Part I : Computer Organization  (25 marks)

(1) A multiprogramming operating system uses an apportioning scheme and divides 60 MB of available memory into four partitions of 10 MB, 12 MB, 18 MB, and 20 MB. The first program to be run needs 17 MB and occupies the third partition. The second program needs 8 MB and occupies the first partition. The third program needs 10.5 MB and occupies the second partition. Finally, the fourth program needs 20 MB and occupies the fourth partition. What is the total memory used? What is the total memory wasted? What percentage of memory is wasted?

[3 marks]

(2) A multiprogramming operating system uses paging. The available memory is 60 MB divided into 15 pages, each of 4 MB. The first program needs 13 MB. The second program needs 12 MB. The third program needs 27 MB.

a) How many pages are used by the first program?
b) How many pages are used by the second program?
c) How many pages are used by the third program?
d) How many pages are unused?
e) What is the total memory wasted?
f) What percentage of memory is wasted?

[3 marks]

(3) Write an assembly language program that reads in a list of integers, one at a time, and, when it reads in 0, stops and prints out the sum of positive integers that it read in. For example, if 3, -2, -1, 5, 0 are read in, then 8 is output.

Be sure to show all necessary .DATA pseudo-ops, and to use the instruction set language used in lectures (or that defined on page 204 of the course textbook).

[10 marks]

(4) The next three problems refer to the assembly language program attached below.

a) Explain what the program does.

b) Give the symbol table that will be produced during the first pass of the assembler when translating the program to machine code (recall that the symbol table records the physical addresses of the labels used in the program). Assume that the program starts in memory cell zero.
c) Give the sequence of machine code instructions and their locations in memory that will be produced during the second pass of the assembler. Use the opcodes given in lectures, corresponding to the instructions here, and assume address fields of 12 bits.

[9 marks]

```
LOOP:  load   SUM
       add    K
       store  SUM
       incr   K
       load   N
       compare K
       jumplt LOOP
       jumpeq LOOP
       output SUM
       halt

N:     .data  100
SUM:   .data  0
K:      .data  1
```

Part II Boolean Algebra and Digital Logic (25 Marks)

Professor Pemberton Partridge – Pete to his friends – has asked for your help. Although a world-renowned astronomer, he found himself hopelessly at sea when it came to designing the control systems for the revolutionary new thought-amplifying telescope to be installed at Mt. Palomar, and intended to detect signs of mental activity in far-flung regions of the universe. Here’s a transcript of your meeting with him.

“Swithins,” he intoned (he calls all his assistants Swithins, irrespective of their real names) “you must help. What we need here is a little gadget that can detect the rotational position of the Apparatus.” He never called the telescope a telescope (or a spade a spade, though that was largely because he never did any gardening), and he always pronounced apparatus with a capital A, when referring to the telescope.

“I was going to use a microprocessor, but the Pentagonal Ones who supply our finance are terrified of viruses and they want to avoid computers wherever possible. So I’ve been toying with the idea of a positional detector based on transmitted light. The idea, Swithins, is that there’d be a horizontal glass disc attached at the base of the Apparatus, and as the Apparatus rotates, the disc would rotate too. Now, the disc would be divided into a number of concentric rings, and each ring would be divided into a number of sectors. Each sector would either be painted black, or left transparent.

A number of photodetectors would be arranged radially out from the centre of the disc, and on the opposite side of the disc would be a light source. If a black-painted sector occurred between the light source and the photodetector, the photodetector will output a logic 0. Conversely, if a transparent sector occurred between the light source and the photodetector, the photodetector will output a logic 1.

“Thus,” he droned on, “it would be possible, using the outputs from the photodetectors to assemble a binary number, representing the rotational position of the disc, and therefore the whole Apparatus. We would decide, quite arbitrarily, that the innermost photodetector would produce the least significant bit of the number, and the outermost photodetector would produce the most significant bit. The trouble, as I see it, is that the base of the Apparatus is, not to put too fine a point on it,” (he went in for clichés) “inclined to be a bit sensitive mechanically, what with all the movement and the breadcrumbs.”

“Breadcrumbs, Professor?” you interjected, respectfully.

“Yes, breadcrumbs Swithins, breadcrumbs. I often listen in to the alien thought patterns while I have my morning tea.” He seemed completely to the telescope had not yet been built. “And it’s possible that breadcrumbs may lodge in the mechanical bits and displace the detectors somewhat. Or some of my younger colleagues who aren’t too careful about where they put their feet,” here he glanced meaningfully at your feet, “could easily displace the mechanism slightly. In such circumstances, it would be a Good Thing,” he leaned forward in his carven chair to emphasise this point “if the result of any errors in positioning the Apparatus were minimised. It would prevent an inexperienced young observer such as yourself from publishing a report containing wildly inaccurate positional information.”

1 There’s a diagram of this arrangement later in the assignment
Everyone knew that that had happened to the Professor in an earlