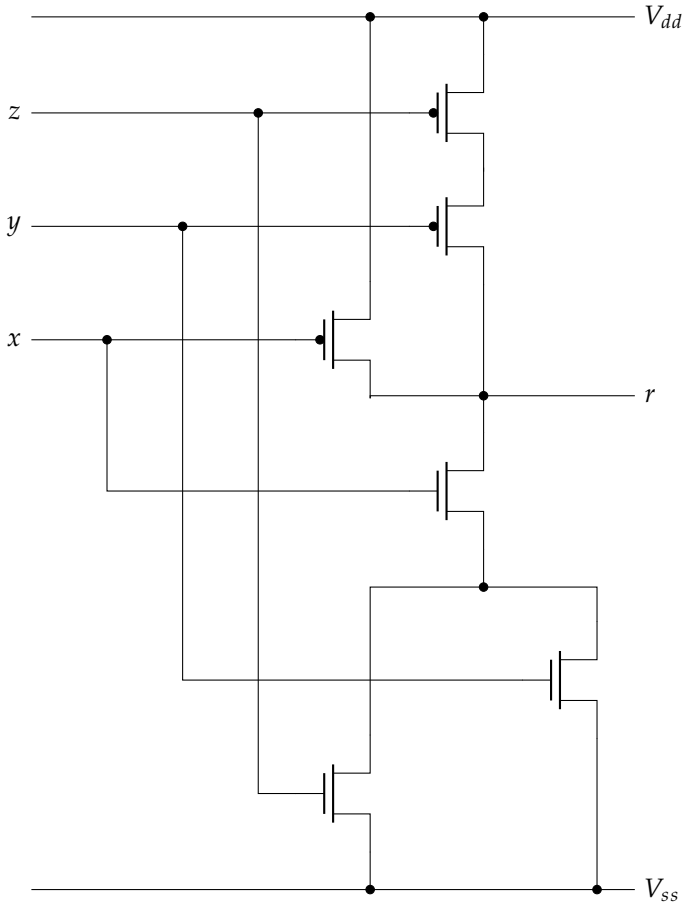


From the following list

`A:` two inputs x and y, and one output r
`B:` a pull-up network of P-MOSFET transistors
`C:` a pull-down network of BJT transistors
`D:` two power rails supplying different voltage levels
`E:` a flux capacitor

identify **each** component evident in the implementation?

▷ **Q34.** Consider the following organisation of MOSFET transistors



which implements a 3-input Boolean function $r = f(x, y, z)$. *Which* function, from the following, do you think it matches?

A: $r = x \wedge y \wedge z$
B: $r = x$
C: $r = \neg(x \wedge (y \vee z))$
D: $r = x \wedge (y \vee z)$
E: $r = x \vee y \vee z$

▷ **Q35.** Recall that a 2-input XOR operator can be described via the following truth table:

x	y	r
0	0	0
0	1	1
1	0	1
1	1	0

An implementation of this operator is realised by combining logic gate instances, e.g., for NOT, NAND, AND, NOR, and OR, while attempting to minimise the total number of underlying MOSFET-based transistors. How many such transistors do you think it uses?

A: 14
B: 16
C: 18
D: 20
E: 22

▷ **Q36.** A buffer can be described as a "pass through" logic gate: although it performs no computation (i.e., the output r matches the input x, so $r = x$), it does impose a delay (often roughly the same as a NOT gate). It may be termed a non-inverting buffer (cf. an *inverting* buffer, or NOT gate) because of this.

You are asked to implement a buffer, using an unconstrained organisation of N- and P-MOSFET transistors alone. Assuming you attempt to minimise the number used, how many transistors do you need?

A: 0
B: 2
C: 4
D: 6
E: 8

▷ **Q37.** Recalling that **?** denotes don't-care, the following truth table

f			
x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	**?**
1	1	1	1

describes a 3-input, 1-output Boolean function f st. $r = f(x, y, z)$. Which of the following Boolean expressions

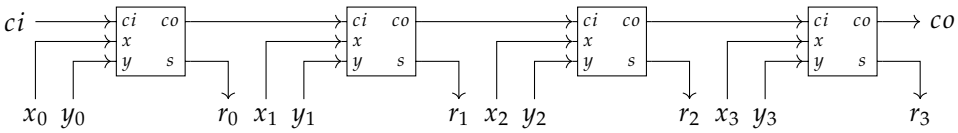
A: $(\neg x \oplus \neg y) \wedge z$
B: $(\neg x \oplus \neg y) \vee z$
C: $(\neg x \wedge \neg y) \wedge z$
D: $(\neg x \wedge \neg y) \vee z$
E: $(\neg x \vee \neg y) \wedge z$

correctly realises f?

▷ **Q38.** Imagine you want to design an 8-input, 8-bit multiplexer. Rather than do so from scratch, you intend to form the design using multiple instances of an existing 2-input, 1-bit multiplexer component. How many do you need?

A: 1
B: 8
C: 24
D: 40
E: 56

▷ **Q39.** The following diagram



illustrates a 4-bit ripple-carry adder circuit, constructed using 4 full-adder instances: it computes the sum $r = x + y + ci$, given two operands x and y and a carry-in ci, and an associated carry-out co. Given the propagation delay of NOT, AND, OR and XOR gates is 10ns, 20ns, 20ns and 60ns respectively, which of the following

A: 120ns
B: 180ns
C: 240ns
D: 280ns
E: 480ns

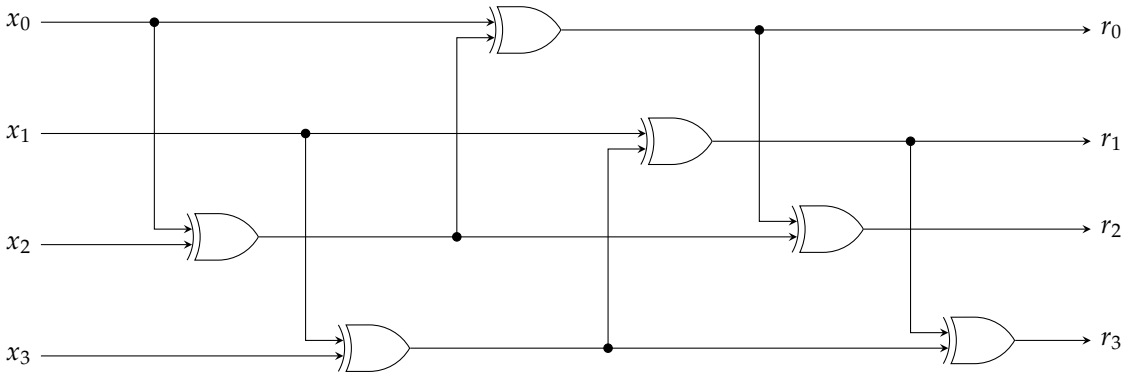
most accurately reflects the critical path of the entire circuit?

▷ **Q40.** Imagine you use the ripple-carry adder in the previous question to compute an unsigned addition within some larger circuit. Having seen your design, your friend suggests they can optimise it: they claim that replacing each full-adder instance with a half-adder instance will halve the total number of logic gates required. However, they admit the optimisation does have a disadvantage. Specifically, although any value of x can be accommodated the optimised circuit can only produce the correct output for *some* values of y. Which of the following values of y

A: -1
B: 0
C: 1
D: any $2 \leq y < 8$
E: any $8 \leq y < 16$

will produce the correct output?

▷ **Q41.** Consider the following combinatorial circuit



with a 4-bit input x and a 4-bit output r. Which of the following best describes the purpose of this circuit?

A: it computes the Hamming weight of x
B: it computes the parity of x
C: it swaps the most-significant 2-bit half of x with the least-significant 2-bit half of x

w	x	y	z	$r = f(w,x,y,z)$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	**?**
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Figure 2: *A truth table for the* 4*-input Boolean function* f*.*

`D:` it adds the most-significant 2-bit half of x to the least-significant 2-bit half of x (treating it as an unsigned, 4-bit integer)
`E:` it negates x (treating it as a signed, 4-bit integer represented using two's-complement)

▷ **Q42.** Recalling that **?** denotes don't-care, consider the truth table shown in Figure 2.

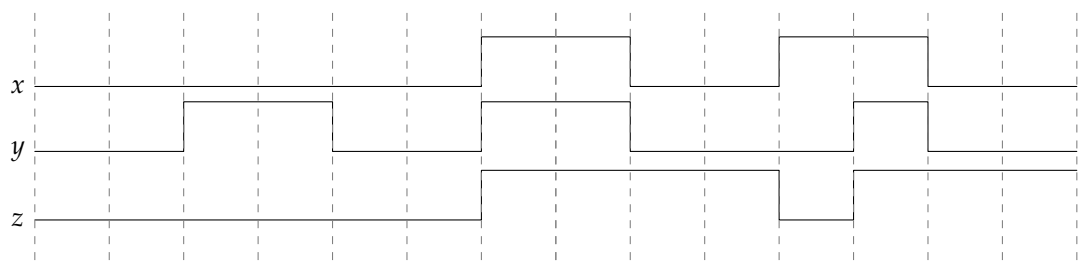
a Construction of a Karnaugh map for f demands formation of a set of groups; these (collectively) cover of all 1 entries. Assuming the most efficient approach is adopted when forming said groups, how many are required?

`A:` 1
`B:` 2
`C:` 3
`D:` 4
`E:` 6

b Using the Karnaugh map above (plus any subsequent optimisation steps you deem necessary), derive a Boolean expression for f that minimises the number of operators required. How many operators remain in said expression?

`A:` 1
`B:` 4
`C:` 5
`D:` 11
`E:` 12

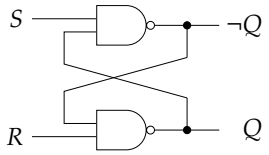
▷ **Q43.** Consider the following waveform



which details the behaviour of three signals labelled x, y and z. Which of the following components *could* the behaviour illustrated relate to?

`A:` an SR-type flip-flop
`B:` an SR-type latch
`C:` a D-type flip-flop
`D:` a D-type latch
`E:` a T-type flip-flop

▷ **Q44.** The following diagram

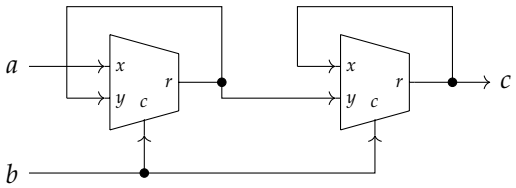


illustrates a preliminary NAND-based SR-latch design, in the sense it currently lacks an enable signal. If Q and Q' denote the current and next state respectively, which of the following excitation tables

			Current		*Next*	
	S	R	Q	$\neg Q$	Q'	$\neg Q'$
`A`	0	0	0	1	0	1
	0	0	1	0	1	0
	0	1	**?**	**?**	0	1
	1	0	**?**	**?**	1	0
	1	1	**?**	**?**	0	0
`B`	0	0	**?**	**?**	1	1
	0	1	**?**	**?**	1	0
	1	0	**?**	**?**	0	1
	1	1	0	1	0	1
	1	1	1	0	1	0
`C`	0	0	**?**	**?**	0	1
	1	1	**?**	**?**	1	0
`D`	0	**?**	**?**	**?**	0	1
	1	**?**	**?**	**?**	1	0
`E`	**?**	0	**?**	**?**	0	1
	?	1	**?**	**?**	1	0

correctly captures the behaviour of this circuit?

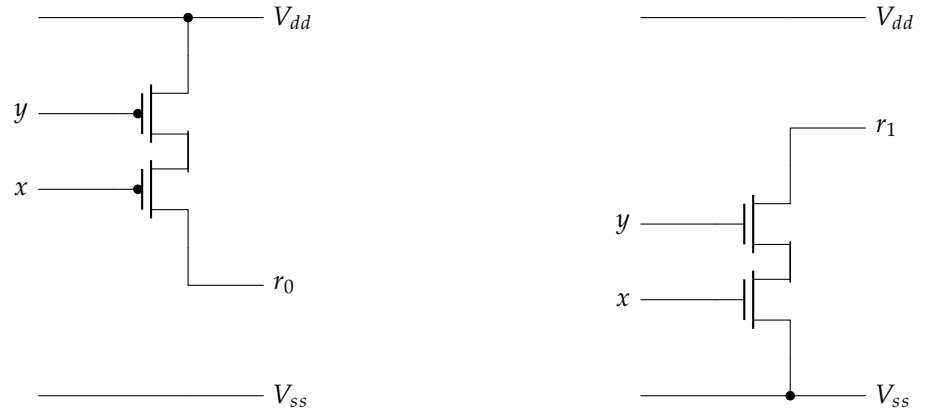
▷ **Q45.** Although perhaps unusual, the following diagram



illustrates a circuit with well defined behaviour. Based on analysis of this behaviour, which of the following components

`A:` a flip-flop
`B:` a latch
`C:` a RAM cell
`D:` a ROM cell
`E:` a clock multiplier

does the circuit implement?



(a) C_0 *(using P-type MOSFETs).* **(b)** C_1 *(using N-type MOSFETs).*

Figure 3: *MOSFET-based implementations of* C_0 *and* C_1.

▷ **Q46.** A m-output, 1-bit demultiplexer connects a 1-bit input x to one of m separate 1-bit outputs (say r_i for $0 \leq i < m$). The output is selected using an l-bit control signal c (or, equivalently, c is a collection of l separate 1-bit control signals). If $m = 5$, what is the minimum value of l required?

`A:` 0
`B:` 1
`C:` 2
`D:` 3
`E:` 4

▷ **Q47.** Figure 3 describes the implementation of two components denoted C_0 and C_1. Each component C_i produces one output r_i given two inputs x and y, and has been implemented using MOSFET transistors.

a The truth table below includes 5 possibilities for outputs r_0 and r_1 (stemming from instances of C_0 and C_1), given x and y. Recall that V_{ss} and V_{dd} are used to represent 0 and 1 respectively: which option is correct?

		A.		B.		C.		D.		E.	
x	y	r_0	r_1	r_0	r_1	r_0	r_1	r_0	r_1	r_0	r_1
0	0	1	0	0	0	1	0	**Z**	0	1	**Z**
0	1	1	1	0	0	0	0	**Z**	**Z**	**Z**	**Z**
1	0	1	1	0	0	0	0	**Z**	**Z**	**Z**	**Z**
1	1	0	1	1	0	0	0	1	**Z**	**Z**	0

b The vendor of these components claims they can be used to implement any Boolean function; their reasoning is based on the fact that a NAND gate can be implemented using instances of C_0 and C_1. Imagine you adhere to a design strategy where any given wire is driven by at most one non-**Z** value at any given time, and want to minimise the number of C_0 and C_1 instances used: how many of each do you need to implement a NAND gate?

A.		B.		C.		D.		E.	
C_0	C_1	C_0	C_1	C_0	C_1	C_0	C_1	C_0	C_1
1	1	5	3	3	5	3	3	5	5

▷ **Q48.** Moore's Law is an observation about the number of transistors which can be fabricated within some fixed unit of area: it observes that this number doubles roughly every two years. Which of the following properties of MOSFET-based transistors act as a constraint with respect to Moore's Law?

`A:` Feature size
`B:` Power consumption
`C:` Heat dissipation
`D:` All of the above

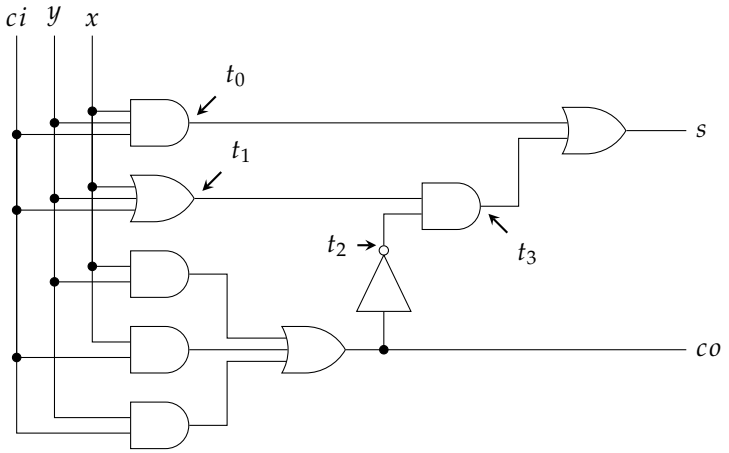


Figure 4: *An implementation of a full-adder cell.*

E: None of the above

▷ **Q49.** Figure 4 shows an implementation of a full-adder cell. It uses three 1-bit inputs denoted x, y, and ci (the carry-in), to compute two 1-bit outputs denoted s (the sum) and co (the carry-out); several other intermediate wires, namely t_0, t_1, t_2, and t_3, are labelled for reference. Let

$$(x, y, ci) \rightarrow (x', y', ci')$$

denote a change in said inputs: the LHS captures current values, whereas the RHS captures next (or new) values. For example,

$$(0, 0, 0) \rightarrow (1, 0, 0)$$

toggles x from 0 to 1, while both y and ci remain 0. Which of the following options will cause s to toggle in the shortest period of time (i.e., with the shortest delay)?

A: $(0, 0, 0) \rightarrow (0, 0, 1)$
B: $(0, 0, 1) \rightarrow (0, 1, 1)$
C: $(0, 1, 1) \rightarrow (0, 0, 1)$
D: $(1, 1, 1) \rightarrow (1, 1, 0)$
E: $(1, 0, 1) \rightarrow (0, 1, 1)$

▷ **Q50.** Figure 5 shows an implementation of a cyclic n-bit counter. While the counter is operational (i.e., while not reset, and given a clock signal), each r_i will transition between 0 and 1 at a different frequency. For the concrete case of $n = 4$, which does so at the lowest frequency?

A: r_4
B: r_3
C: r_2
D: r_1
E: r_0

▷ **Q51.** Consider a 16-bit register, constructed from CMOS-based D-type latches. Based on high-level reasoning about this component alone, if the initial value stored is $DEAD_{(16)}$ then overwriting it with which of the following

A: $BEEF_{(16)}$
B: $F00D_{(16)}$
C: $1234_{(16)}$
D: $FFFF_{(16)}$
E: $0000_{(16)}$

might you expect to consume the most power?

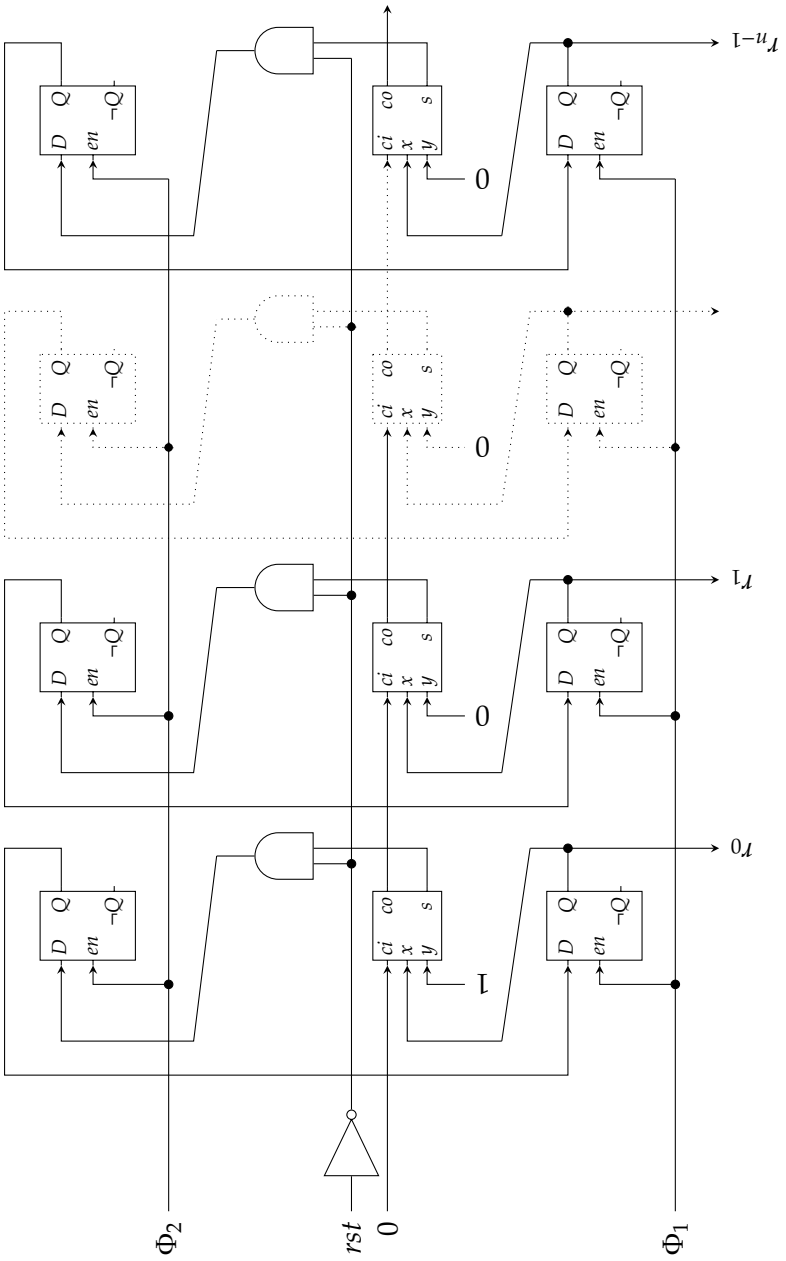


Figure 5: *An implementation of a cyclic n-bit counter.*

▷ **Q52.** You are tasked with implementing the Boolean function

$$r = f(x, y, z) = (\neg x \wedge \neg y \wedge \neg z) \vee (y \wedge \neg z) \vee (y \wedge z).$$

Each of the five options (i.e., columns) in the table below

	A	B	C	D	E
1.	×	✓	×	×	✓
2.	×	×	✓	×	✓
3.	×	×	×	✓	✓

states whether f can (a tick) or cannot (a cross) be implemented using a given set of components (i.e., row), namely

a an 8-input, 1-bit multiplexer,

b a 4-input, 1-bit multiplexer,

c a 2-input, 1-bit multiplexer, an OR gate, and a NOT gate,

plus the constant values 0 and 1. For example, option C states that f *can* be implemented by using component set 2 but not 1 or 3. Which option do you think is correct?

▷ **Q53.** Consider the fact that

i.e., that one can implement a NOT gate using one instance of a 2-input, 1-bit multiplexer component. Assuming you want to minimise the number of multiplexer instances, identify how many are required to implement the expression

$$(x \wedge y) \vee z.$$

A: 1
B: 2
C: 3
D: 6
E: 8

▷ **Q54.** Consider the combinatorial logic design as shown in Figure 6, which is described using N-type and P-type MOSFET transistors. Within the design, three inputs (i.e., x, y, and z) and one output (i.e., r) can be identified; note that several transistors (e.g., m_0) and intermediate signals (e.g., t_0) are annotated for reference. Which of the following Boolean expressions

A: $\neg x$
B: $\neg((x \mathbin{\overline{\vee}} y) \mathbin{\overline{\wedge}} z)$
C: $(\neg(x \mathbin{\overline{\vee}} y)) \mathbin{\overline{\wedge}} z$
D: $\neg(x \wedge y \wedge \neg z)$
E: $\neg(x \vee y \vee \neg z)$

does the design implement?

▷ **Q55.** Consider the sequential logic design as shown in Figure 7, which contains two D-type flip-flops. Within the design, one output (i.e., r) can be identified; note that several intermediate signals (e.g., t_0) are annotated for reference. If the clock signal clk has a frequency of 400MHz, what is the frequency of r?

A: 100MHz
B: 200MHz
C: 400MHz

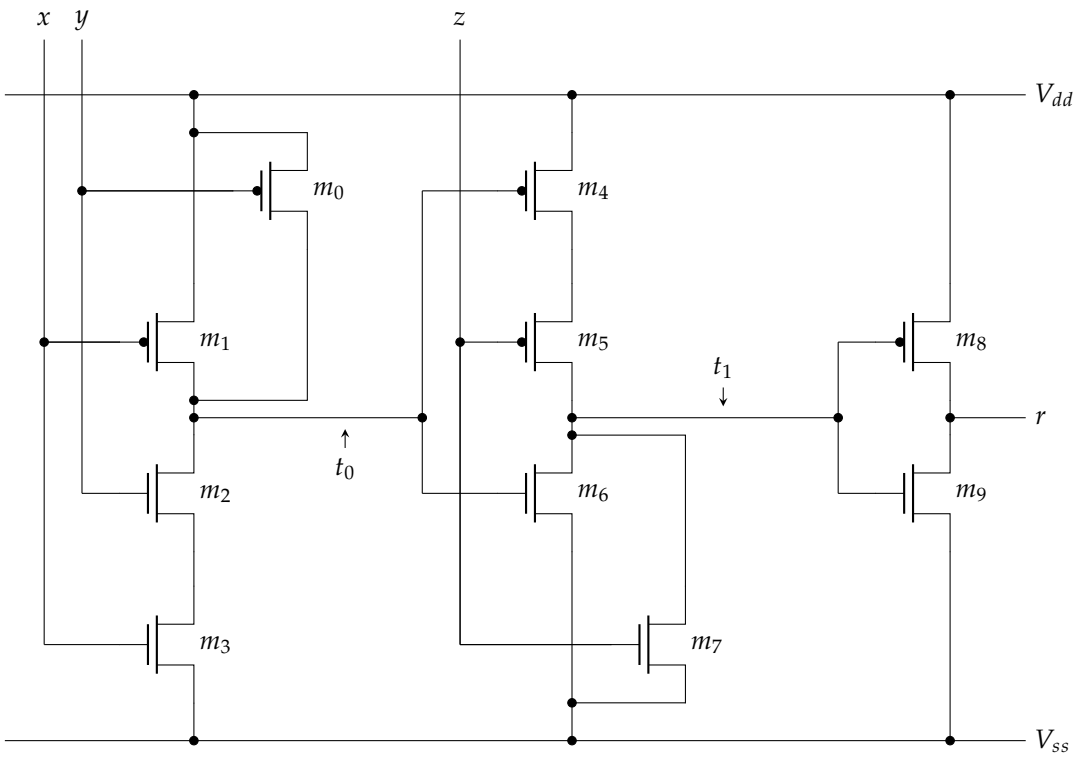


Figure 6: *A combinatorial logic design, described using N-type and P-type MOSFET transistors.*

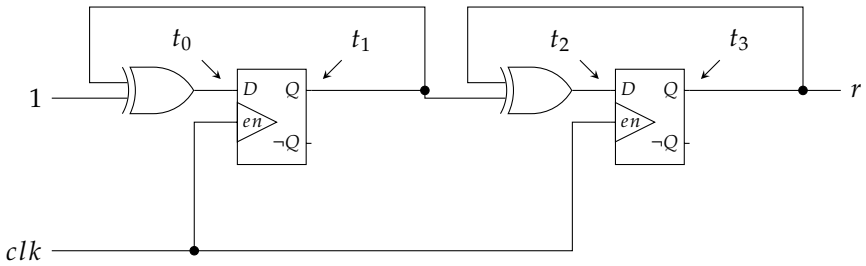
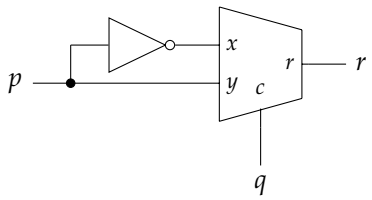


Figure 7: *A sequential logic design, containing two D-type flip-flops.*

`D:` 800MHz
`E:` 1600MHz

▷ **Q56.** Consider the following combinatorial logic design



which is described using a 2-input, 1-bit multiplexer. Within the design, two inputs (i.e., p and q) and one output (i.e., r) can be identified. Which of the following Boolean expressions

`A:` $r = \neg p$
`B:` $r = p \wedge q$
`C:` $r = \neg(p \wedge q)$
`D:` $r = p \oplus q$
`E:` $r = \neg(p \oplus q)$

correctly reflects the relationship between inputs and output?

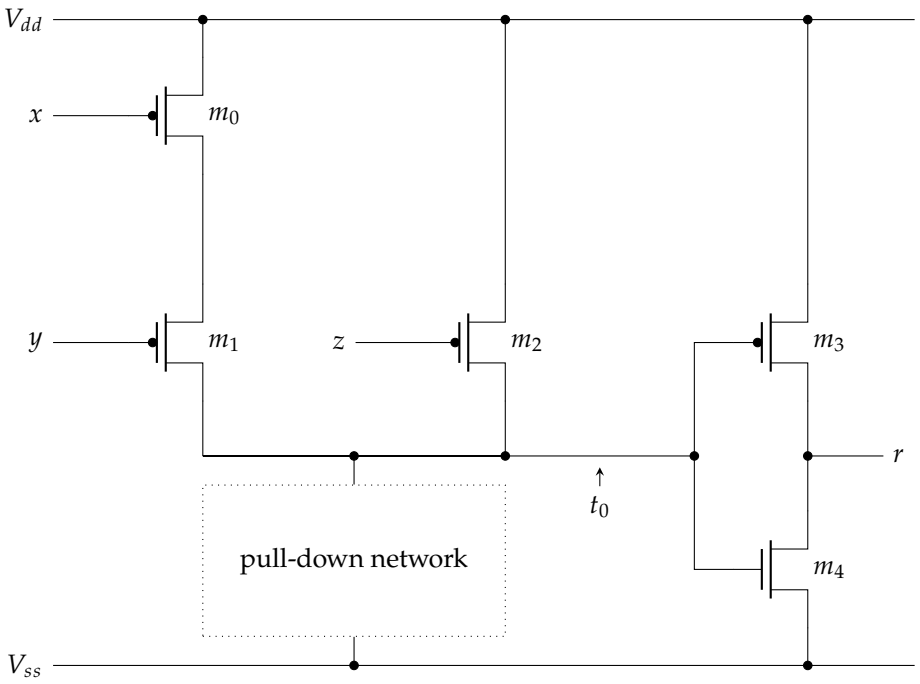
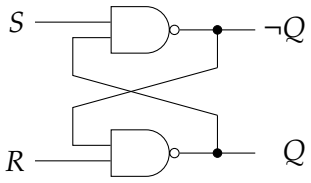


Figure 8: *A combinatorial logic design, described using N-type and P-type MOSFET transistors; note that the pull-down network is (partially) missing.*

▷ **Q57.** A NAND-based SR latch implementation can be realised as follows: The following



Imagine that the two NAND gates have a non-zero, but unequal gate delay associated with them, i.e., the top gate has the delay x whereas the bottom gate has the delay $x \pm \delta$ for some x and $\delta > 0$. If the current input $S = R = 0$ is changed instantaneously to $S = R = 1$, what will the outputs be?

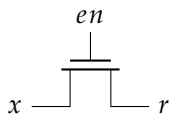
`A:` $Q = 1, \neg Q = 1$
`B:` either $Q = 0, \neg Q = 1$ or $Q = 1, \neg Q = 0$
`C:` either $Q = 1, \neg Q = 1$ or $Q = 0, \neg Q = 0$
`D:` $Q = 0, \neg Q = 0$
`E:` None of the above

▷ **Q58.** Consider the combinatorial logic design as shown in Figure 8, which is described using N-type and P-type MOSFET transistors. Within the design, three inputs (i.e., x, y, and z) and one output (i.e., r) can be identified; note that several transistors (e.g., m_0) and intermediate signals (e.g., t_0) are annotated for reference. Despite the fact that the pull-down network is (partially) missing, it is still possible to infer how the design works: which of the following Boolean expressions

`A:` $r = x \oplus y$
`B:` $r = (\neg x \wedge \neg y) \vee \neg z$
`C:` $r = (\neg x \vee \neg y) \wedge \neg z$
`D:` $r = (x \wedge y) \vee z$
`E:` $r = (x \vee y) \wedge z$

correctly reflects the relationship between inputs and output?

▷ **Q59.** Consider the following MOSFET transistor



If $x \in \{0, 1\}$ and $en \in \{0, 1\}$, how many different values can r potentially take?

A: 1
B: 2
C: 3
D: 4
E: 5

▷ **Q60.** Consider a combinatorial logic component defined by

$$r = f(x, y) = \begin{cases} 1 & x > y \\ 0 & \text{otherwise} \end{cases}$$

For how many combinations of the unsigned, 2-bit inputs x and y is the output $r = 1$?

A: 1
B: 2
C: 4
D: 6
E: 8

▷ **Q61.** Consider a micro-processor which is compatible with the ARMv7-A ISA. During execution of an instruction, the fetch stage of the fetch-decode-execute cycle computes PC + 4, which (potentially) forms the program counter in the next cycle. In an initial implementation of the micro-processor, PC + 4 is computed by using a general-purpose ripple-carry adder. Said adder is subsequently optimised, however, by capitalising on the special-purpose for of computation: ARMv7-A demands that PC is word-aligned, for example.

Assuming that logic gates for NOT, AND, OR, and XOR require 1, 2, 2, and 4 units of area respectively, the general-purpose solution requires

$$32 \cdot (2 \cdot \text{XOR} + 2 \cdot \text{AND} + 1 \cdot \text{OR}) = 32 \cdot (2 \cdot 4 + 2 \cdot 2 + 1 \cdot 2) = 448$$

units of area due to the use of 32 full-adder cells. If the optimisation aims to minimise area, which of the following options

A: 2.00
B: 2.31
C: 2.33
D: 2.52
E: 3.14

most accurately reflects the improvement factor offered by the special-purpose solution?

▷ **Q62.** Binary-Coded Decimal (BCD) is a representation for decimal integers, where each decimal digit in some x is represented independently by a 4-bit binary sequence in r. For example,

$$x = 123_{(10)} \mapsto \langle 0011_{(2)}, 0010_{(2)}, 0001_{(2)} \rangle = r.$$

Note that because each $0 \leq x_i < 10$ and $2^4 = 16 > 10$, some values of the associated BCD-encoded digit r_i are impossible.

Imagine you are asked to implement a 4-input Boolean function f using combinatorial logic, which will be used to process BCD-encoded digits. Select an option to complete blanks in the sentence "*a Karnaugh map cell which contains a _____ can be treated as either _____ or _____ in order to _____ the resulting term*" , so that it correctly describes how you might deal with an impossible BCD-encoded digit.

A: don't care, AND, OR, eliminate
B: duplicate, 1, 0, verify
C: unknown, 1, 0, simplify
D: don't care, 1, 0, simplify

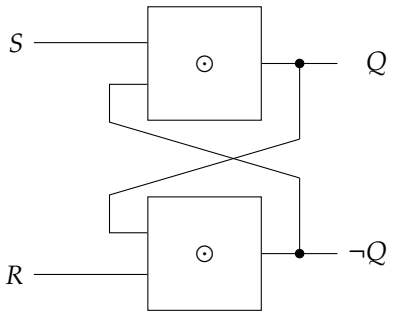


Figure 9: *An SR-latch, described in terms of placeholder components labelled ⊙.*

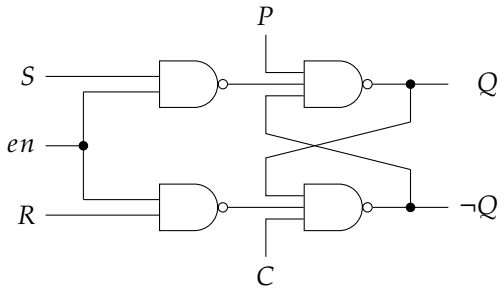


Figure 10: *An SR-latch variant, which includes additional inputs P, C, and en.*

`E:` unknown, 0, 1, optimise

▷ **Q63.** The block diagram in Figure 9, describes a sequential logic component, or, more specifically, an SR-type latch: it does so using two placeholder components labelled ⊙. If the associated excitation table is as follows

S	R	Q	$\neg Q$	Q'	$\neg Q'$
1	1	0	1	0	1
1	1	1	0	1	0
1	0	**?**	**?**	1	0
0	1	**?**	**?**	0	1
0	0	**?**	**?**	**?**	**?**

which of the following gate types

`A:` XOR
`B:` AND
`C:` OR
`D:` NAND
`E:` NOR

has been used to instantiate the placeholder components (i.e., replace each ⊙)?

▷ **Q64.** Figure 10, describes a sequential logic component, or, more specifically, a variant of the SR-latch: in addition to S and R, it also includes the inputs labelled P, C, and en.

a Which of the following options

	S and R	P and C
`A`	synchronous	synchronous
`B`	synchronous	asynchronous
`C`	asynchronous	synchronous
`D`	asynchronous	asynchronous

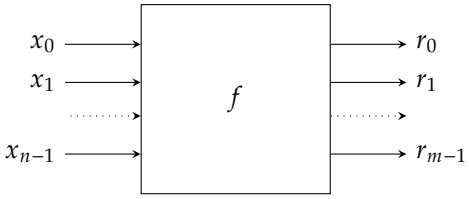
most accurately classifies the inputs?

b Which of the following options

	S and R	P and C
A	active low	active low
B	active low	active high
C	active high	active low
D	active high	active high

most accurately classifies the inputs?

▷ **Q65.** Abstractly, any functionality implemented using combinatorial logic can be viewed as an n-input, m-output Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}^m$, i.e., a block



with n 1-bit input and m 1-bit output wires.

a Which of the following

A: $n = 4, m = 1$
B: $n = 5, m = 1$
C: $n = 16, m = 4$
D: $n = 18, m = 1$
E: $n = 18, m = 4$

lists the correct values of n and m for a multiplexer that selects between four 4-bit values?

b Which of the following statements

A: For a half-adder $n = 2, m = 1$; for a full-adder $n = 4, m = 2$
B: For a half-adder $n = 2, m = 1$; for a full-adder $n = 3, m = 1$
C: For a half-adder $n = 2, m = 2$; for a full-adder $n = 3, m = 2$
D: For a half-adder $n = 3, m = 1$; for a full-adder $n = 2, m = 2$
E: For a half-adder $n = 3, m = 2$; for a full-adder $n = 2, m = 2$

is correct?

▷ **Q66.** Classify each of the following statements as either **true** or **false**, then explain why using at most a few sentences.

a Assuming $m > 1$, a combinatorial logic component with n inputs and m outputs can always be decomposed into m separate (potentially simpler) components.

b Application of the Karnaugh map technique to a truth table describing a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ will always yield the optimal implementation of said function.

▷ **Q67.** Write the simplest (i.e., with **fewest** operators) possible Boolean expression that implements the Boolean function

$$r = f(x, y, z)$$

described by

f			
x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	**?**
1	0	0	1
1	0	1	0
1	1	0	**?**
1	1	1	1

where **?** denotes don't care.

▷ **Q68.** Take the Boolean expression

$$\neg(x \vee y)$$

and draw a gate-level **circuit diagram** that computes an equivalent resulting using only 2-input NAND gates.

▷ **Q69.** Recall that an SR latch has two inputs S (or set) and R (or reset); if $S = R = 1$, the two outputs Q and $\neg Q$ are undefined. This issue can be resolved by using a reset-dominate latch: the alternative design has the same inputs and outputs, but resets the latch (i.e., has $Q = 0$ and $\neg Q = 1$) whenever $S = R = 1$.

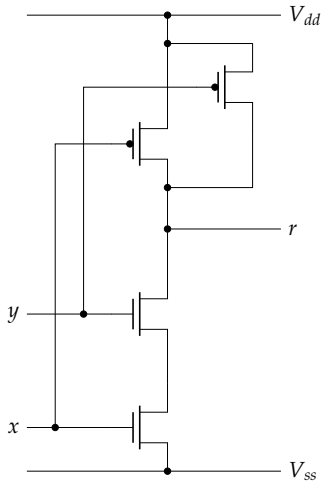
Using a gate-level **circuit diagram**, describe how a reset-dominate latch can be implemented using **only** NOR gates and at most **one** AND gate.

▷ **Q70.** The quality of the design for some hardware component is often judged by measuring efficiency, for example how quickly it can produce output on average. Name **two** other metrics that might be considered.

▷ **Q71.** a Describe how N-type and P-type MOSFET transistors are constructed using silicon and how they operate as switches.

b Draw a diagram to show how N-type and P-type MOSFET transistors can be used to implement a NAND gate. Show your design works by describing the transistor states for each input combination.

▷ **Q72.** The following diagram



details a 2-input NAND gate comprised of two P-MOSFET transistors (top) and two N-MOSFET transistors (bottom). Draw a similar diagram for a 3-input NAND gate.

▷ **Q73.** Moore's Law predicts the number of CMOS-based transistors we can manufacture within a fixed sized area will double roughly every two years; this is often *interpreted* as doubling computational efficiency over the same period. Briefly explain **two** limits which mean this trend cannot be sustained indefinitely.

▷ **Q74.** Given that **?** is the don't care state, consider the following truth table which describes a function p with

four inputs (a, b, c and d) and two outputs (e and f):

p					
a	b	c	d	e	f
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	**?**	**?**
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	**?**	**?**
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	**?**	**?**
1	1	0	0	**?**	**?**
1	1	0	1	**?**	**?**
1	1	1	0	**?**	**?**
1	1	1	1	**?**	**?**

a From the truth table above, write down the corresponding Sum of Products (SoP) equations for e and f.

b Simplify the two SoP equations so that they use the minimum number of logic gates possible. You can assume the two equations can share logic.

▷ **Q75.** Using a Karnaugh map, derive a Boolean expression for the function

$$r = f(x, y, z)$$

described by the truth table

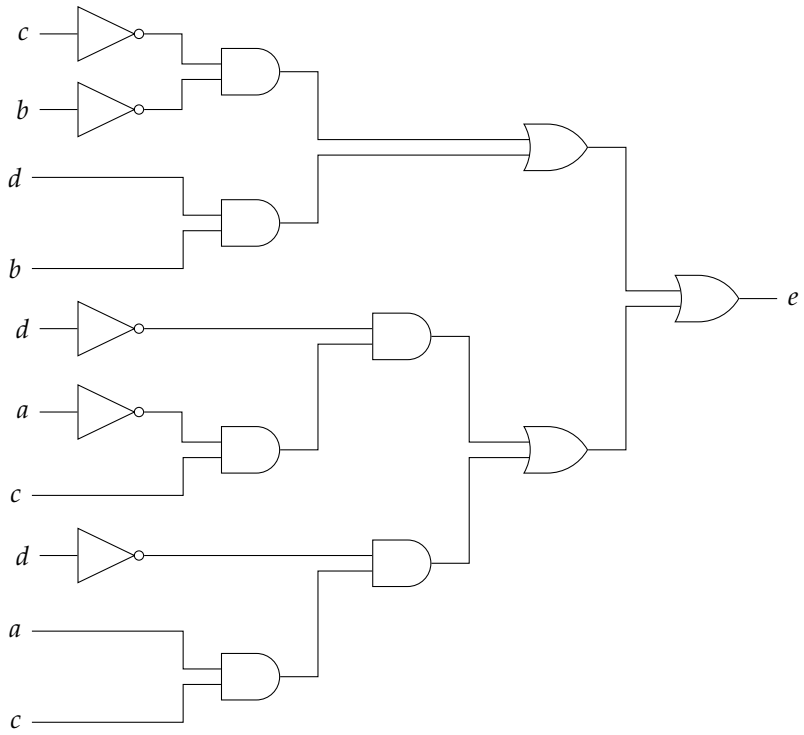
f			
x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	**?**

where **?** denotes don't care.

▷ **Q76.** NAND is a universal logic gate in the sense that the behaviour of NOT, AND and OR gates can be implemented using only NAND. Show how this is possible using a truth table to demonstrate your solution.

▷ **Q77.** Both NAND and NOR gates are described as universal because *any* other Boolean gate (i.e., AND, OR, NOT) can be constructed using them. Imagine your friend suggests a 4-input, 1-bit multiplexer (that selects between four 1-bit inputs using two 1-bit control signals to produce a 1-bit output) is also universal: state whether or not you believe them, and explain why.

▷ **Q78.** Consider the following circuit where the propagation delay of logic gates in the circuit are 10ns for NOT, 20ns for AND, 20ns for OR and 60ns for XOR:



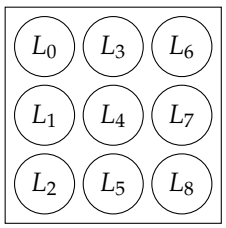
a Draw a Karnaugh map for this circuit and derive a Sum of Products (SoP) expression for the result.

b Describe advantages and disadvantages of your SoP expression and the dynamic behaviour it produces.

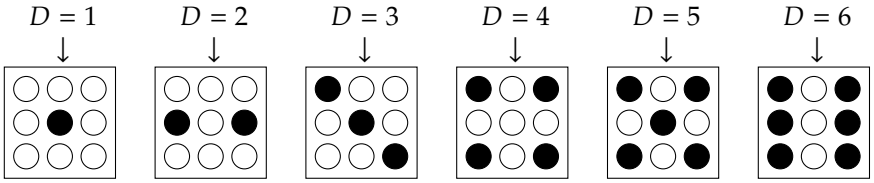
c If the circuit is used as combinatorial logic within a clocked system, what is the maximum clock speed of the system?

▷ **Q79.** A game uses nine LEDs to display the result of rolling a six-sided dice; the i-th LED, say L_i for $0 \leq i < 9$, is driven with 1 or 0 to turn it on or off respectively. A 3-bit register D represents the dice as an unsigned integer.

a The LEDs are arranged as follows,



and the required mapping between dice and LEDs, given a filled dot means an LED is on, is



Using Karnaugh maps as appropriate, write a simplified Boolean expression for **each** LED (i.e., for **each** L_i in terms of D).

b The 2-input XOR, AND, OR and NOT gates used to implement your expressions have propagation delays of 40, 20, 20 and 10 nanoseconds respectively. Calculate how many times per-second the dice can be rolled, i.e., D can be updated, if the LEDs are to provide the correct output.

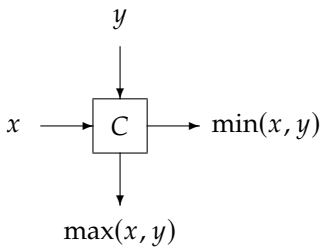
c The results of individual dice throws will be summed using a ripple-carry adder circuit, to give a total; each 3-bit output D will be added to and stored in an n-bit accumulator register A.

i Using a high-level **block diagram**, show how an n-bit ripple-carry adder circuit is constructed from full-adder cells.

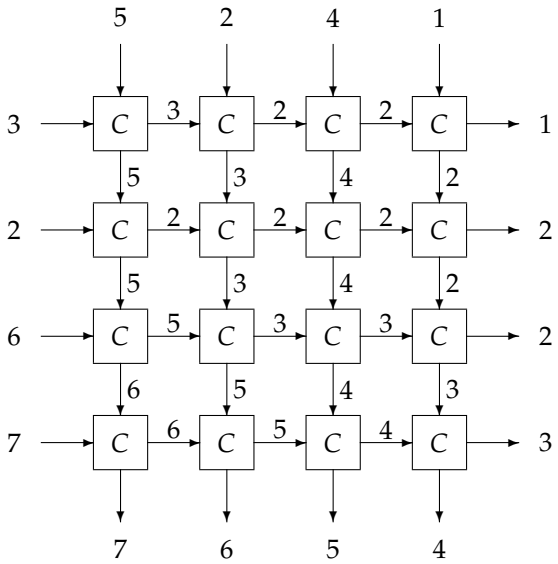
ii If $m = 8$ throws of the dice are to be summed, what value for n should be selected?

iii Imagine that instead of D, we want to add $2 \cdot D$ to A. Doubling D can be achieved by computing either $D + D$ or $D \ll 1$ (i.e., a left-shift of D by 1 bit). Carefully state which method is preferable, and why.

▷ **Q80.** Consider a simple component called C that compares two inputs x and y (both are unsigned 8-bit integers) in order to produce their maximum and minimum as two outputs:



Instances of C can be connected in a mesh to sort integers: the input is fed into the top and left-hand edges of the mesh, the sorted output appears on the bottom and right-hand edges. An example is given below:



a Using standard building blocks (e.g., adder, multiplexer etc.) rather than individual logic gates, draw a **block diagram** that implements the component C.

b Imagine that an $n \times n$ mesh of components is created. Based on your design for C and clearly stating any **assumptions** you need to make, write down an expression for the critical path of such a mesh.

c Algorithms for sorting integers can clearly be implemented on a general-purpose processor. Explain **two** advantages and **two** disadvantages of using such a processor versus using a mesh like that above.

▷ **Q81.** Imagine you are working for a company developing the "Pee", a portable games console. The user interface is a fancy controller that has

- three fire buttons represented by the 1-bit inputs F_0, F_1 and F_2, and
- a 8-direction D-pad represented by the 3-bit input D

and you are charged with designing some aspects of it.

a The fire button inputs are described as level triggered and active high; explain what this means (in comparison to the alternatives in each case).

b Some customers want an "autofire" feature that will automatically and repeatedly press the F_0 fire button for them. The autofire can operate in four modes, selected by a switch called M: off (where the fire button F_0 works as normal), slow, fast or very fast (where the fire button F_0 is turned on and off repeatedly at the selected speed). Stating any assumptions and showing your working where appropriate, design a circuit that implements such a feature.

c In an attempt to prevent counterfeiting, each controller can only be used with the console it was sold with. This protocol is used:

$$\begin{array}{ll} \mathcal{P} & \mathcal{C} \\ \hline c \xleftarrow{\$} \{0,1\}^3 & \\ & \xrightarrow{c} \\ & r = T(c) \\ & \xleftarrow{r} \\ r \stackrel{?}{=} T(c) & \\ \hline \end{array}$$

which, in words, means that

- the console generates a random 3-bit number c and sends it to the controller,
- the controller computes a 3-bit result $r = T(c)$ and sends it to the console,
- the console checks that r matches $T(c)$ and assumes the controller is valid if so.

i There is some debate as to whether the protocol should be synchronous or asynchronous; explain what your recommendation would be and why.

ii The function T is simply a look-up table. For example

$$T(x) = \begin{cases} 2 & \text{if } x = 0 \\ 6 & \text{if } x = 1 \\ 7 & \text{if } x = 2 \\ 1 & \text{if } x = 3 \\ 4 & \text{if } x = 4 \\ 0 & \text{if } x = 5 \\ 5 & \text{if } x = 6 \\ 3 & \text{if } x = 7 \end{cases}$$

Each pair of console and controller has such a T fixed inside them during the manufacturing process. Stating any assumptions and showing your working where appropriate, explain how **this** T might be implemented as a circuit.

▷ **Q82.** Imagine you have three Boolean values x, y, and z. Given access to **as many** AND and OR gates as you want but **only two** NOT gates, write a set of Boolean expressions to compute all three results $\neg x$, $\neg y$ and $\neg z$.

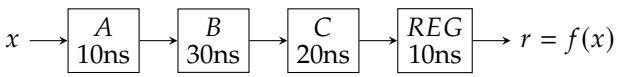
▷ **Q83.** SAT is the problem of finding an assignment to n Boolean variables which means a given Boolean expression is satisfied, i.e., evaluates to 1. For example, given $n = 3$ and the expression

$$(x \wedge y) \vee \neg z,$$

$x = 1$, $y = 1$, $z = 0$ is one assignment (amongst several) which solves the associated SAT problem.

The ability to solve SAT can be used to test whether or not two n-input, 1-output combinatorial circuits C_1 and C_2 are equivalent. Show how this is possible.

▷ **Q84.** Consider the following combinatorial circuit, which is the composition of four parts (labelled A, B, C and REG): each part is annotated with a name and an associated critical path. The circuit computes an output $r = f(x)$ from the corresponding input x.

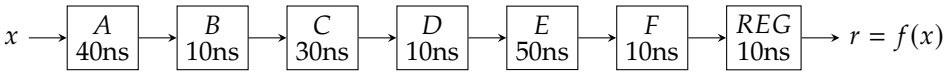


With respect to this circuit,

a first define the terms latency and throughput, then

b explain how and why you would expect use of pipelining to influence both metrics.

▷ **Q85.** The figure below shows a block of combinatorial logic built from seven parts; the name and latency of each part is displayed inside it. Note that the last part is a register which stores the result:



It is proposed to pipeline the block of logic using two stages such that there is a pipeline register in between parts D and E:

a Explain the terms latency and throughput in relation to the idea of pipelining.

b Calculate the overall latency and throughput of the initial circuit described above.

c Calculate the overall latency and throughput of the circuit after the proposed change.

d Calculate the number of extra pipeline registers required to maximise the circuit throughput; state this new throughput and the associated latency. Explain the advantages and disadvantages of this change.

Part III: Basics of digital logic: minimisation via Karnaugh maps

This is a (large) set of example Boolean minimisation questions: each asks you to transform some truth table describing an n-input Boolean function into a Boolean expression. Each solution includes

1. a reference implementation (produced by forming a SoP expression with a full term for *each* minterm, i.e., row where $r = 1$), and
2. a Karnaugh map annotated with sensible groups, and an optimised implementation based on those groups.

The goal is to focus on producing the latter, since the former is somewhat easier. Keep in mind and take care wrt. the following:

- There are 2^{2^n} Boolean functions with n inputs (or 3^{2^n} if you include don't-care as a valid output); whereas for small n a complete set of functions is included, but for large n there is only a random sub-set.
- No real effort is made to order the questions, and only minor effort to avoid duplicates. That said, there should be no trivial (in the sense $r = 1$ or $r = 0$ for all inputs, e.g., tautological) cases.
- The questions and solutions are generated automatically, meaning a small but real chance of bugs in the associated implementation!

▷ **Q86.**

y	z	r
0	0	1
0	1	1
1	0	0
1	1	1

▷ **Q87.**

y	z	r
0	0	1
0	1	0
1	0	1
1	1	1

▷ **Q88.**

y	z	r
0	0	1
0	1	1
1	0	0
1	1	0

▷ **Q89.**

y	z	r
0	0	0
0	1	1
1	0	1
1	1	1

▷ **Q90.**

y	z	r
0	0	0
0	1	1
1	0	0
1	1	1

▷ **Q91.**

y	z	r
0	0	0
0	1	0
1	0	1
1	1	0

▷ **Q92.**

y	z	r
0	0	0
0	1	1
1	0	1
1	1	0

▷ **Q93.**

y	z	r
0	0	1
0	1	0
1	0	1
1	1	0

▷ **Q94.**

y	z	r
0	0	0
0	1	0
1	0	0
1	1	1

▷ **Q95.**

y	z	r
0	0	1
0	1	0
1	0	0
1	1	0

▷ **Q96.**

y	z	r
0	0	0
0	1	?
1	0	?
1	1	1

▷ **Q97.**

y	z	r
0	0	0
0	1	1
1	0	?
1	1	1

▷ **Q98.**

y	z	r
0	0	1
0	1	1
1	0	1
1	1	0

▷ **Q99.**

y	z	r
0	0	0
0	1	0
1	0	1
1	1	?

▷ **Q100.**

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

▷ **Q101.**

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

▷ **Q102.**

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

▷ **Q103.**

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

▷ **Q104.**

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

▷ **Q105.**

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

▷ **Q106.**

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

▷ **Q107.**

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

▷ **Q108.**

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

▷ **Q109.**

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

▷ **Q110.**

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

▷ **Q111.**

x	y	z	r
0	0	0	?
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	?
1	0	1	1
1	1	0	1
1	1	1	0

▷ **Q112.**

x	y	z	r
0	0	0	0
0	0	1	?
0	1	0	0
0	1	1	?
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

▷ **Q113.**

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	?
1	0	0	1
1	0	1	1
1	1	0	?
1	1	1	1

▷ **Q114.**

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	0
1	1	1	?

▷ **Q115.**

x	y	z	r
0	0	0	?
0	0	1	0
0	1	0	?
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	?

▷ **Q116.**

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	?
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	?

▷ **Q117.**

x	y	z	r
0	0	0	?
0	0	1	?
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	?
1	1	1	0

▷ **Q118.**

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	?
1	1	1	?

▷ **Q119.**

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	?
0	1	1	?
1	0	0	1
1	0	1	?
1	1	0	0
1	1	1	0

▷ **Q120.**

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

▷ **Q121.**

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

▷ **Q122.**

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

▷ **Q123.**

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

▷ **Q124.**

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

▷ **Q125.**

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

▷ **Q126.**

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

▷ **Q127.**

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

▷ **Q128.**

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

▷ **Q129.**

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

▷ **Q130.**

w	x	y	z	r
0	0	0	0	?
0	0	0	1	?
0	0	1	0	0
0	0	1	1	1
0	1	0	0	?
0	1	0	1	0
0	1	1	0	?
0	1	1	1	?
1	0	0	0	?
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	?

▷ **Q131.**

w	x	y	z	r
0	0	0	0	1
0	0	0	1	?
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	?
1	0	0	0	1
1	0	0	1	?
1	0	1	0	0
1	0	1	1	?
1	1	0	0	?
1	1	0	1	?
1	1	1	0	?
1	1	1	1	?

▷ **Q132.**

w	x	y	z	r
0	0	0	0	?
0	0	0	1	0
0	0	1	0	0
0	0	1	1	?
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	?
1	1	1	0	0
1	1	1	1	1

▷ **Q133.**

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	?
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	?
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	?
1	1	1	0	?
1	1	1	1	0

▷ **Q134.**

w	x	y	z	r
0	0	0	0	?
0	0	0	1	0
0	0	1	0	?
0	0	1	1	0
0	1	0	0	0
0	1	0	1	?
0	1	1	0	?
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	?
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

▷ **Q135.**

w	x	y	z	r
0	0	0	0	?
0	0	0	1	0
0	0	1	0	1
0	0	1	1	?
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	?
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	?
1	1	1	0	?
1	1	1	1	?

▷ **Q136.**

w	x	y	z	r
0	0	0	0	0
0	0	0	1	?
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	?

▷ **Q137.**

w	x	y	z	r
0	0	0	0	?
0	0	0	1	0
0	0	1	0	?
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	?
0	1	1	1	?
1	0	0	0	1
1	0	0	1	0
1	0	1	0	?
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	?
1	1	1	1	?

▷ **Q138.**

w	x	y	z	r
0	0	0	0	?
0	0	0	1	1
0	0	1	0	?
0	0	1	1	0
0	1	0	0	?
0	1	0	1	1
0	1	1	0	?
0	1	1	1	1
1	0	0	0	?
1	0	0	1	0
1	0	1	0	0
1	0	1	1	?
1	1	0	0	1
1	1	0	1	?
1	1	1	0	?
1	1	1	1	1

▷ **Q139.**

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	?
0	1	0	0	?
0	1	0	1	?
0	1	1	0	?
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	?
1	1	0	1	?
1	1	1	0	1
1	1	1	1	0

Part IV: Basics of computer arithmetic

▷ **Q140.** Mike Rowchip was an engineer, working on an ALU design for a new processor: he had completed the design and implementation of most but not all modules, before he was, unfortunately, run over by a bus. You have been tasked with completing the missing modules.

Mike first prototyped each module in software, using C functions, which he then used as a specification for (and reference for testing) the associated hardware implementation. The prototype for one of the missing modules is

```
uint8_t f( uint8_t x, uint8_t y ) {
  uint8_t m = 1;

  while( x & m ) {
    x = x & ~m;
    m = m << 1;
  }

  return x | m;
}
```

However, Mike left no documentation beyond this. Thanks Mike. What do you think it does?

A: add `x` to `y`
B: compute the Hamming weight of `x`
C: left-rotate `x` by 1 bit
D: increment `x`
E: decrement `x`

▷ **Q141.** Consider the following conditional statement written in C

```
if( c ) {
  ...
}
```

wherein `c` is a placeholder for the condition expression; the statement body (i.e., the continuation dots) is executed iff. evaluating the condition expression yields a non-zero result. C represents signed integers using two's-complement: for an *unsigned*, 32-bit integer `x`, imagine we want to execute the statement body if either every bit of `x` is 0 *or* every bit of `x` is 1. Which of the following choices for the condition expression would achieve this?

A: `( x == 0 ) | ( x == -1 ) |`
B: `!x | !( x) |`
C: `( x + 1 ) < 2`
D: All of the above
E: None of the above

▷ **Q142.** Figure **??** captures the design of an n-bit ripple-carry adder, constructed using n full-adder instances connected by a carry chain denoted c. If $c_0 = ci$ (the carry-in) and $c_n = co$ (the carry-out), then c_i would more generally denote the carry into the i-th full-adder instance. If $n = 4$ and $ci = 0$, which of the following options

A	x	=	$0000_{(2)}$	y	=	$0000_{(2)}$
B	x	=	$1100_{(2)}$	y	=	$0001_{(2)}$
C	x	=	$0100_{(2)}$	y	=	$0100_{(2)}$
D	x	=	$1011_{(2)}$	y	=	$1001_{(2)}$
E	x	=	$0110_{(2)}$	y	=	$0101_{(2)}$

would produce $c_2 = 1$?

▷ **Q143.** Consider two integers x and y, whose sum $r = x + y$ is computed using a ripple-carry adder; x, y, and r are all 8-bit signed integers, represented using two's-complement. The associated flag

$$f = \begin{cases} 0 & \text{if } x + y \text{ did not overflow} \\ 1 & \text{if } x + y \text{ did overflow} \end{cases}$$

is used to signal whether an overflow occurred during computation of r. Which of the following

A: $f = r_8$
B: $f = (x_7 \wedge y_7 \wedge \neg r_7) \vee (\neg x_7 \wedge \neg y_7 \wedge r_7)$
C: $f = (x_7 \vee y_7 \vee \neg r_7) \wedge (\neg x_7 \vee \neg y_7 \vee r_7)$
D: $f = x_7 \wedge y_7 \wedge r_7$
E: $f = x_7 \oplus y_7 \oplus r_7$

is the correct Boolean expression for f?

▷ **Q144.** An n-bit ripple-carry adder has a critical path that can be described as $O(n)$ gate delays. Explain intuitively why this is the case, and name an alternative whose critical path is shorter.

▷ **Q145.** Give a single-line C expression to test if a non-zero integer x is an exact power-of-two; i.e., if $x = 2^n$ for some n then the expression should evaluate to a non-zero value, otherwise it evaluates to zero.

▷ **Q146.** Imagine you are writing a C program that includes a variable called `x`. If `x` has the type `char` and a current value of 127, what is the new value after

a decrementing (i.e., subtracting 1 from it), or

b incrementing (i.e., adding 1 to it)

the variable?

▷ **Q147.** Imagine x represents a two's-complement, signed integer using 4 bits; x_i denotes the i-th bit of x. Write a human-readable description (i.e., the meaning) of what the Boolean function

$$f(x) = \neg x_3 \wedge (x_2 \vee x_1 \vee x_0)$$

computes arithmetically.

▷ **Q148.** Given an n-bit input x, draw a **block diagram** of an efficient (i.e., with a **short** critical path) combinatorial circuit that can compute $r = 7 \cdot x$ (i.e., multiply x by the constant 7). Take care to label each component, and the size (in bits) of each input and output.

▷ **Q149.** Let x_i and y_i denote the i-th bit of two unsigned, 2-bit integers x and y (meaning that $0 \leq i < 2$). Design a (2×2)-bit combinatorial multiplier circuit that can compute the 4-bit product $r = x \cdot y$.

▷ **Q150.**

a Comparison operations for a given processor take two 16-bit operands and return zero if the comparison is false or non-zero if it is true. By constructing some of the comparisons using combinations of other operations, show that implementing all of $=, \neq, <, \leq, >$ and $\geq$ is wasteful. State the smallest set of comparisons that need dedicated hardware such that all the standard comparisons can be executed.

b The ALU in the same processor design does not include a multiply instruction. So that programmers can still multiply numbers, write an efficient C function to multiply two 16-bit inputs together and return the 16-bit lower half of the result. You can assume the inputs are always positive.

c The population count or Hamming weight of x, denoted by $\text{HW}(x)$ say, is the number of bits in the binary expansion of x that equal one. Some processors have a dedicated instruction to do this but the proposed one does not; write an efficient C function to compute the population count of 16-bit inputs.

▷ **Q151.** Imagine we want to compute the result of multiplying two n-bit numbers x and y together, i.e., $r = x \cdot y$, where n is even. One can adopt a divide-and-conquer approach to this computation by splitting x and y into two parts each of size $n/2$ bits

$$\begin{aligned} x &= x_1 \cdot 2^{n/2} + x_0 \\ y &= y_1 \cdot 2^{n/2} + y_0 \end{aligned}$$

and then computing the full result

$$r = r_2 \cdot 2^n + r_1 \cdot 2^{n/2} + r_0$$

via the parts

$$\begin{aligned} r_2 &= x_1 \cdot y_1 \\ r_1 &= x_1 \cdot y_0 + x_0 \cdot y_1 \\ r_0 &= x_0 \cdot y_0. \end{aligned}$$

The naive approach above uses four multiplications of $(n/2)$-bit values. The Karatsuba-Ofman method reduces this to three multiplications (and some extra low-cost operations); show how this is achieved.

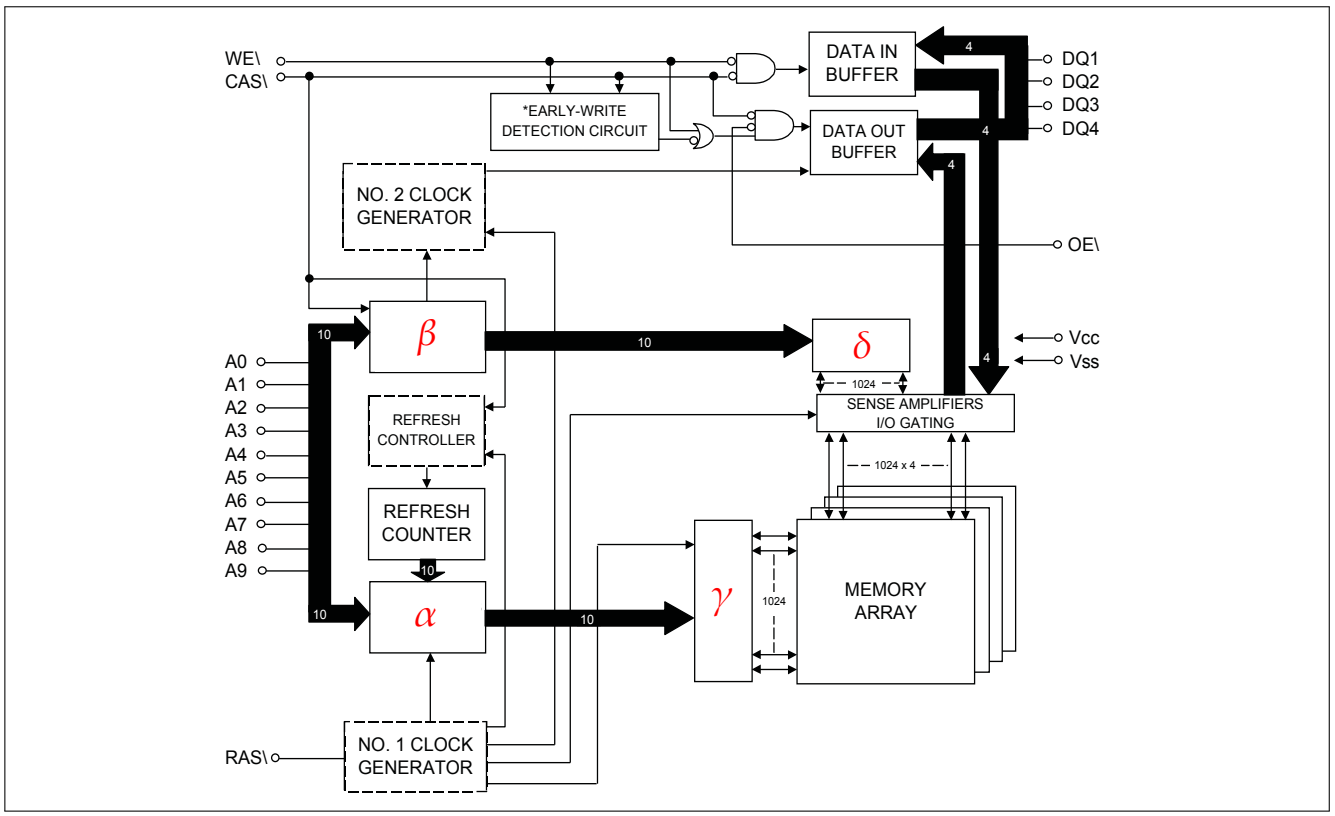


Figure 11: *A* 4Mbit *DRAM block diagram (source:* http://www.micross.com/pdf/MT4C4001J.pdf*).*

▷ **Q152.** Assume that unsigned integers are represented in 4 bits.

a What is the result of using a normal 4-bit adder circuit to compute the sum 10 + 12?

b A saturating (or clamped) adder is such that if an overflow occurs, i.e., the result does not fit into 4 bits, the highest possible result is returned instead. With a clamped 4-bit addition denoted by $\uplus$, we have that $10 \uplus 12 = 15$ for example. In general, for an n-bit clamped adder

$$x \uplus y = \begin{cases} x + y & \text{if } x + y < 2^n \\ 2^n - 1 & \text{otherwise} \end{cases}$$

Design a circuit that implements a 4-bit adder of this type.

▷ **Q153.** A software application needs 8-bit, unsigned modular multiplication, i.e., it needs to compute

$$x \cdot y \pmod{N}$$

which is the same as

$$t - (N \cdot \lfloor t/N \rfloor)$$

where $t = x \cdot y$. You have been asked to extend an existing ALU to support this operation. The high cost of a dedicated circuit for division rules out that option; using standard building blocks (e.g., adder, multiplexer) rather than individual logic gates, draw a **block diagram** of an alternative solution.

Part V: Basics of memory technology

▷ **Q154.** Figure 11 illustrates the design of a DRAM memory. The labels on four components in the block diagram

have been blanked-out, then replaced with the symbols α, β, γ, and δ: which of the following mappings

A	α	$\mapsto$	row address buffer
	β	$\mapsto$	row address decoder
	γ	$\mapsto$	column address buffer
	δ	$\mapsto$	column address decoder
B	α	$\mapsto$	row address buffer
	β	$\mapsto$	column address buffer
	γ	$\mapsto$	row address decoder
	δ	$\mapsto$	column address decoder
C	α	$\mapsto$	column address buffer
	β	$\mapsto$	row address buffer
	γ	$\mapsto$	column address decoder
	δ	$\mapsto$	row address decoder
D	α	$\mapsto$	row address decoder
	β	$\mapsto$	column address decoder
	γ	$\mapsto$	row address buffer
	δ	$\mapsto$	column address buffer
E	α	$\mapsto$	column address decoder
	β	$\mapsto$	row address decoder
	γ	$\mapsto$	column address buffer
	δ	$\mapsto$	row address buffer

do you think is correct?

▷ **Q155.** Identify **each** statement which is **correct**:

A: SRAM-based memories typically have a lower access latency than DRAM-based alternatives
B: SRAM-based memories typically have a lower density than DRAM-based alternatives
C: a memory that uses big-endian byte ordering will have a higher access latency than one using a little-endian order
D: a Harvard-style organisation means both instructions and data are stored in the same memory

▷ **Q156.** Consider a DRAM-based memory device with a capacity of 65536 addressable bytes. Of the following options

A: 8 address pins, 65536 cells
B: 16 address pins, 65536 cells
C: 8 address pins, 524288 cells
D: 16 address pins, 524288 cells
E: none of the above

which offers the most likely description of said device?

▷ **Q157.** Consider an SRAM-based memory device, which has a 4-bit data bus and 12-bit address bus. If your goal is to construct a 32KiB memory and *only* have such devices available, how many will you need?

A: 1
B: 2
C: 4
D: 8
E: 16

▷ **Q158.** Imagine you are using an 1kB, byte-addressable memory within some larger system. In doing so, you make a mistake which means the 4-th address wire A_4 is not correctly connected: it therefore has the fixed value $A_4 = 0$. Which of the following options

A: 1
B: 4
C: 256
D: 512

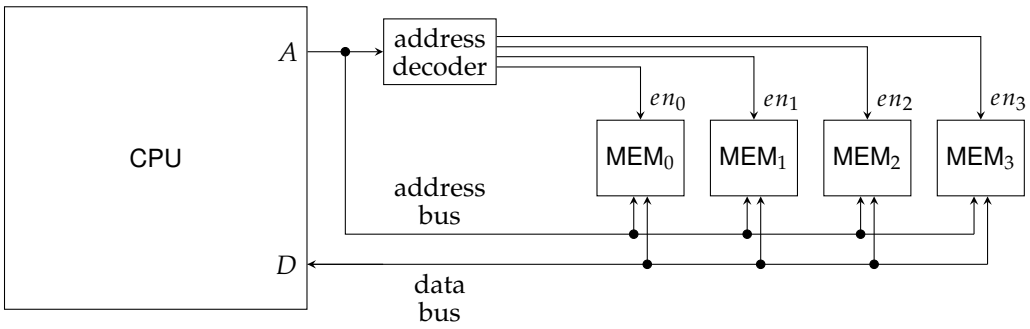


Figure 12: *A diagrammatic description of an 8-bit micro-processor and associated memory system.*

`E:` 1024

reflects the number of addresses now accessible if the memory is

a an SRAM, or

b a DRAM.

▷ **Q159.** Consider an 8-bit micro-processor, connected to a memory system via an 18-bit address bus: let A denote said bus, such that A_i for $0 \leq i < 18$ is the i-th bit. The memory system is comprised of 4 separate memory devices (either RAMs or ROMs) denoted MEM_0, MEM_1, MEM_2, and MEM_3. An address decoder maps addresses to memory devices by controlling a set of associated enable (or chip select) signals, i.e., en_0, en_1, en_2, and en_3. Figure 12 offers a diagrammatic version of the same description, noting various extraneous control signals are omitted for clarity.

If the enable signals are

$$\begin{aligned} en_0 &= \neg A_{17} \wedge \neg A_{16} \wedge \neg A_{15} \\ en_1 &= \neg A_{17} \wedge \neg A_{16} \wedge A_{15} \wedge \neg A_{14} \\ en_2 &= \neg A_{17} \wedge A_{16} \\ en_3 &= A_{17} \wedge A_{16} \wedge A_{15} \wedge A_{14} \wedge A_{13} \wedge A_{12} \wedge A_{11} \wedge A_{10} \wedge A_9 \wedge A_8 \wedge A_7 \wedge A_6 \wedge A_5 \end{aligned}$$

which memory device is address $A = 48350$ mapped to?

`A:` MEM_0
`B:` MEM_1
`C:` MEM_2
`D:` MEM_3

▷ **Q160.** Classify each of the following statements as either **true** or **false**, then explain why using at most a few sentences.

a One SRAM memory device has 1024 addressable 8-bit elements, whereas another has 1024 addressable 32-bit elements: the latter will have a higher access latency than the former.

▷ **Q161.** Draw a transistor-level **circuit diagram** describing a 6T SRAM memory cell.

▷ **Q162.** Consider a 1Mbit SRAM memory device (i.e., housing a total of 10^6 SRAM memory cells, each holding a 1-bit value), and a DRAM-based alternative with the same capacity: you are tasked with deciding which device to use within some larger system. After reading the data sheets, it seems that

a the DRAM-based device might be harder to integrate into the system, and

b the SRAM-based device should have a lower access latency.

Briefly explain why **each** statement is accurate.

▷ **Q163.** At a high level, a DRAM memory device could be described as an array (or matrix) of 1-bit cells with an interface including a data pin, address pins and control pins (e.g., chip select, output and write enable, row and column strobes). Carefully explain the purpose of

a row and column buffers, and

b row and column decoders

which represent components in such a device.

Part VI: Digital logic design using Verilog

▷ **Q164.** Write the following as Verilog declarations:

a An 8-bit little-endian wire vector called `a`.

b A 5-bit big-endian wire vector called `b`.

c A 32-bit register called `c`.

d A signed 16-bit register called `d`.

e A memory of 1024 elements, each 8 bits in size, called `e`.

f A generate variable called `f`.

▷ **Q165.** Given the declarations:

```
3 : 0 ] a;
3 : 0 ] b;
1 : 0 ] c;
3 : 0 ] d;
        e;

a = 4'b1101;
b = 4'b01XX;
```

what are the values resulting from the following assignments:

a `assign c = a[ 1 : 0 ];`

b `assign c = a[ 3 : 2 ];`

c `assign d = a & b;`

d `assign d = a ^ b;`

e `assign d = { a[3:2], a[1:0] };`

f `assign d = { a[1:0], a[3:2] };`

g `assign c = { 2{ b[ 1 ] } };`

h `assign c = { 2{ b[ 2 ] } };`

i `assign e = &a;`

j `assign e = ^a;`

▷ **Q166.** a Consider the following Verilog processes for appropriately defined 1-bit wires `a`, `b`, `x`, `y`, `p` and `q`:

```
always @ ( posedge p ) begin
  x <= a;
  y <= b;
end

always @ ( posedge q ) begin
  x <= b;
  y <= a;
end
```

Given that `p` and `q` are independent and may change at any time, write down **one** potential problem with this design and outline **one** potential solution.

b Consider the following Verilog process, for appropriately defined 1-bit wire `clk` and 2-bit wire vector `state`, which implements a state machine with three states:

```
always @ ( posedge clk ) begin
  case( state )
    0 : begin do_0; state = 1; end
    1 : begin do_1; state = 2; end
    2 : begin do_2; state = 0; end
  endcase
end
```

If this process constitutes the entirety of the design, write down **one** potential problem with it and outline **one** potential solution.

▷ **Q167.** A Decimal Digit Detector (DDD) is a device that accepts a 1-bit input at each positive clock edge, and waits until four such bits have been received. At this point, it sets a 1-bit output signal to true if the 4-bit accumulated input (interpreted in little-endian form) is a valid Binary Coded Decimal (BCD) digit and false if it is not; it then repeats the process for the next set of four input bits.

Design a Verilog module to model the DDD device; your design should incorporate a reset signal that is able to initialise the DDD.

▷ **Q168.** A comparator C is a function which takes two unsigned n-bit integers x and y as input and produces $\max(x, y)$ and $\min(x, y)$ as outputs. One can think of C as sorting the 2-element sequence (x, y) into the resulting sequence $(\min(x, y), \max(x, y))$. Design a Verilog module to model a single comparator for n-bit numbers.

▷ **Q169.** An n-bit shift register Q is a register (say n D-type flip-flops) whose content is right-shifted on each positive of a shared clock edge. This means if Q_i refers to the i-th bit of Q,

- Q_0 is discarded (i.e., shifted out of the register),
- every Q_i for $0 \leq i < n - 1$ is set to Q_{i+1}, and
- Q_{n-1} is replaced by some new value (i.e., shifted in to the register).

A Linear Feedback Shift Register (LFSR) is a type of pseudo-random number generator based on this component: at each positive clock edge

- Q_0 is used as a 1-bit pseudo-random output from the LFSR, and
- Q_{n-1} is replaced by a bit dictated by other bits in Q, which are XOR'ed together according to a tap sequence.

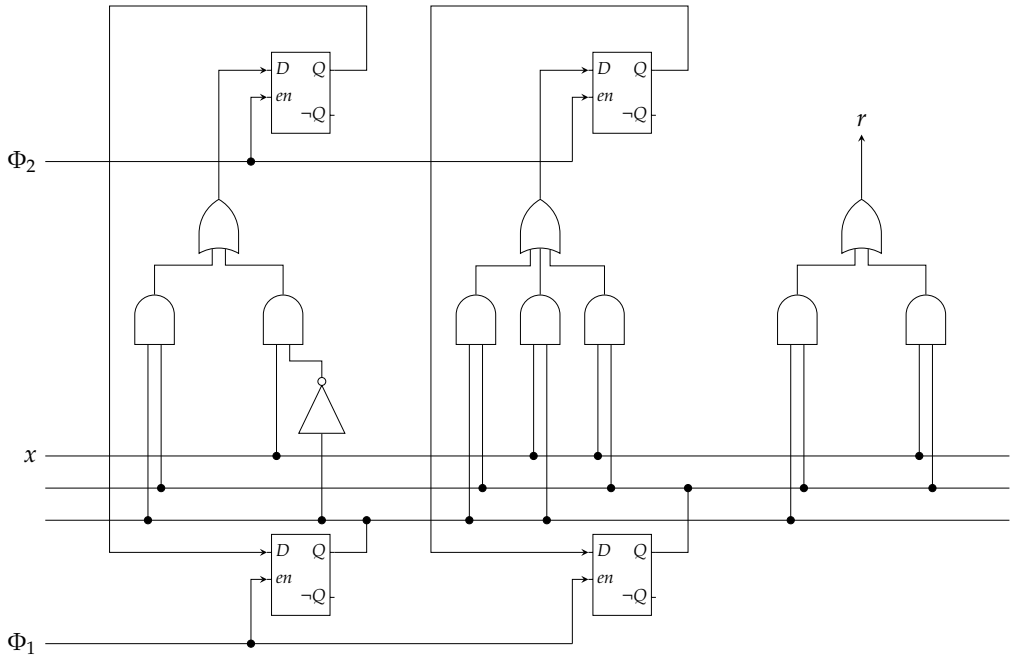
For example, if the tap sequence is $T = \langle 0, 3, 4, 5, 7 \rangle$ then the new bit that replaces Q_{n-1} is

$$t = \bigoplus_{t \in T} Q_t = Q_0 \oplus Q_3 \oplus Q_4 \oplus Q_5 \oplus Q_7.$$

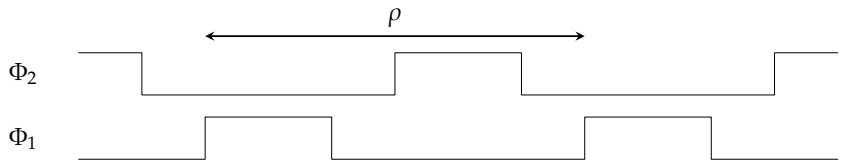
Design a Verilog module to model an 8-bit LFSR with the tap sequence above; your design should incorporate a reset signal which initialises the LFSR with a seed value given as input.

Part VII: Computational machines: Finite State Machines (FSMs)

▷ **Q170.** Consider a Finite State Machine (FSM) whose concrete implementation is as follows:



Notice that the implementation is based on use of four D-type latches, and a 2-phase clock supplied via Φ_1 and Φ_2; one additional input x plus one output r are also evident. To function correctly, a clock generator ensures Φ_1 and Φ_2 are driven as follows:



a From the following list

`A:` Φ_1 and Φ_2 are digital signals
`B:` Φ_1 and Φ_2 are non-overlapping
`C:` Φ_1 and Φ_2 are gated
`D:` Φ_1 and Φ_2 are unskewed
`E:` Φ_1 and Φ_2 each have a duty cycle of 33%

identify **each** property the clock generator *must* guarantee is true for the implementation to function correctly.

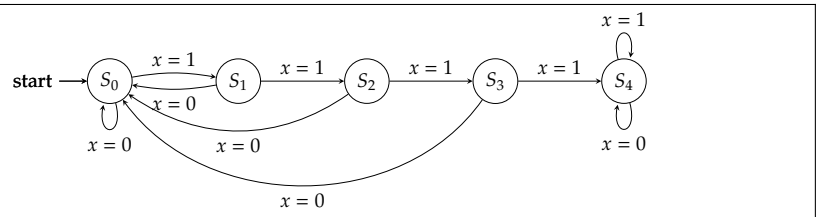
b Consider the two D-type latches at the bottom of the diagram, which form a 2-bit register. Imagine the value stored in this register is expressed as a 2-bit integer: when the implementation is initially powered-on, is this value equal to

`A:` $00_{(2)}$
`B:` $01_{(2)}$
`C:` $10_{(2)}$
`D:` $11_{(2)}$
`E:` any of the above

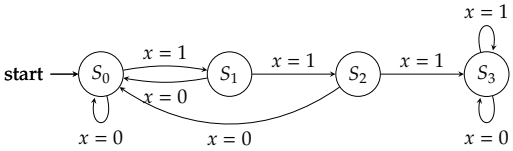
c Any FSM specification will include a transition function, often denoted δ, which can be described in either

tabular or diagrammatic form. Of the following options

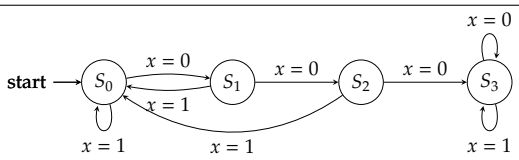
A



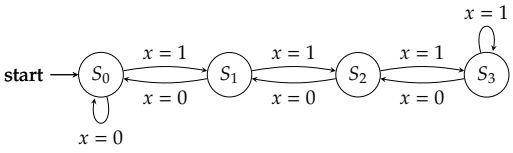
B



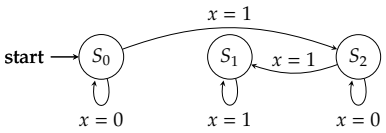
C



D



E



which captures the transition function of this FSM?

d Which of the following FSM types, namely

`A:` Mealy
`B:` Moore

does this implementation represent?

e In the 2-phase clock waveform above, ρ illustrates the clock period: recall this is inversely proportional to the clock frequency. Imagine the gate delay for NOT, AND, and 2- and 3-input OR gates are 10ns 20ns 20ns, 30ns respectively, and the critical path associated for a D-type latch is 60ns. Which of the following best matches the maximum possible clock frequency of this implementation?

`A:` 1.0kHz
`B:` 5.9MHz
`C:` 9.0MHz
`D:` 9.5MHz
`E:` 1.0GHz

f Which of the following best describes the purpose of this FSM?

`A:` set $r = 1$ iff. the current value of x is different from the previous value of x,
`B:` act as a modulo 4 counter that is incremented by the value of x, and set $r = 1$ the current counter value is zero,
`C:` compute the Hamming weight of a sequence fed as input bit-by-bit via x, and set $r = 1$ once this is equal to 3

D: count the number of consecutive times $x = 1$, and set $r = 1$ once this is equal to 3
E: inspect the sequence fed as input bit-by-bit via x, and set $r = 1$ iff. this sequence, when interpreted as an unsigned integer, is odd

▷ **Q171.** Figure 13 and Figure 14, describe an FSM implementation and an associated waveform. When read left-to-right, the waveform captures how values of Φ_1 and Φ_2 (a 2-phase clock), and rst (a reset signal) change over time; the other input s maintains the value $A6_{(16)}$ throughout. Note that the waveform is annotated with some instances and periods in time (e.g., ρ, and each t_i).

a What is the value of r at time t_0?

A: 0
B: 1
C: undefined

b What is the value of r at time t_1?

A: 0
B: 1
C: undefined

c What is the value of r at time t_2?

A: 0
B: 1
C: undefined

d Consider the following NAND-based implementations

D-type latch	↦	Figure 15
2-input XOR gate	↦	Figure 16
2-input, 1-bit multiplexer	↦	Figure 17

relating to components used within Figure **??**. The waveform is annotated with ρ, which illustrates the clock period. If a 2-input NAND gate imposes a gate delay of $T_{\text{nand}} = 10\text{ns}$, which value most closely reflects the maximum possible clock frequency?

A: 1.0MHz
B: 1.2GHz
C: 3.8MHz
D: 5.9MHz
E: 6.6MHz

▷ **Q172.** Consider the design as shown in Figure 18, which implements a simple Finite State Machine (FSM) using D-type latches and a 2-phase clock. Note that the r output reflects whether the FSM is in an accepting state, the rst input resets the FSM into the start state, and the X_i input drives transitions between states: the idea is that the i-th element of a sequence

$$X = \langle X_0, X_1, \ldots, X_{n-1} \rangle$$

is provided as input, via X_i, in the i-th step. Assuming the entirety of X is consumed, which of the following

A: $r = X_0 \oplus X_1 \oplus \cdots \oplus X_{n-1}$
B: $r = X_0 \mathbin{\overline{\wedge}} X_1 \mathbin{\overline{\wedge}} \cdots \mathbin{\overline{\wedge}} X_{n-1}$
C: $r = X_0 \wedge X_1 \wedge \cdots \wedge X_{n-1}$
D: $r = X_0 \mathbin{\overline{\vee}} X_1 \mathbin{\overline{\vee}} \cdots \mathbin{\overline{\vee}} X_{n-1}$
E: $r = X_0 \vee X_1 \vee \cdots \vee X_{n-1}$

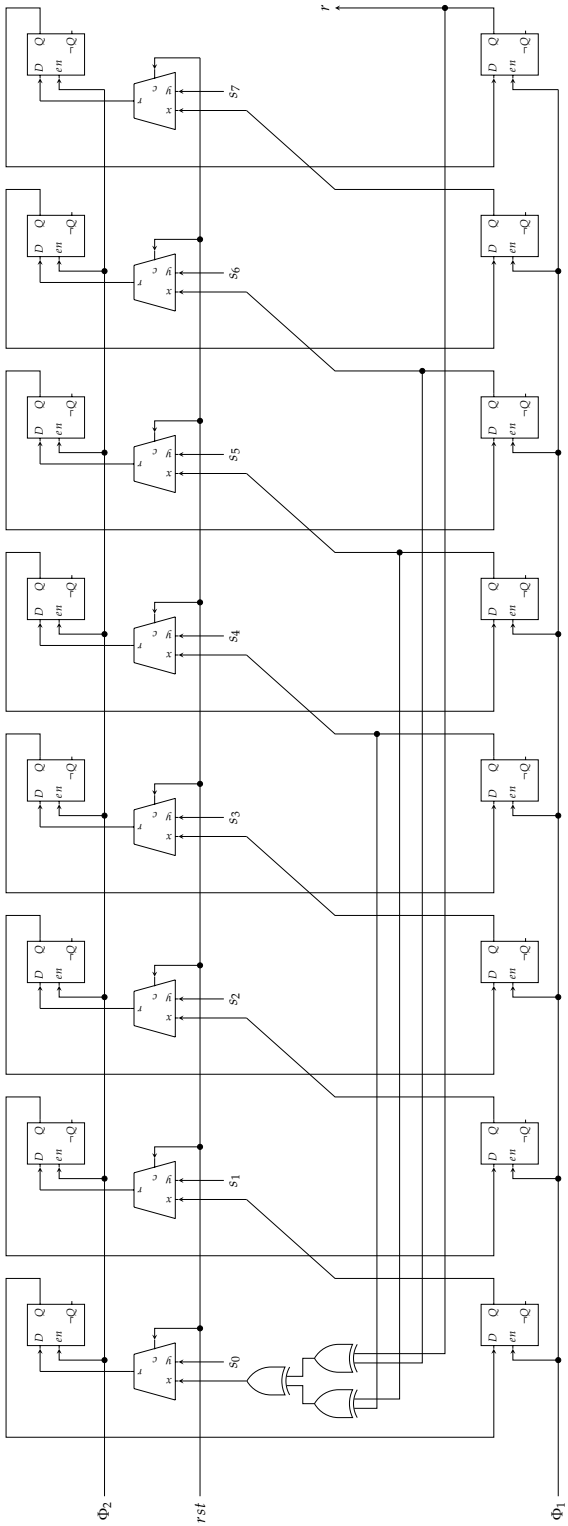


Figure 13: *An FSM implementation, which has* 4 *inputs (*1*-bit* Φ_1*,* Φ_2 *and* `rst` *on the left-hand side;* 8*-bit* `s` *spread within the design) and* 1 *output (*1*-bit* `r` *on the right-hand side).*

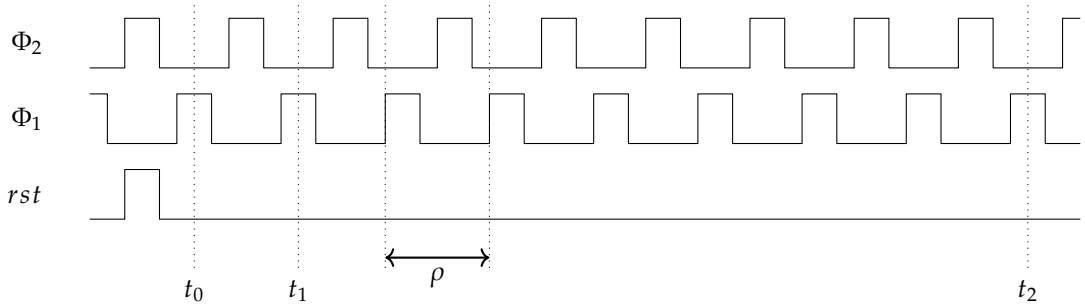


Figure 14: *A waveform describing behaviour of Φ_1, Φ_2, and rst within Figure 13.*

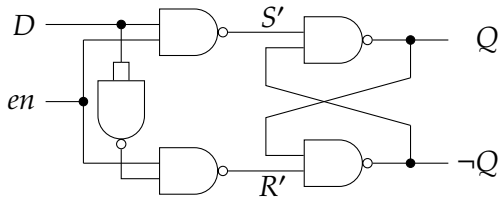


Figure 15: *A NAND-based implementation of a D-type latch.*

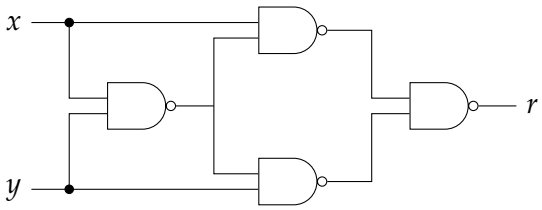


Figure 16: *A NAND-based implementation of a 2-input XOR gate.*

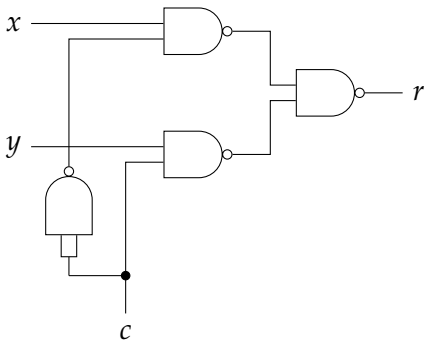


Figure 17: *A NAND-based implementation of a 2-input, 1-bit multipliexer.*

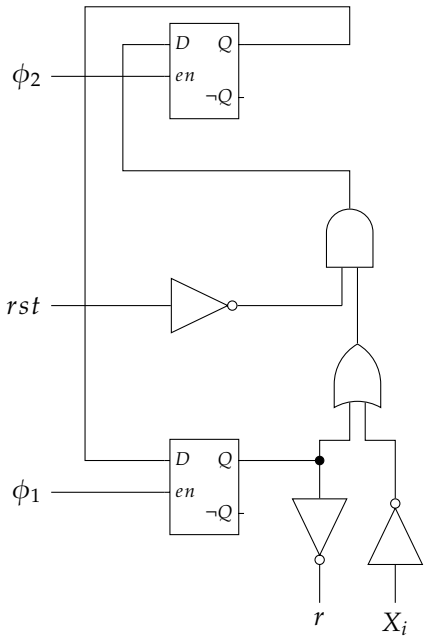


Figure 18: *Implementation of a simple FSM, using D-type latches and a 2-phase clock.*

best describes the output from, or functionality of the FSM?

▷ **Q173.** Consider the design as shown in Figure 19, which implements a simple Finite State Machine (FSM) using D-type latches and a 2-phase clock; note that it includes one output labelled r, and one input labelled x. Which of the following options

`A:` 2
`B:` 3
`C:` 5
`D:` 6
`E:` 9

reflects

a the number of gates involved in the output function implementation?

b the number of gates involved in the transition function implementation?

▷ **Q174.** Consider a FSM defined abstractly by Figure 20 and concretely by Figure 21, meaning the latter is an implementation of the former. Note that there is one output labelled r and one input labelled x, and that D-type latches are used to store the current (resp. next) state denoted Q (resp. Q'); doing so implies used of a 2-phase clock. The AND gate labelled $\odot$ has 2 inputs, which are currently disconnected. Which of the following

`A:` Q_1 and Q_0
`B:` $\neg x$ and Q_0
`C:` $\neg x$ and Q_1
`D:` x and Q_0
`E:` x and Q_1

should those inputs be connected to if the FSM is to function correctly?

▷ **Q175.** Classify each of the following statements as either **true** or **false**, then explain why using at most a few sentences.

a The clock frequency used to control a FSM must be limited by the critical path of the transition or output function; otherwise, the FSM will function incorrectly.

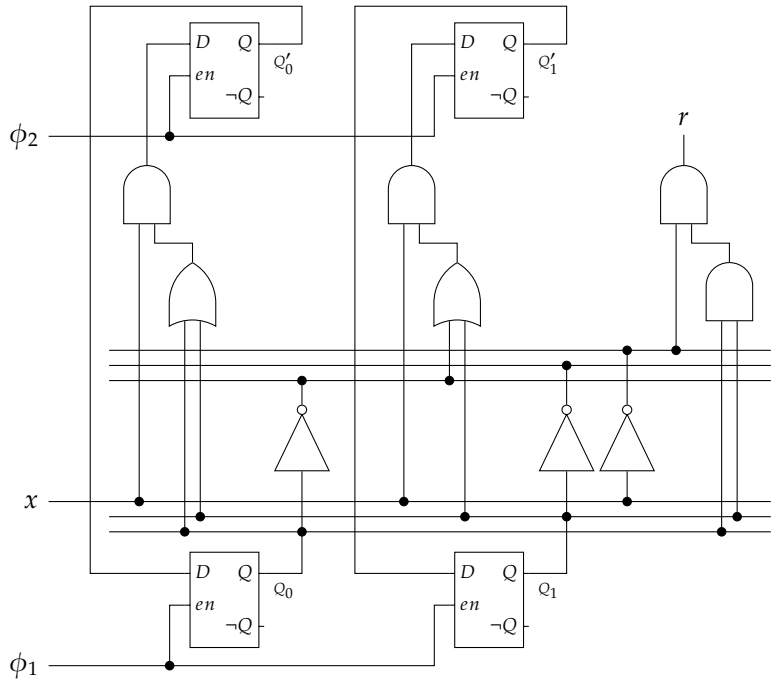


Figure 19: *Implementation of a simple FSM, using D-type latches and a 2-phase clock.*

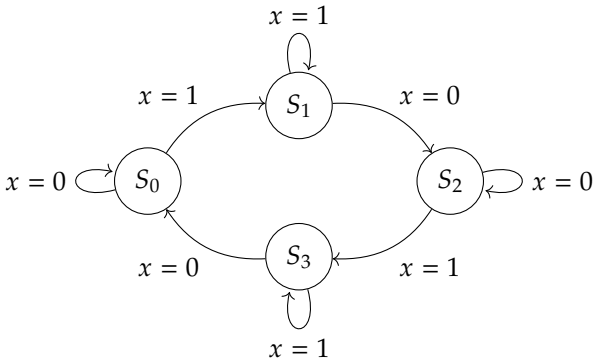


Figure 20: *The abstract design of an example FSM.*

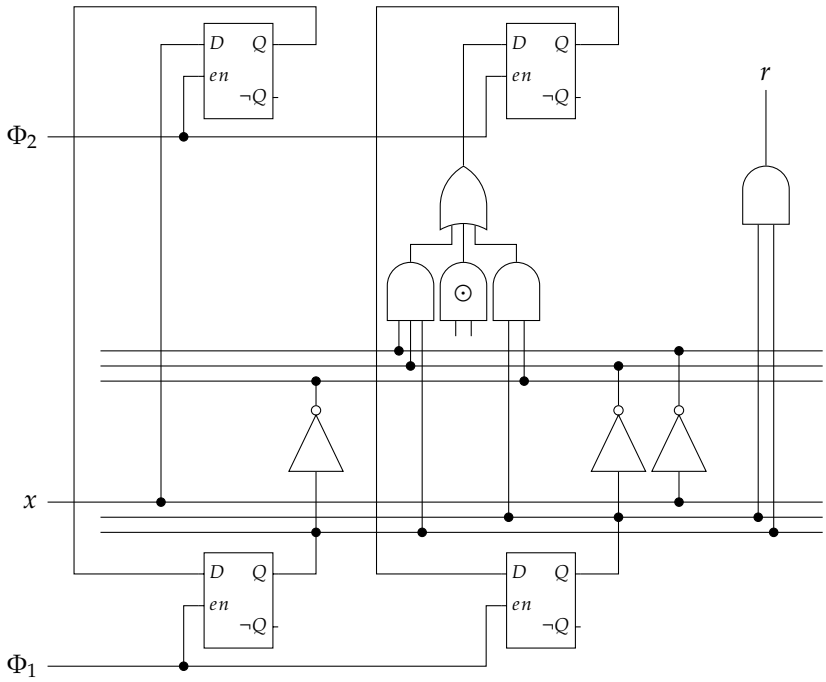


Figure 21: *The concrete implementation of an example FSM.*

▷ **Q176.** The **parity function** f accepts an n-bit sequence X as input, and yields $f(X) = 1$ iff. X has an odd number of elements equal to 1. If $f(X) = 1$ (resp. $f(X) = 0$), we say the parity of X is odd (resp. even). Using a combinatorial circuit, one can compute this as

$$f(X) = X_0 \oplus X_1 \oplus \cdots \oplus X_{n-1}$$

since XOR can be thought of as addition modulo two. However, how could we design a Finite State Machine (FSM) to compute $f(X)$ when supplied with X one element at a time? Explain step-by-step how you would solve this challenge: start with a high-level design for any FSM then fill in detail required for *this* FSM. Are there any features or requirements you can add to this basic description so the FSM is deemed "better" somehow?

▷ **Q177.** Imagine you are asked to to build a simple DNA matching hardware circuit as part of a research project. The circuit will be given DNA strings which are sequences of tokens that represent chemical building blocks. The goal is to search a large input sequence of DNA tokens for a small sequence indicative of some feature.

The circuit will receive one token per clock cycle as input; the possible tokens are adenine (A), cytosine (C), guanine (G) and thymine (T). The circuit should, given the input sequence, set an output flag to 1 when the matching sequence ACT is found somewhere in the input or 0 otherwise. You can assume the inputs are infinitely long, i.e., the circuit should just keep searching forever and set the flag when the match is a success.

a Design a circuit to perform the required task, show all your working and explain any design decisions you make.

b Now imagine you are asked to build two new matching circuits which should detect the sequences CAG and TTT respectively. It is proposed that instead of having three separate circuits, they are combined into a single circuit that matches the input sequence against one matching sequence selected with an additional input. Describe **one** advantage and **one** disadvantage you can think of for the two implementation options.

▷ **Q178.** A revolutionary, ecologically sound washing machine is under development by your company. When turned on, the machine starts in the *idle* state awaiting input. The washing cycle consists of the three stages: *fill* (when it fills with water), *wash* (when the wash occurs), *spin* (when spin dying occurs); the machine then returns to *idle* when it is finished. Two buttons control the machine: pressing B_0 starts the washing cycle, pressing B_1 cancels the washing cycle at any stage and returns the machine to *idle*; if both buttons are pressed at the same time, the machine continues as normal as if neither were pressed.

a You are asked to design a circuit to control the washing machine. Draw a diagram illustrating states the washing machine can be in, and valid transitions between them.

b Translate your diagram from above into a corresponding, tabular description of the transition function.

c Using an appropriate technique, derive Boolean expressions which allow computation of the transition function; note that because the washing machine is ecologically sound, minimising the overall gate count is important.

▷ **Q179.** Recall that an n-bit Gray code is a cyclic, 2^n-element sequence S where each i-th element S_i is itself an n-element binary sequence, and the Hamming distance between adjacent elements is one, i.e.,

$$\mathrm{HD}(S_i, S_{i-1 \pmod{2^n}}) = \mathrm{HD}(S_i, S_{i+1 \pmod{2^n}}) = 1.$$

a Using an expression (rather than words), define

i $\mathrm{HW}(X)$, the Hamming weight of a binary sequence X, and

ii $\mathrm{HD}(X, Y)$, the Hamming distance between binary sequences X and Y.

b Consider a D-type flip-flop, capable of storing a 1-bit value, realised using CMOS-based transistors arranged into logic gates. Using a gate-level **circuit diagram**, describe the design of such a component (clearly explaining the purpose of each part).

c Imagine successive elements of a 3-bit Gray code sequence are stored, one after another, in a register realised using flip-flops of the type described above. The fact only one bit changes each time the register is updated could be viewed as advantageous: explain why.

d Using a **block diagram**, draw a generic Finite State Machine (FSMs) framework, including for example δ, ω and any input and output; clearly explain the purpose of each component in the framework.

e Using the framework outlined above, design a concrete FSM which has

- two 1-bit inputs rst and clk, and
- one 3-bit output r.

and whose behaviour is as follows: at each positive edge of the clock signal clk, if $rst = 0$ then r should be updated with the next element of a 3-bit Gray code, otherwise r should be reset to the first element.

Note that your answer should provide enough detail to fully specify each component in the framework (e.g., Boolean expressions for δ).

▷ **Q180.** An electronic security system, designed to prevent unauthorised use of a door, is attached to a mains electricity supply. The system has the following components:

- Three buttons, say B_i for $0 \leq i < 3$, whose value is initially 0; when pressed, a button remains pressed and the value changes to 1.
- A door handle modelled by

$$H = \begin{cases} 1 & \text{when the handle is turned} \\ 0 & \text{when the handle is unturned} \end{cases}$$

- A lock mechanism modelled by

$$L = \begin{cases} 1 & \text{when the door is locked} \\ 0 & \text{when the door is unlocked} \end{cases}$$

If the door handle is turned after the order of button presses matches a 3-element password sequence P, the door should be unlocked; if there is a mismatch, it should remain locked. The mechanism is reset (and all buttons released) whenever the handle is turned (whether or not the door is unlocked). If $P = \langle B_1, B_0, B_2 \rangle$, then for example

- B_1 then B_0 then B_2 is pressed, then the handle is turned, the door is unlocked, i.e., L is set to 0, and the mechanism is reset,
- B_0 then B_1 then B_2 is pressed, then the handle is turned, the door remains locked, i.e., L is set to 1, and the mechanism is reset,

- B_1 then B_0 is pressed, then the handle is turned, the door remains locked, i.e., L is set to 1, and the mechanism is reset.

a Using a **block diagram**, draw a generic Finite State Machine (FSMs) framework, including for example the transision and output functions (i.e., δ and ω) and any input and output; clearly explain the purpose of each component in the framework.

b Imagine the password is fixed to $P = \langle B_2, B_0, B_1 \rangle$. Using the framework outlined above, design a concrete FSM which can be used to control the security system as required.

Note that your answer should provide enough detail to fully specify each component in the framework (e.g., Boolean expressions for the transision function).

c After inspecting your design, someone claims they can avoid the need for a clock signal: explain how this is possible.

d The same person suggests an alternative approach whereby P is not fixed, but rather stored in an SRAM memory device. Although this approach could be more useful, explain **one** reason it could be viewed as disadvantageous.

e Before being sold, each physical system needs to be tested to ensure it functions as advertised. Explain a suitable testing strategy for your design, **and** any alterations required to facilitate it.

▷ **Q181.** Imagine you are John Connor in the film Terminator II: your aim is to design a device that guesses ATM (or cash machine) Personal Identification Numbers (PINs) using brute-force search. The ATM uses 4-digit decimal PINs, examples being 1234 and 9876. The device stores a current PIN denoted P: it performs each guess in sequence by first checking whether P is correct, then incrementing P ready for the next step. The process concludes when P is deemed correct.

a Two potential representations for the PIN are suggested:

a decimal representation in which the PIN is stored as a sequence of four unsigned integers, i.e., $P = \langle P_0, P_1, P_2, P_3 \rangle$, with each $0 \leq P_i < 10$, or

a binary representation in which the PIN is stored as a single unsigned integer, i.e., P, with $0 \leq P < 10000$.

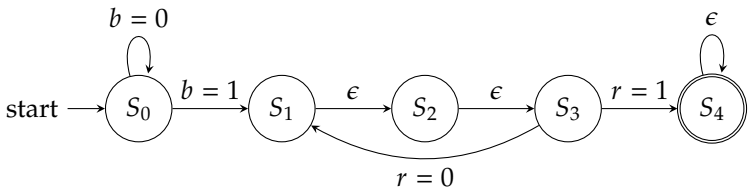
State **one** advantage of **each** option, and explain which you think is more appropriate.

b A combinatorial component within the device should take the current PIN P as input, and produce two outputs:

- the guess sent to the ATM, i.e., $G = \langle G_0, G_1, G_2, G_3 \rangle$, where each $0 \leq G_i < 10$ is the i-th decimal digit of the current PIN, and
- the incremented PIN P' ready for the next guess.

Produce a design for this component; include a **block diagram** and enough detail to fully specify how a gate-level implementation could be performed.

c The device is controlled by a simple Finite State Machine (FSM) which can be described diagrammatically:



In a more explanatory form, the idea is as follows:

- The device starts in state S_0, in which P is initialised; once the start button b is pressed, it moves into state S_1.
- In state S_1, P is driven as input into combinatorial component and the device moves into state S_2.
- In state S_2, G is sent to the ATM and P' is latched to form the new value of P; the device moves into state S_3.

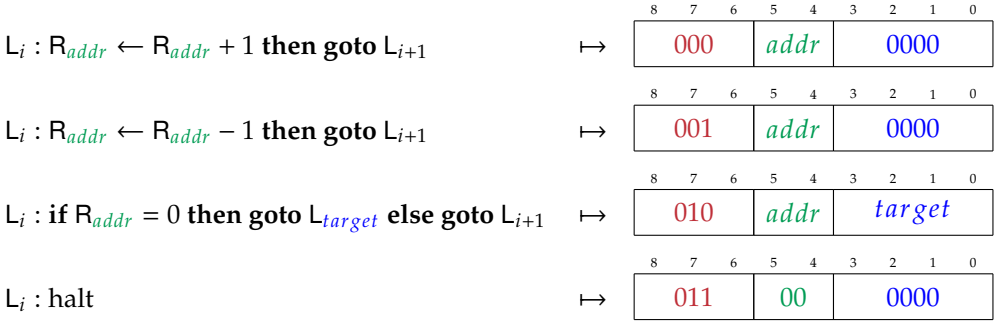


Figure 22: *The instruction set for an example 4-register counter machine.*

- In state S_3 the device checks the ATM response r. If $r = 1$ then G was the correct guess and the device moves into state S_4 where it halts (i.e., remains in S_4); otherwise, the device moves into state S_1 and the process repeats.

Focusing on the diagram above only, produce a design for the FSM; include a **block diagram**, and enough detail to fully specify how a gate-level implementation could be performed.

Part VIII: Computational machines: Register Machines (RMs)

▷ **Q182.** A given counter machine has $r = 4$ registers, and supports an instruction set detailed in Figure 22. Consider two configurations of this counter machine:

a the program, held in memory as machine code, is fixed to

$$\begin{aligned} \mathsf{MEM} = \langle\ & 0A3_{(16)},\ 060_{(16)},\ 080_{(16)},\ 097_{(16)}, \\ & 050_{(16)},\ 020_{(16)},\ 083_{(16)},\ 0C0_{(16)}\ \rangle \end{aligned}$$

and the initial configuration is

$$C_0 = (l = 0, v_0 = 0, v_1 = 2, v_2 = 1, v_3 = 0),$$

b the program, held in memory as machine code, is fixed to

$$\begin{aligned} \mathsf{MEM} = \langle\ & 0B3_{(16)},\ 070_{(16)},\ 080_{(16)},\ 097_{(16)}, \\ & 030_{(16)},\ 050_{(16)},\ 083_{(16)},\ 0AB_{(16)}, \\ & 030_{(16)},\ 060_{(16)},\ 087_{(16)},\ 0C0_{(16)}\ \rangle \end{aligned}$$

and the initial configuration is

$$C_0 = (l = 0, v_0 = 0, v_1 = 3, v_2 = 2, v_3 = 1).$$

For each configuration, a) produce a trace of execution, then b) decide which of the following options

`A:` Compare the values in R_1 and R_2, setting R_3 to reflect the result
`B:` Add the values in R_1 and R_2, setting R_3 to reflect the result
`C:` Swap the values in R_1 and R_2
`D:` Copy the value in R_1 into R_2, retaining the value in R_1
`E:` Copy the value in R_1 into R_2, clearing the value in R_1

is the best description of what the associated program does.

▷ **Q183.** Figure 22 describes the instruction set of an example 4-register counter machine. Consider some i-th encoded, machine code instruction $0A5_{(16)}$ expressed in hexadecimal. Which of the following

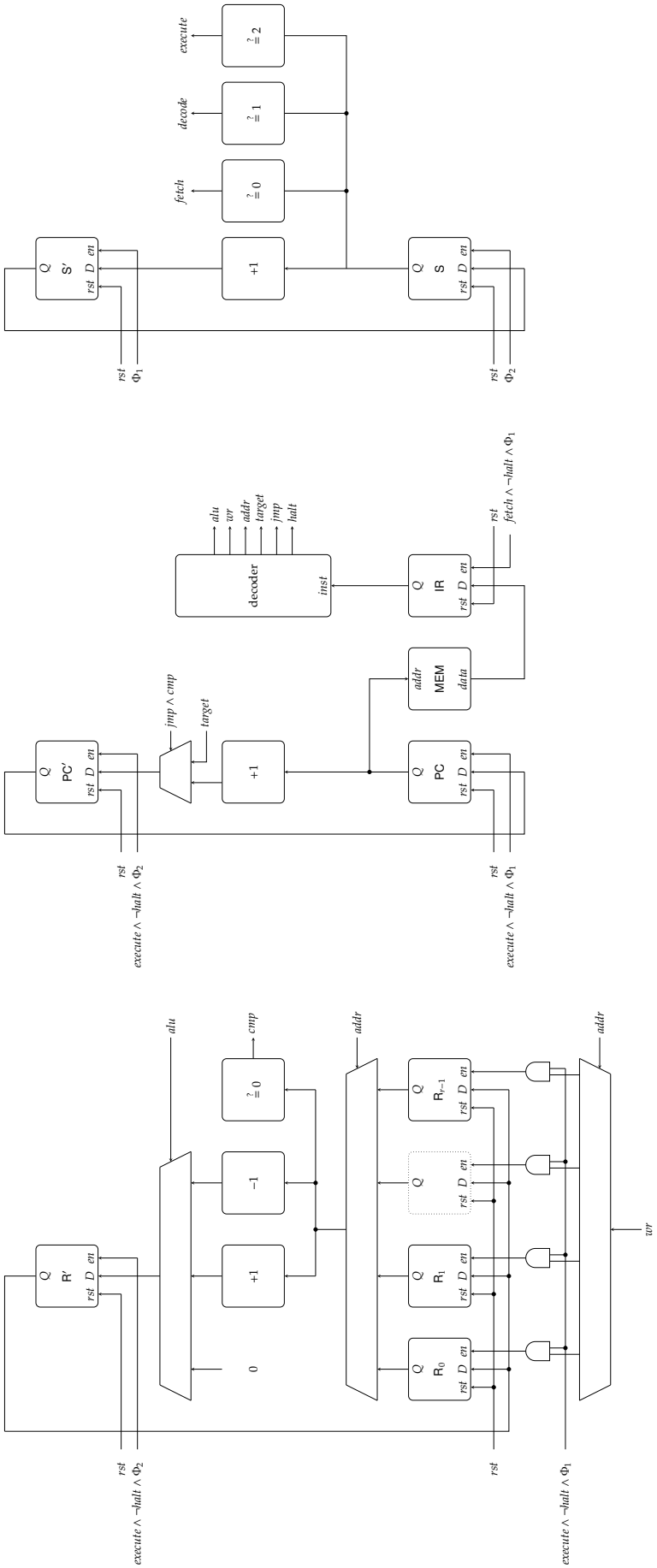


Figure 23: *The high-level data- and control-path for an example* 4*-register counter machine.*

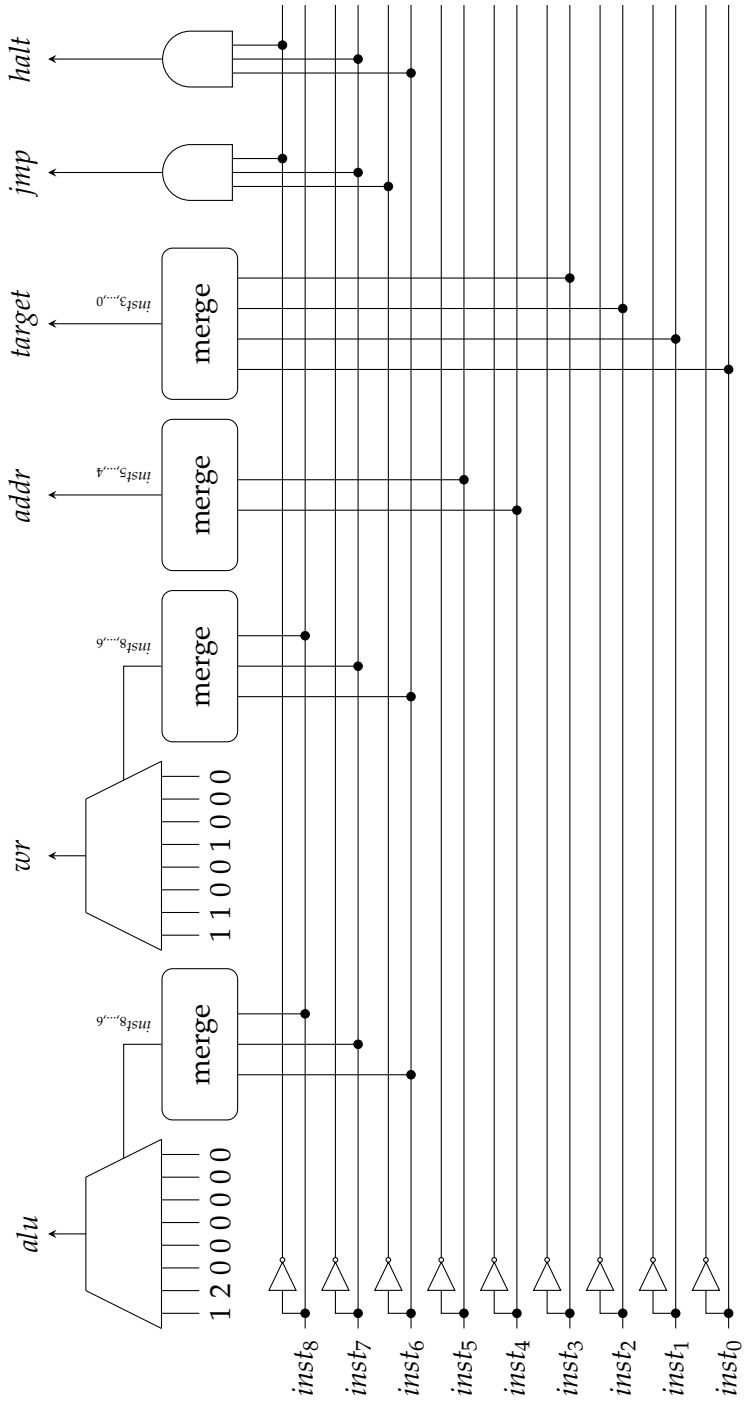


Figure 24: *The low-level decoder implementation for an example* 4*-register counter machine.*

`A:` halt computation
`B:` if register 2 equals 0 then goto instruction 5, else goto instruction $i + 1$
`C:` if register 10 equals 0 then goto instruction 5, else goto instruction $i + 1$
`D:` increment register 2, then goto instruction $i + 1$
`E:` decrement register 10, then goto instruction $i + 1$

best describes the instruction semantics?

▷ **Q184.** Figure 23 outlines, at a high level, a 4-register counter machine implementation; Figure 24 completes said implementation, detailing internals of the decoder component. Note that the multiplexer inputs should be read left-to-right, and use zero-based indexing. Using the left-most multiplexer in the decoder as an example, if the 3-bit control-signal derived from $inst$ is $001_{(2)} = 1_{(10)}$ then the 1-st input is selected; this means the output is $2_{(10)}$. Which of the following

`A:` L_i : **if** $\mathsf{R}_3 = 0$ **then goto** L_9 **else goto** L_{i+1}
`B:` L_i : **if** $\mathsf{R}_3 + 1 = 0$ **then goto** L_9 **else goto** L_{i+1}
`C:` L_i : $\mathsf{R}_3 \leftarrow \mathsf{R}_3 + 1$ **then goto** L_{i+1}
`D:` L_i : $\mathsf{R}_3 \leftarrow 0$ **then goto** L_{i+1}
`E:` None of the above

describes the semantics of a machine code instruction $100111001_{(2)}$ for this counter machine?

▷ **Q185.** Recall that instructions for a register machine will explicitly use a register file for source and destination operands; in contrast, instructions for a stack machine will implicitly use a stack for source and destination operands. Consider an instance of the latter, where a stack pointer sp and two algorithms PUSH and POP are used to control an in-memory stack. The associated ISA includes

$$\begin{array}{lcl} \texttt{ld}\ x & \mapsto & t_0 \leftarrow \mathsf{MEM}[x]\ ;\ \textsc{push}(\mathsf{sp}, t_0) \\ \texttt{st}\ x & \mapsto & t_0 \leftarrow \textsc{pop}(\mathsf{sp})\ ;\ \mathsf{MEM}[x] \leftarrow t_0 \\ \texttt{add} & \mapsto & t_0 \leftarrow \textsc{pop}(\mathsf{sp})\ ;\ t_1 \leftarrow \textsc{pop}(\mathsf{sp})\ ;\ \textsc{push}(t_1 + t_0) \\ \texttt{sub} & \mapsto & t_0 \leftarrow \textsc{pop}(\mathsf{sp})\ ;\ t_1 \leftarrow \textsc{pop}(\mathsf{sp})\ ;\ \textsc{push}(t_1 - t_0) \\ \texttt{mul} & \mapsto & t_0 \leftarrow \textsc{pop}(\mathsf{sp})\ ;\ t_1 \leftarrow \textsc{pop}(\mathsf{sp})\ ;\ \textsc{push}(t_1 \cdot t_0) \end{array}$$

noting the availability of load and store instructions which use a direct addressing mode. Which of the following

`A:` Compute $r = (a + b) \cdot (c - d)$, and store r in memory at address 4
`B:` Compute $r = (a + b - c) \cdot d$, and store r in memory at address 4
`C:` Compute $r = (b + a) \cdot (d - c)$, and store r in memory at address 4
`D:` Set the values of a and b to zero
`E:` Swap the values of a and b

best describes the purpose of

a this program

$$\texttt{ld}\ 0\ ;\ \texttt{ld}\ 1\ ;\ \texttt{add}\ ;\ \texttt{ld}\ 2\ ;\ \texttt{ld}\ 3\ ;\ \texttt{sub}\ ;\ \texttt{mul}\ ;\ \texttt{st}\ 4,$$

assuming that a, b, c, and d are initially located in memory at addresses 0, 1, 2, and 3 respectively?

b this program

$$\texttt{ld}\ 0\ ;\ \texttt{ld}\ 1\ ;\ \texttt{st}\ 0\ ;\ \texttt{st}\ 1,$$

assuming that a, b, c, and d are initially located in memory at addresses 0, 1, 2, and 3 respectively?

Part IX: Hardware test and debug

▷ **Q186.** Sara Latch is employed as a digital logic design engineer. As part of this role, Sara designs an 8-bit ripple-carry adder; said design is subsequently used within a larger project by her colleague Andy Gate. However, the project, once it has been manufactured, is found to function incorrectly: representing the adder inputs (i.e., x and y) and output (i.e., r) as unsigned, 8-bit hexadecimal integers, Andy observes that

$$x = 50_{(16)},\ y = 56_{(16)} \quad \rightsquigarrow \quad r = 86_{(16)}$$

so asks Sara for help. Although a definitive conclusion remains difficult, which of the following descriptions of the problem

`A:` The 0-th input bit x_0 has been hard-wired to 1 by mistake
`B:` The 1-st output bit r_1 has been hard-wired to 0 by mistake
`C:` Each AND gate in the 2-nd full-adder cell has been mistakenly replaced with an OR gate
`D:` Each OR gate in the 4-th full-adder cell has been mistakenly replaced with an AND gate
`E:` The carry-out of the 6-th full-adder cell is not connected to the carry-in of 7-th full-adder cell (meaning that carry-in is always 0)

is consistent with the evidence available?

▷ **Q187.** Imagine a circuit can compute the sum or difference of two n-bit operands x and y based on a control signal z, i.e.,

$$r = \begin{cases} x + y & \text{if } z = 0 \\ x - y & \text{if } z = 1 \end{cases}$$

You are asked to test whether it computes the correct result or not: assuming a large n, e.g., $n = 64$, explain **two** different ways to approach this challenge.

▷ **Q188.** Imagine you design then manufacture a ripple-carry adder circuit: given two unsigned, 8-bit integer inputs x and y, it will compute the unsigned, 8-bit sum $r = x + y$. You suspect two faults may have occurred during the manufacturing process, namely

a a stuck-at fault, whereby r_0 (the 0-th bit of r) always has the value 1, and

b an open connection fault, whereby the carry-out of the 1-st full-adder is not connected to the carry-in of the 2-nd full-adder.

For **each** case, explain which values of x and y could be used to (dis)prove your suspicion within a diagnostic testing strategy.

Part X: Processor design: general

▷ **Q189.** Imagine you have two micro-processors, denoted CPU_0 and CPU_1, which are designed by *different* vendors but support the *same* ISA: you write an assembly language program A, then use an assembler $\mathcal{A}$ to produce the associated machine code program $M = \mathcal{A}(A)$.

a Which of the following properties

`A:` The number of clock cycles elapsed
`B:` The amount of energy consumed
`C:` The number of instructions executed
`D:` The order that accesses to main memory are performed
`E:` The number of accesses to main memory performed

will remain the same between executions of M on CPU_0 and CPU_1 (assuming the same input is provided in both cases)?

b Which of the following properties

`A:` The amount of memory used to store data
`B:` The amount of memory used to store instructions
`C:` The average Cycles Per Instruction (CPI)
`D:` The content of memory before execution
`E:` The content of memory after execution

might differ between executions of M on CPU_0 and CPU_1 (assuming the same input is provided in both cases)?

▷ **Q190.** Arrange these phases of instruction execution in the **order** they must be performed:

a Write-back.

b Fetch.

c Execute.

d Decode.

▷ **Q191.** Identify **each** task which is typically associated with the instruction decode phase:

a Load operands from register file.

b Load data from memory.

c Increment program counter.

d Perform sign extension.

▷ **Q192.** The description of a stored program memory architecture often implies it is superior to, or at least more advanced than, a Harvard memory architecture. In reality both have advantages and disadvantages; discuss **two** advantages that a Harvard memory architecture can provide.

Part XI: Processor design: Instruction Set Architecture (ISA)

▷ **Q193.** Which of the following options would you **not** expect (or at least it would be uncommon) to see in the specification of an ISA:

`A:` The word size
`B:` The set of accessible general-purpose registers
`C:` The way a given instruction is expressed as machine code
`D:` The execution latency of a given instruction
`E:` The set of accessible special-purpose registers

▷ **Q194.** Which of the following statements best describes the semantics of a relative branch instruction?

`A:` The program counter is reset to zero, so is independent from the current program counter value
`B:` The branch target is at an offset from, and so is dependent on the current program counter value
`C:` The branch target is relatively far from the current program counter value
`D:` The branch target is greater than the current program counter value
`E:` The program counter is only updated if a condition is true

▷ **Q195.** Consider the specification of an ISA, which includes a) a fixed-length, 32-bit instruction encoding, and b) a byte addressable memory, with a 32-bit address space; instructions are required to be aligned in memory. Imagine the ISA is implemented by some micro-architecture, in which the program counter is a register comprised of n D-type latches: what is the minimum n possible?

`A:` 0
`B:` 14
`C:` 16
`D:` 30
`E:` 32

▷ **Q196.** Which of the following options would you **not** expect (or at least it would be uncommon) to see in the specification of an ISA:

`A:` The number of instructions supported by any micro-architecture which implements the ISA
`B:` The effect that execution of a division instruction has on any status register(s)
`C:` The type of (e.g., number of bits in) the operands used during execution of a division instruction
`D:` The conditions which would cause an error during execution of a division instruction
`E:` The energy consumed during execution of a division instruction

		Before	*After*
GPR[0]	=	$00000000_{(16)}$	$0000BAAD_{(16)}$
GPR[1]	=	$00000082_{(16)}$	$00000086_{(16)}$
GPR[2]	=	$00000004_{(16)}$	$00000004_{(16)}$
	⋮		
GPR[15]	=	$000000FC_{(16)}$	$00000100_{(16)}$
	⋮		
MEM[126]	=	$DE_{(16)}$	$DE_{(16)}$
MEM[127]	=	$C0_{(16)}$	$C0_{(16)}$
MEM[128]	=	$EF_{(16)}$	$EF_{(16)}$
MEM[129]	=	$BE_{(16)}$	$BE_{(16)}$
MEM[130]	=	$AD_{(16)}$	$AD_{(16)}$
MEM[131]	=	$BA_{(16)}$	$BA_{(16)}$
MEM[132]	=	$ED_{(16)}$	$ED_{(16)}$
MEM[133]	=	$FE_{(16)}$	$FE_{(16)}$
	⋮		

Figure 25: *A subset of ARMv7A architectural state before (left) and after (right) execution of some instruction.*

▷ **Q197.** Consider the specification of an ISA, which includes b) a fixed-length, 16-bit instruction encoding, and a) a 16-element register file. What is the maximum number of 3-address arithmetic instructions the specification can include?

`A:` 1
`B:` 3
`C:` 4
`D:` 16
`E:` 65536

▷ **Q198.** Figure 25 shows a subset of ARMv7A architectural state, i.e., the content of *some* general-purpose registers and memory: the LHS (resp. RHS) shows the state before (resp. after) execution of some instruction. From the list below, which instruction do you think was executed?

`A:` The load `ldrh r0, [r1], #4`
`B:` The branch `b 100`
`C:` The store `str r0, [r1, #4]`
`D:` The addition `add r1, r1, r2`
`E:` The move `mov r2, r1`

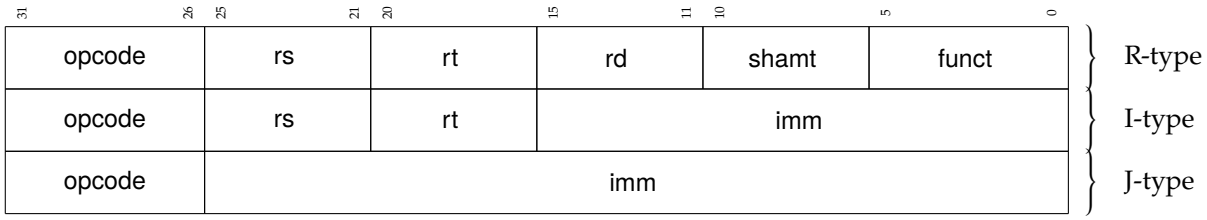


Figure 26: *A diagramatic description of the 3 instruction formats used by MIPS32.*

▷ **Q199.** Consider the 3 instruction formats used by MIPS32, as shown in Figure 26. Imagine the number of general-purpose registers is halved: which of the following statements

`A:` There must be a greater number of R-type instructions

`B:` There must be a greater number of I-type instructions
`C:` There could be 9 times the number of R-type instructions
`D:` R-type instructions could be 2 bits smaller
`E:` I-type instructions could have an immediate field which is 2 bits larger

is most likely to then be true?

▷ **Q200.** Imagine that an ISA includes an instruction whose semantics are

$$\mathsf{GPR}[x] \leftarrow \mathsf{MEM}[\mathsf{MEM}[\mathsf{GPR}[y]] + \mathsf{GPR}[z]].$$

Based on this instruction *alone*, which of the following

`A:` CISC
`B:` RISC
`C:` Neither CISC nor RISC
`D:` Both CISC and RISC

best classifies the ISA?

▷ **Q201.** Consider two instructions, drawn from different ISAs:

1. $\mathsf{R}_0 \leftarrow \mathsf{R}_0 + 13_{(10)}$
2. $\mathsf{R}_0 \leftarrow \mathsf{R}_1 + 13_{(10)}$

Assuming each instruction is representative of the associated ISA, which of the following options

	1.	2.
`A`	stack machine	register machine
`B`	register machine	accumulator machine
`C`	accumulator machine	register machine
`D`	accumulator machine	accumulator machine
`E`	register machine	stack machine

provides the most precise classification of both the ISAs and the machines which implement them?

▷ **Q202.** MIPS32 is an ISA which fixes $\mathsf{GPR}[0] = 0$ (i.e., the register always yields 0 when read, with any write ignored) and includes instructions for

$$\begin{array}{lcl} \texttt{add} & : & \mathsf{GPR}[x] \leftarrow \mathsf{GPR}[y] + \mathsf{GPR}[z] \\ \texttt{sub} & : & \mathsf{GPR}[x] \leftarrow \mathsf{GPR}[y] - \mathsf{GPR}[z] \\ \texttt{addi} & : & \mathsf{GPR}[x] \leftarrow \mathsf{GPR}[y] + imm \end{array}$$

but *not*

$$\texttt{subi} : \quad \mathsf{GPR}[x] \leftarrow \mathsf{GPR}[y] - imm$$

Given this information, which of the following explanations

`A:` One can realise the semantics of `subi` by using `addi`
`B:` One can realise the semantics of `subi` by using `addi` and `add`
`C:` There is never a need to subtract an immediate value from a register
`D:` There were no unused opcodes to allocate to `subi`
`E:` The immediate value imm is too large to use during subtraction, and may cause overflow

for omitting `subi` seems the most plausible?

▷ **Q203.** While executing an ARMv7-A program, imagine you find that $\mathsf{PC} = x$ *before* fetch-decode-execute cycle i (i.e., the instruction executed in execution cycle i is fetched from address x). In certain cases, the instruction located at address x makes it *possible* that $\mathsf{PC} = x$ *after* fetch-decode-execute cycle i (i.e., the instruction executed in execution cycle $i + 1$ is *also* fetched from address x). Which of the following options

`A:` `add r15, r15, r2`
`B:` `j r3`
`C:` `mv pc, r7`
`D:` All of the above

`E:` None of the above

describes such a case?

▷ **Q204.** Consider a micro-processor, connected to a byte-addressable memory whose capacity is 64KiB. The processor supports a CISC-style ISA, which uses a fixed-length instruction encoding; it specifies 16 general-purpose registers, and a word size of 64 bits.

Instructions in the ISA allow a style of access to operands making it a memory-memory architecture. In each 3-address instruction, of which there are 120, each operand exists in memory and is specified by an (independent) indexed addressing mode involving an immediate-based base plus a register-based offset; using an addition instruction for example, the ARM-like syntax

```
add [ri, #i], [rj, #j], [rk, #k]
```

maps to the following semantics

$$\mathsf{MEM}[\mathtt{i} + \mathsf{GPR}[\mathtt{ri}]] \leftarrow \mathsf{MEM}[\mathtt{j} + \mathsf{GPR}[\mathtt{rj}]] + \mathsf{MEM}[\mathtt{k} + \mathsf{GPR}[\mathtt{rk}]].$$

A base address in such an instruction must be able to specify any address in memory. Given the information available, what is the minimum instruction length for this ISA?

`A:` 16 bits
`B:` 35 bits
`C:` 64 bits
`D:` 67 bits
`E:` 120 bits

▷ **Q205.** In a seminal essay on the relationship between compilers and computer architecture, Wulf makes two statements:

a "*[a]s a compiler writer, I must applaud the trend in many recent machines to allow each instruction operand to be specified by any of the addressing modes*"

b "*provide primitives, not solutions*"

Which of the following ISA design philosophies

`A:` RISC
`B:` CISC
`C:` Both of the above
`D:` None of the above

is best aligned with each statement?

▷ **Q206.** MIPS32 is an ISA which uses fixed-length, 32-bit instruction formats. The ISA includes a set of relative, conditional branch instructions such as

$$\mathtt{beq\ rs,\ rt,\ imm} \mapsto \textbf{if } \mathsf{GPR}[rs] = \mathsf{GPR}[rt] \textbf{ then } \mathsf{PC} \leftarrow \mathsf{PC} + \mathrm{ext}^{32}_{\pm}(\mathtt{imm} \ll 2),$$

each of which involves a 16-bit, signed immediate imm represented using two's-complement; note that $\mathrm{ext}^{32}_{\pm}(x)$ is used to denote sign-extension of x to 32-bits. Which of the following options

`A:` ±16KiB
`B:` ±18KiB
`C:` ±32KiB
`D:` ±34KiB
`E:` ±128KiB

is the most reasonable description of the branch range (if we measure it in bytes)?

▷ **Q207.** Imagine the ARMv7A instruction

```
ldrh r0, [ r1 ], r2
```

is executed by a compliant micro-processor. Considering the *entire* fetch-decode-execute cycle associated with said instruction, how many bytes does the ISA suggest are transferred between the micro-processor and memory?

A: 1
B: 2
C: 4
D: 6
E: 8

▷ **Q208.** Imagine you use a micro-processor to write, compile, then execute a C program; the micro-processor has a byte-addressable memory. The program defines a pointer

```
uint32_t* p;
```

and, while debugging a problem, you find out that

```
sizeof( p ) = 8.
```

How many addressable bytes of memory are there, i.e., how large is the associated address space?

A: 2^{64}
B: 2^{32}
C: 2^{16}
D: 2^{8}
E: None of the above

▷ **Q209.** Consider a hypothetical ISA, which includes a single instruction and no registers: the assembly language short-hand (left) and formal semantics (right) of this instruction, i.e.,

$$\texttt{sbn x, y, z} \;\mapsto\; \mathsf{MEM}[\texttt{x}] \leftarrow \mathsf{MEM}[\texttt{x}] - \mathsf{MEM}[\texttt{y}] \text{ ; } \textbf{if } \mathsf{MEM}[\texttt{x}] < 0 \textbf{ then } \mathsf{PC} \leftarrow z,$$

could be summarised informally as "*subtract, then branch if negative*". Note that the instruction uses 3 immediate operands, namely `x`, `y`, and `z`, each of which is used as an address. Imagine that two programs are written using this ISA, then executed on an associated micro-architecture. The programs are described as follows

a The memory content is

i	MEM[i]
0	`sbn 100, 100, 0`
1	`sbn 100, 101, 2`
2	`sbn 102, 100, 3`
3	⋮

with execution starting with PC = 0 and terminating once PC = 3.

b The memory content is

i	MEM[i]
0	`sbn 100, 100, 0`
1	`sbn 100, 101, 2`
2	`sbn 102, 102, 0`
3	`sbn 102, 100, 4`
4	⋮

with execution starting with PC = 0 and terminating once PC = 4.

where MEM is being (partially) described using a table where the left-hand column is an address (i.e., i) and the right-hand column is the associated content (i.e., MEM[i]), and, purely for convenience, we list the instructions rather than encodings of them. For each program, which of the following

A: It adds MEM[101] to MEM[102], storing the result in MEM[102]
B: It subtracts MEM[101] from MEM[102], storing the result in MEM[102]
C: It copies MEM[101] into MEM[102]
D: It clears MEM[102], i.e., sets MEM[102] = 0
E: None of the above, e.g., it will never terminate

best describes the purpose?

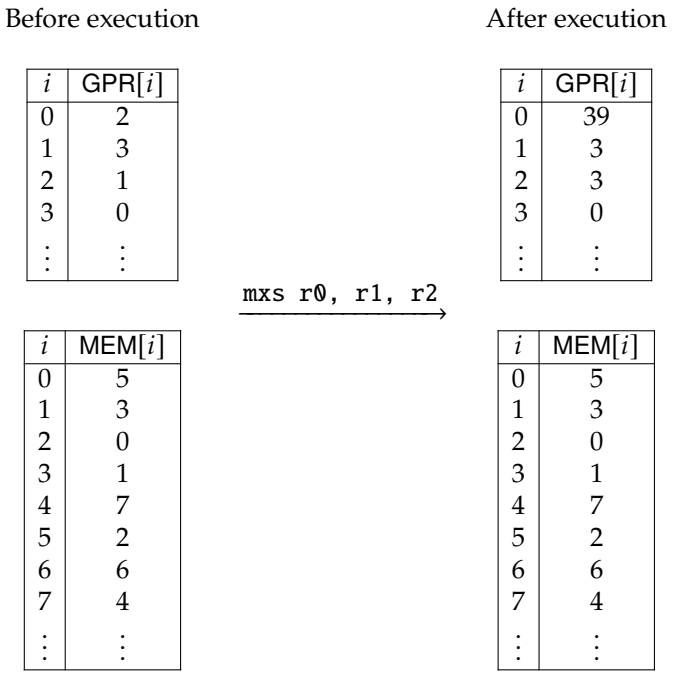


Figure 27: *The general-purpose register file and byte-addressable memory state, before (left) and after (right) execution of a hypothetical memory access instruction.*

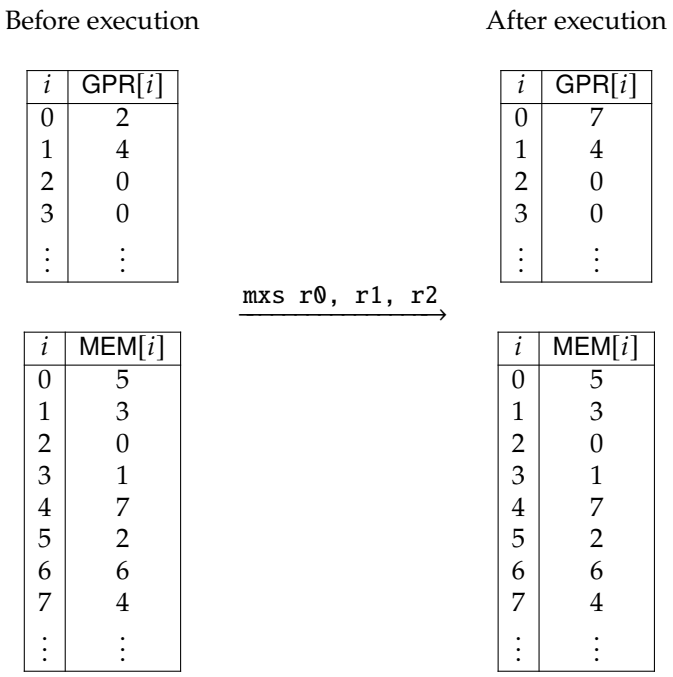


Figure 28: *The general-purpose register file and byte-addressable memory state, before (left) and after (right) execution of a hypothetical memory access instruction.*

▷ **Q210.** Imagine a hypothetical "ARMv7-A style" memory access instruction

```
mxs r0, r1, r2
```

is executed. Which of the following options

`A:` Direct store
`B:` Indexed, 8-bit load
`C:` Indexed, 16-bit load
`D:` Auto-indexed, 8-bit load
`E:` Auto-indexed, 16-bit load

is the most reasonable description of the instruction semantics

a based on Figure 27,

b based on Figure 28,

where both Figures offer a (partial) description of the general-purpose register file and byte-addressable memory accessed. The left-hand description reflects *before* execution of the instruction, whereas the right-hand description reflects *after* execution of the instruction; within each description, the left-hand column lists an address (i.e., i) whereas the right-hand column lists the associated content (i.e., $\mathsf{GPR}[i]$ or $\mathsf{MEM}[i]$).

▷ **Q211.** ARMv7-A allows *any* instruction to be conditionally executed, using the concept of predicated execution. Consider a hypothetical alternative ARMv7-X, which is identical to ARMv7-A except it does *not* support predicated execution. Which of the following instructions

```
A:  add r0, r1, r2
B:  nop
C:  ldr r0, [ r1 ]
D:  mov r0, r1
E:  cmp r0, r1
```

could be viewed as having the *same* semantics in both ARMv7-A and ARMv7-X?

▷ **Q212.** Classify each of the following statements as either **true** or **false**, then explain why using at most a few sentences.

a In a load-store architecture, execution of a single instruction can both load a value from memory and store a value into memory.

b Within the context of ARMv7-A, `mov r0, #0` is the only single instruction able to set the 0-th general-purpose register to zero.

c Within a micro-processor design, the program counter *must* be addressable as a general-purpose register.

d Once defined, it is more difficult to alter an ISA than it is to alter the design of an associated micro-architecture.

e Consider a micro-processor based on a von Neumann (or stored-program) architecture; the number of instructions it executes per unit of time is limited by the access latency of memory.

f Doubling the word size of an ISA (e.g., from 32 to 64 bits), will half the execution latency of any program executed on the associated micro-architecture.

▷ **Q213.** Comparing a fixed-length instruction encoding scheme to a variable-length alternative, identify the statement which is **incorrect**:

a A variable-length scheme can more easily cope with instructions with many operands a fixed-length scheme.

b A fixed-length scheme means instructions are easier to decode than a variable-length scheme.

c A fixed-length scheme allows larger immediate operands than a variable-length scheme.

d A variable-length scheme places fewer restrictions on the program counter than a fixed-length scheme.

▷ **Q214.** Describe the following addressing modes, giving examples of their use in a C program:

a Immediate.

b Direct.

c Indirect.

d Indexed.

▷ **Q215.** a A change in the design of a particular processor means it will support the concept of predicated execution. Explain what this means and translate the following C fragment into an assembly language style program for the processor:

```
int t = 0;

for( int i = 0; i < n; i++ ) {
  if( ( x >> i ) & 1 ) {
    t = t + 1;
  }
}
```

Use predicated execution or branch instructions where appropriate, and take care to justify your choice in each case.

b It is suggested that the code density of programs executed by the processor is too low. Explain what is meant by code density and outline **two** approaches to improving this situation.

▷ **Q216.** You have been asked to define the instruction set of a new processor. The processor contains a Program Counter (PC), four general-purpose registers (A, B, C and D), uses 8-bit instructions and has a 256-byte memory that is addressed indirectly via the registers. The instructions set includes the following instructions:

Name	Operands	Meaning
LDA	x	Loads constant x, with $0 \leq x \leq 127$, into register A.
MOV	R, S	Moves register S into register R.
NOT	R	Performs a bit-wise complement of register R.
NOP		Performs no operation.
NEG	R	Performs two's-complement negation of register R.
BEZ	R, x	Branches by incrementing the program counter by an offset x, with $-8 \leq x \leq 7$, if register R is equal to 0.

As such, one might write the example program

```
LDA 10
MOV B, A
NEG B
BEZ B, 4
```

to load the constant 10 into register A, then move this value into register B and finally negate it before jumping forward by four instructions if register B turns out to be zero.

a Design an encoding for the instructions given above.

b What is the maximum number of instructions with

i 2 register operands,

ii 1 register operand,

iii 0 register operands

that you can add to the instruction set based on your encoding?

c Write down a more complete instruction set, adding those you view as currently missing and giving reasons why you want to add them.

d Finally, write down the instruction encoding for this complete instruction set.

▷ **Q217.** There are 32 pieces on a chess board represented by an (8×8)-element grid; a black and a white version of 1 king, 1 queen, 2 bishops, 2 knights, 2 rooks and 8 pawns exist.

You are involved in the design of a robot that automates the game of chess **only** in terms of moving the physical pieces around. The robot needs to be fed a list of moves so that it knows how to move a piece on one board position to another position; it does not care about the legality of moves. However, it is intelligent enough to know whose turn it is (i.e., black or white), how to find a piece on the board given its unique name (e.g., pawn #1) and can cope with capturing pieces if a move places a piece on an already occupied square. It cannot play any version of chess with the advanced rules of promotion, castling, resignation or en passant: it simply moves and captures pieces. It can deal with the fact that pawns can move two spaces forward from their starting position but otherwise has no memory.

Design a compact encoding of instructions that can be fed to the robot to provoke the movement of any piece on the board. State any assumptions you make in your design and the advantages and disadvantages of the result.

▷ **Q218.** Imagine you are asked to design a RISC style instruction set architecture for a new 32-bit processor which is modelled roughly on the MIPS32 design. The processor has a 32-entry general-purpose register file, and the instruction set is to include:

- Arithmetic, logic and comparison instructions that use a mix of general-purpose register and immediate operands.
- Memory access instructions that use an single indexed addressing mode whereby a base address is specified by a general-purpose register operand and an offset is specified by an immediate operand.
- Branch and jump instructions that include both conditional and unconditional variants and use mix of general-purpose register and immediate operands to specify the absolute and offset target address.

a One engineer wants to use a fixed length instruction encoding while another suggests that a variable length encoding would be better.

i Explain the advantages and disadvantages of each approach.

ii Select **one** approach and using your selection, design basic instruction encoding formats for the instruction types listed above.

b The "add immediate" instruction adds the contents of a general-purpose register to a 16-bit immediate and stores the result back into a general-purpose register. For example, the instruction

$$\mathsf{GPR}[1] \leftarrow \mathsf{GPR}[2] + 3$$

takes the contents of general-purpose register 2, adds the immediate value 3 to it and stores the result into general-purpose register 1. Using just the instruction encoding formats from above, show how

i $\mathsf{GPR}[4] \leftarrow \mathsf{GPR}[8] + 10$

ii $\mathsf{GPR}[4] \leftarrow \mathsf{GPR}[8] - 10$

are encoded into 32-bit "add immediate" instructions; state any assumptions you make.

c A major customer for the processor has identified the following C function as crucial to the performance of their applications:

```
int H( uint32_t x ) {
  int t = 0;

  for( int i = 0; i < 32; i++ ) {
    if( ( x >> i ) & 1 )
      t = t + 1;
  }

  return t;
}
```

Explain what this function does and how you might alter the processor design (including any changes to the ALU) to satisfy the requirements of this customer.

d There is some debate about which endian convention the processor should use. Assuming the memory is byte addressed, consider the execution of an instruction that stores a 32-bit decimal integer $305419896_{(10)}$ at address 0. List the content of the addresses updated by the store instruction if the processor uses

  i a little-endian byte ordering, or

  ii a big-endian byte ordering.

e The team responsible for writing a compiler for the new processor is concerned about the number of general-purpose registers. Explain the positive and negative implications for the instruction set if the number of general-purpose registers is

  i increased from 32 to 64, or

  ii decreased from 32 to 16.

In **each** case, describe how this might alter performance of programs that execute on the processor.

▷ **Q219.** a It can be difficult to definitively categorise a given processor design as either RISC or CISC. Even so, explain **four** distinct features which most RISC processor exhibit.

b Some processors make use of a status register; this often includes a set of 1-bit flags that signal when specific conditions or events occur. An example is the carry flag, which is set (resp. cleared) when a carry occurs (resp. does not occur) during execution of integer addition instructions.

  i In some processors the status register is accessed explicitly (e.g., "copy status register into general-purpose register"), in others it is accessed implicitly (e.g., "add with carry"). Outline **one** distinct advantage of **each** approach.

  ii Name **two** other flags that a status register may include, and the rationale for their inclusion.

c Imagine a given 32-bit processor does not have a carry flag.

  i Outline an alternative mechanism by which the processor could support access to the carry resulting from integer addition.

  ii Imagine two 32-bit values x and y are held in GPR[1] and GPR[2], and a 1-bit carry-in ci is held in GPR[3]. Using standard arithmetic, logic and comparison instructions **only**, describe how an integer add with carry operation can be achieved *without* processor support for carries. Specifically, write a sequence of instructions that computes the 33-bit result $x + y + ci$, storing the 32-bit sum s in GPR[4] and 1-bit carry-out co in GPR[5].

▷ **Q220.** Consider an 8-bit RISC-like processor that uses fixed-length 8-bit instructions. It has two 8-bit general-purpose registers called A and B, and a program counter called PC.

a In reference to the processor, briefly explain what these terms mean:

  i general-purpose and special-purpose register,

  ii absolute and relative branch.

b The processor has two types of load instruction, namely

$$A \leftarrow \mathsf{MEM}[B]$$

and

$$A \leftarrow \mathsf{MEM}[i],$$

where i is an immediate operand and MEM represents memory.

  i For **each** load instruction above, outline the largest effective address possible (i.e., the largest element in MEM that can be accessed).

ii Explain **two** approaches (e.g., describe any additional features) which could allow the processor to address a larger memory. For each approach again outline the largest effective address possible, and also any advantages and disadvantages.

c Consider the following C function

```
short strlen( char* S ) {
  short i = 0;

  while( S[ i ] != '\0' ) {
    i = i + 1;
  }

  return i;
}
```

where `char` and `short` are signed 8-bit and 16-bit integer data-types. Explain **one** problem you might face when implementing this function on the processor, and features the processor could provide to allow a solution.

▷ **Q221.** This question relates to a C function (left-hand side) and an associated, assembly language style implementation (right-hand side) for an imaginary 32-bit RISC processor:

```
void H( uint32_t* A, int n ) {
  int t = 0;

  for( int i = 0; i < n; i++ ) {
    for( int j = 0; j < 32; j++ ) {
      if( ( A[ i ] >> j ) & 1 ) {
        t = t + 1;
      }
    }
  }

  return t;
}
```

$$
\begin{array}{ll}
 & \ldots \\
 & \mathsf{GPR}[2] \leftarrow 0 \\
 & \mathsf{GPR}[3] \leftarrow 0 \\
loop_i: & \text{if } \mathsf{GPR}[3] \geq \mathsf{GPR}[2], PC \leftarrow exit_i \\
 & \mathsf{GPR}[4] \leftarrow 0 \\
loop_j: & \text{if } \mathsf{GPR}[4] \geq 32, PC \leftarrow exit_j \\
 & \mathsf{GPR}[5] \leftarrow \mathsf{MEM}[\&\mathtt{A} + \mathsf{GPR}[3]] \\
 & \mathsf{GPR}[5] \leftarrow \mathsf{GPR}[5] \gg \mathsf{GPR}[4] \\
 & \mathsf{GPR}[5] \leftarrow \mathsf{GPR}[5] \wedge 1 \\
 & \text{if } \mathsf{GPR}[5] = 0, PC \leftarrow skip \\
 & \mathsf{GPR}[2] \leftarrow \mathsf{GPR}[2] + 1 \\
skip: & \mathsf{GPR}[4] \leftarrow \mathsf{GPR}[4] + 1 \\
 & PC \leftarrow loop_j \\
exit_j: & \mathsf{GPR}[3] \leftarrow \mathsf{GPR}[3] + 1 \\
 & PC \leftarrow loop_i \\
exit_i: & \ldots
\end{array}
$$

The function computes the Hamming weight of all unsigned 32-bit integers in an `n`-element array called `A`; the assembly language implementation allocates registers GPR[1], GPR[2], GPR[3] and GPR[4] to hold `n`, `t`, `i`, and `j`.

a Explain the difference between logical and arithmetic right-shift operations, and state which is required for this function.

b The programmer estimates that executing the compiled function on a given processor means

- 50% of instructions perform arithmetic via the ALU, and take an average of 1 cycle,
- 10% of instructions perform memory accesses and take an average of 4 cycles,
- 40% of instructions perform branches and take an average of 2 cycles.

Calculate the average Cycles Per Instruction (CPI) of the processor based on use of this function as a benchmark kernel.

c Using motivating examples from the function, suggest **three** addressing modes that would be useful; in each case, explain how the addressing mode works.

d The programmer and processor designers are interested in removing the explicit branch

$$\text{if } \mathsf{GPR}[5] = 0, PC \leftarrow skip$$

in some way that does not alter the result computed. Describe **one** hardware-oriented approach (where you can alter the processor instruction set and/or micro-architecture) **or one** software-oriented approach (where you cannot) by which this can be achieved.

e The programmer and processor designers are interested in further ways of reducing the execution time of this function (i.e., in addition to the above). Describe **one** hardware-oriented approach (where you can alter the processor instruction set and/or micro-architecture) **and one** software-oriented approach (where you cannot) by which this can be achieved.

Part XII: Processor design: basic micro-architecture

▷ **Q222.** Identify the statement which is **incorrect**:

a A register file with many read ports implies higher implementation cost.

b A register file with many read ports implies fewer sequential read operations.

c More general-purpose registers implies greater register pressure.

d More general-purpose registers implies larger register addresses.

▷ **Q223.** Consider a new 16-bit processor, whose specifications include:

- 16 general-purpose registers.
- 8 ALU instructions that take two input register operands and write a result into a third output register operand.
- 2 memory access instructions that take two input register operands and one output register operand. 2 memory access instructions that take one input register operand, one output register operand and a 4-bit immediate offset.
- 4 branch instructions that take an input register operand and an 8-bit immediate offset.

Imagine you are on the design team for a this processor:

a Devise an instruction encoding for the processor, taking care to explain any advantages and disadvantages of your design.

b Draw a block diagram of the basic components within this processor, explain the function of each one and how they are connected together.

c Consider an instruction that adds together integers read from two registers and writes the result into a third register; using such an instruction as an example, describe the steps involved during the following phases of execution:

i Fetch.

ii Decode.

iii Execute.

d The control unit that operates the data-path can be implemented using hardwired logic or a micro-code based system. Explain the idea of both of these methods, including the advantages and disadvantages of each.

▷ **Q224.** The ARM7TDMI is aimed mainly at mobile and embedded processor markets; there are numerous interesting features in the design which have helped make it a success.

a Most ARM arithmetic instructions takes four operands: three register addresses x, y and z and an immediate i. The semantics for addition, for example, are thus roughly

$$\mathsf{GPR}[x] \leftarrow \mathsf{GPR}[y] + (\mathsf{GPR}[z] \cdot 2^i).$$

Using a C program as an example, demonstrate when this form of instruction might be useful.

b Assuming a byte-addressed memory, one form of ARM load instruction has roughly the following semantics

$$\begin{aligned} \mathsf{GPR}[x] &\leftarrow \mathsf{MEM}[\mathsf{GPR}[y]] \\ \mathsf{GPR}[y] &\leftarrow \mathsf{GPR}[y] + 1 \end{aligned}$$

given register addresses x and y. Explain what this addressing mode is called and, using a C program as an example, demonstrate when it might be useful.

c As standard, an ARM7TDMI has no instruction or data cache. Outline **two** reasons the designers may have made this choice.

d Rather than being defined as a special-purpose register, the program counter is accessible as $GPR[*][15]$. Outline **two** reasons the designers may have made this choice.

Part XIII: Processor design: advanced micro-architecture

▷ **Q225.** Consider a processor which implements the ARMv7-A ISA, using a 5-stage pipelined micro-architecture: the stages reflect a "classic" RISC-like approach, in that they are 1) *fetch*, which fetches instructions from memory using the program counter, 2) *decode*, which decodes instructions and reads operands from the register file, 3) *execute*, which and executes ALU-based instructions and resolves branch conditions, 4) *memory access*, which performs access to, i.e., loads from and stores to, memory, and 5) *write-back*, which writes results into the register file.

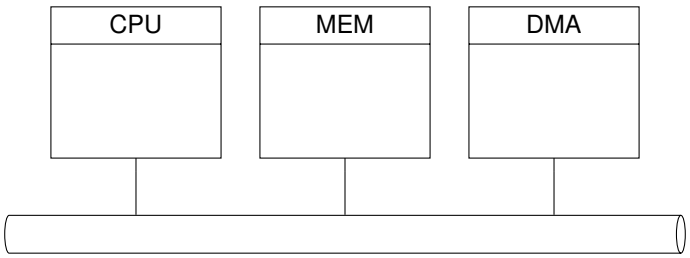
As a short-hand, let F, D, E, M, and W denote those pipeline stages, and F/D, D/E, E/M, and M/W denote the pipeline registers placed between them. If the following sequence of instructions

```
mov a1, a2
ldr r0, [ r1 ]
add r2, r3, r4
sub r5, a1, r2
```

is executed by this processor, where will the operands used by the `sub` instruction come from when it reaches the execute stage?

`A:` D and D
`B:` D/E and D/E
`C:` D/E and E/M
`D:` M/W and E/M
`E:` W and W

▷ **Q226.** Consider the following diagram of a computer system



which involves a processor, some memory, and a Direct Memory Access (DMA) controller, all connected by a memory bus; the processor implements the ARMv7-A ISA, and uses a pipelined micro-architecture. The DMA controller can be used to, e.g., accelerate transfer of data from one region of memory to another: rather than the processor performing the transfer by executing instructions, the DMA controller, which can operate independently and so concurrently, does so instead.

If the DMA controller needs to use or is using the memory bus at the same time as the processor needs to, the processor is forced to wait. Even though the processor is continually executing instructions, the DMA controller will often find the memory bus unused: of the following, which is the most plausible explanation for this?

`A:` Some instructions may not perform access to data, e.g., are not `ldr` or `str` instructions
`B:` The region of memory accessed by the DMA controller may be protected
`C:` An interrupt may have occurred

D: Some instructions and/or data may reside in an L1 cache rather than memory
E: The processor may be executing a `nop` instruction

▷ **Q227.** a It has been suggested that in designing a new processor, a pipelined approach should be used instead of a non-pipelined alternative. Explain how and why a pipelined design could improve performance; draw a data-path block diagram to describe which components fit into which pipeline stages and why.

b In the context of pipelined processor design, explain the meaning of the following terms and give examples of each:

i Structural dependency.
ii Control dependency.
iii Data dependency.

c A pipelined processor might stall when such dependencies are encountered; the performance is decreased as a result. Explain how the three types of dependency listed above can be avoided so that stalls are minimised.

d In a pipelined processor design, branches can cause control dependencies that stall the processor until resolved. There are several options for dealing with this problem; select **two** of the **three** options below and describe how they work:

i Branch delay slot.
ii Predicated execution.
iii Branch prediction.

For each option, your answer should explain how the scheme works, how it reduces the performance impact, and the trade-off between complexity and performance for the programmer and/or in the hardware.

▷ **Q228.** Consider the following C fragment which adds together two n-element vectors B and C to produce a result A where each element is a 32-bit integer:

```
for( int i = 0; i < n; i++ ) {
  A[ i ] = B[ i ] + C[ i ];
}
```

The fragment can be parallelised since each iteration of the loop is independent. Explain how this could be exploited using

a A vector processor.

b A VLIW processor.

▷ **Q229.** a Using a diagram and an example program if appropriate, describe the motivation for and design of (i.e., components in) a vector processor.

b Describe **two** reasons why use of a vector processor may improve performance (for suitably written programs) versus a scalar processor.

c The algorithm below describes the core loop of RC6, which operates in-place on a state, and uses an array of pre-computed round keys; each element of the state (i.e., A, B, C and D) and round key (i.e., each $K[i]$) are 32-bit unsigned integers. The loop includes an operation $x \lll y$ which denotes left-rotation of x by a distance y (with the constraint $0 \leq y < 32$, i.e., just $y_{4 \ldots 0}$ is used as the distance).

Input: The state (A, B, C, D) which is updated in-place, and K, an array of round keys

1 **for** $i = 0$ **upto** 44 **step** +2 **do**
2 $\quad t_0 \leftarrow (B \cdot ((2 \cdot B) + 1)) \lll 5$
3 $\quad t_1 \leftarrow (D \cdot ((2 \cdot D) + 1)) \lll 5$
4 $\quad A \leftarrow ((A \oplus t_0) \lll t_1) + K[i + 0]$
5 $\quad C \leftarrow ((C \oplus t_1) \lll t_0) + K[i + 1]$
6 $\quad t_2 \leftarrow A$
7 $\quad A \leftarrow B$
8 $\quad B \leftarrow C$
9 $\quad C \leftarrow D$
10 $\quad D \leftarrow t_2$
11 **end**

i The Intel Pentium 4 processor allows vector processing via the MMX and SSE extensions; it packs four 32-bit sub-words into each 128-bit register. Stating any assumptions (for example about the instruction set) and using either a diagram or pseudo-code, write a vectorised implementation of the straight-line RC6 loop body.

ii Stating any assumptions, estimate the performance improvement as a result of vectorisation. Comment on any reason(s) your result is lower than the theoretical limit, and any changes you might make to the processor design or instruction set to solve this problem.

▷ **Q230.** a In a superscalar processor, performance is reliant on the front-end supplying enough instructions to keep computational resources in the back-end busy. Explain how and when speculative execution helps to solve this problem.

b Speculative execution is often based on prediction of branch behaviour (i.e., if a branch is taken and what the target address is). One might consider either

i static branch prediction, or

ii dynamic branch prediction.

For **each** case, describe a mechanism to predict branch behaviour; take care to describe the structures within the processor, how they are used and any advantages or disadvantages of the mechanism.

c Imagine that a given processor demands function calls are implemented using dedicated "call" and "return" instructions instead of generic branches.

i Using an example and stating any reasonable assumptions, show how such instructions would be used to implement a function call.

ii Outline a mechanism that could provide more accurate branch target prediction for these instructions than in those from the previous question.

Part XIV: The memory hierarchy

▷ **Q231.** Consider a processor which uses an L1 data cache to accelerate accesses to memory. A write policy controls operation of the cache whenever the processor tries to write to memory (i.e., executes a store instruction). Which of the following write policies would you expect to generate the most transactions across the bus connecting the processor and memory?

`A:` Write-through plus write-allocate
`B:` Write-back plus write-allocate
`C:` Write-through plus write-around
`D:` Write-back plus write-around

▷ **Q232.** Consider a 32-bit processor, connected to a byte-addressable RAM. The processor makes use of an 8-way set-associative L1 cache to accelerate accesses to the RAM; the cache has a total capacity of 128KiB, which is divided into 64B cache lines. Calculate the number of bits in the word, set, and tag fields derived by the cache from a given address.

	Word	Set	Tag
A	8	8	16
B	24	8	0
C	64	8	32
D	6	8	18
E	2	2	28

▷ **Q233.** Draw a diagram of a typical memory hierarchy explaining the types of device one would expect to find in each level, their key characteristics and the principles that allow the memory hierarchy to improve system performance.

▷ **Q234.** A given processor accesses memory using 32-bit addresses. Suppose there is a cache between the processor and a byte-addressable main memory; the cache has a total size of 512B split into lines, each of which holds 8B. For each of the following architectures specify how the 32-bit address would be used by the address translation mechanism in the cache:

a A direct-mapped cache.

b A 2-way set-associative cache.

c A 4-way set-associative cache.

d A fully-associative cache.

For example, in the direct-mapped case you would need to specify which bits of the address are used for the tag, the line number and the word address.

▷ **Q235.** Cache-misses can be classified as either compulsory misses, capacity misses or conflict misses. For **each** class of cache-miss, explain why it might occur and how changes in the cache architecture can reduce how often it occurs.

▷ **Q236.** A given 32-bit processor has a unified level-one (or L1) cache. The cache has a total size of 1kB, holds 4 bytes per-line and is direct-mapped. A programmer writes, compiles and executes the following program on the processor:

```
char A[ 1024 ], B[ 1024 ], C[ 1024 ];

int main( int argc, char* argv[] ) {
  for( int i = 0; i < 1024; i++ ) {
    A[ i ] = B[ i ] + C[ i ];
  }
}
```

a Calculate the number of bits of an address required for each of the following quantities required to locate items in the cache:

i word address,

ii line address,

iii tag

and show how they are mapped onto the address.

b Explain the following terms in relation to the cache hardware using the above program as an example:

i spatial locality,

ii temporal locality,

iii cache interference or contention.

c Describe the sequence of cache-hits and cache-misses caused by accesses to `A[ i ]`, `B[ i ]` and `C[ i ]` as the loop progresses.

d The definition of arrays `A`, `B` and `C` are changed to read:

```
char A[ 1028 ], B[ 1028 ], C[ 1028 ];
```

i Explain how this changes the sequence of cache-hits and cache-misses described above and why it improves on the original program.

ii Outline a different cache architecture that could increase the performance of the original program without changing.

▷ **Q237.** a Using one or more C programs to illustrate your answers, explain the concepts of

i temporal locality, and

ii spatial locality.

b A given direct-mapped cache holds a total of $c = l \cdot w$ bytes in l lines, each containing w 1-byte words. Let l and w be powers-of-two, i.e., $l = 2^{l'}$ and $w = 2^{w'}$ for some l' and w'.

i For the given 8-bit effective address x, compute the word number, line number and tag where

i. $l' = 3$ and $w' = 3$, with $x = 42$, and
ii. $l' = 2$ and $w' = 4$, with $x = 81$.

ii For a fixed c one can select a

i. larger l, implying a smaller w, or
ii. larger w, implying a smaller l.

For each option, explain **one** potential advantage versus the alternative; carefully include any assumptions.

c i Explain what is meant by a segregated (or split-use) cache architecture, and **one** reason such an architecture might be used.

ii In a stored-program or von Neumann architecture, both data *and* instructions are held in memory. Self-modifying programs rely on this feature: instructions are written into memory during execution, then subsequently executed.

Explain, in detail, **one** issue presented by use of a segregated cache architecture in this context **and** how it might be resolved.

▷ **Q238.** The following text represents an extract from the data sheet for an ARM740T, a 32-bit processor designed by ARM, downloaded from www.arm.com:

The ARM740T can incorporate either an 8KB or 4KB general-purpose cache. Both variants are functionally equivalent. The cache:

- *is 4-way set associative,*
- *is write through,*
- *has four words and a a valid flag per line,*
- *uses a random replacement algorithm.*

a With reference to the description, explain

i the motivation for including a cache in the processor design, and
ii what is meant by a general-purpose cache (versus some form of special-purpose alternative).

b i Effectiveness of the cache depends on

i. spatial locality, and
ii. temporal locality

in address streams produced by programs during execution. Explain **both** concepts, using short C programs to illustrate your answers.

ii Explain potential advantages **and** disadvantages of selecting the ARM740T model with an 8kB rather than 4kB cache.

iii The cache is described as being 4-way set-associative. Explain what this means, and what the motivation for such a design versus a direct-mapped alternative is.

c Consider the ARM740T model with an 8kB cache, and assume for simplicity that by "four words per line" the description means each cache line holds four bytes. Compute the number of bits used for

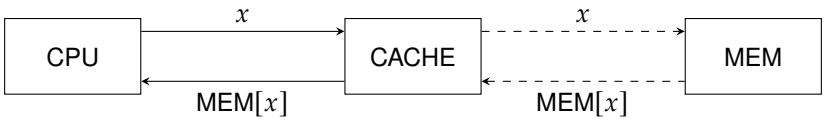
i the (sub)word address,
ii the line address,
iii the set number, and
iv the tag

while dealing with a load operation from address x.

d The cache is described as write-through. This refers to how it deals with store operations: the idea is that to ensure consistency between the cache and main memory, data is stored into *both*, i.e., written *through* the cache into main memory.

Explain **one** alternative scheme for dealing with stores, clearly listing any advantages **and** disadvantages.

▷ **Q239.** Consider a direct-mapped cache placed between a 32-bit RISC processor and a 1GB main memory:



The cache can hold at most 1kB of data, organised as $l = 128$ lines each containing $w = 8$ separate 8-bit words.

a One option for implementing storage within the cache is to use 6T SRAM cells. Draw a transistor-level **circuit diagram** that describes such a cell.

b Write an algorithm that details how the cache satisfies a load, performed by the processor, from address x.

c i Calculate the number of bits required to specify

i. the word address,
ii. the line address, and
iii. the tag.

Finally, calculate the number of SRAM cells required to store the data and meta-data (i.e., the valid flags and tags).

ii Imagine a new SRAM cell is available that can operate in a low-power mode. When a control-signal $pwr = 1$, a given cell operates as normal; when $pwr = 0$ the cell is in low-power mode but cannot retain any content.

Assuming such cells are used to store the cache data and meta-data, describe **two** distinct policies to control them so the overall power consumption is reduced.

Part XV: Performance measurement

▷ **Q240.** a CPI and MIPS are both measures of processor performance. Explain how they are calculated and their advantages and disadvantages.

b After some experimentation, someone shows that programs written for a given processor have the following instruction mix:

	Frequency	Cycles
Arithmetic	50%	1
Branch	20%	2
Load	20%	5
Store	10%	3

The addition of a new addressing mode and a better compiler means there are less accesses to memory required. The instruction mix after this change is as follows:

	Frequency	Cycles
Arithmetic	55%	1
Branch	25%	2
Load	10%	5
Store	10%	3

Calculate the overall CPI for programs that run on the processor before **and** after the change, and the speed-up that results from the change.

▷ **Q241.** Outline the difficulties faced when writing benchmark software that is to collect information about processor performance. Include at least one method that could be used to artificially increase the performance, i.e., "cheat" a benchmark.

Part XVI: Techniques for efficient implementation

▷ **Q242.** The following questions ask you to write C functions that operate on unsigned 16-bit integer values represented by the `uint16_t` type. Each function should be fairly short, i.e., roughly 10 to 15 lines or fewer; a more efficient function will gain more marks. For each answer, briefly state any advantages and disadvantages of your approach in comparison to alternatives.

a Without using the operations <, =, or > or any branch or conditional operations, write a C function that replicates the behaviour of

```
uint16_t choose( bool c, uint16_t x,  uint16_t y ) {
  if( c ) {
    return x;
  }
  else {
    return y;
  }
}
```

where you can assume `c` is used as a Boolean variable, i.e., it always equals either 0 or 1.

b Without using the multiply operation, write a C function to implement the multiplication $x \cdot 15$ for an unsigned 16-bit integer x.

c The population count or Hamming weight of x, denoted by $\mathrm{HW}(x)$, is the number of bits in the binary expansion of x that equal one. Write a C function to compute $\mathrm{HW}(x)$ for an unsigned 16-bit integer x.

d Pick one of your solutions from the previous question and show how to accelerate it using the concept of pre-computation; you should write one C function to perform any pre-computation, and one C function that uses the pre-computed data to implement the original operation.

▷ **Q243.** Assume that unsigned 16-bit and 8-bit integer values are represented by the `uint16_t` and `uint8_t` types. The C function below performs "packed" division by two on elements of a vector called `A`: it takes each 16-bit element `A[ i ]`, unpacks it into two 8-bit values `l` and `h`, divides both by two, and packs them back into the 16-bit result:

```
void packed_div2( uint16_t* A ) {
  for( int i = 0; i < 3; i++ ) {
    uint8_t l = ( x[ i ]      ) & 0xFF;
    uint8_t h = ( x[ i ] >> 8 ) & 0xFF;

    l = l / 2;
    h = h / 2;

    A[ i ] = ( ( uint16_t )( l )      ) |
             ( ( uint16_t )( h ) << 8 ) ;
  }
}
```

Several optimisation techniques could make this function execute faster:

a Offline pre-computation.

b Specialisation.

c Parallelism.

d Program restructuring.

Show how the function could be re-written (using pseudo-code where appropriate) utilising each of these techniques.

▷ **Q244.** Consider the following C fragment which adds together two `n`-element vectors `B` and `C` to produce a result `A` (where each element is a 32-bit integer):

```
for( int i = 0; i < n; i++ ) {
  A[ i ] = B[ i ] + C[ i ];
}
```

a Describe **two** reasons why the use of loop unrolling might increase performance of the fragment, and **one** reason it might decrease performance.

b Sometimes loop unrolling requires a loop prelude and/or epilogue. Using an example, describe what these are and why they might be required.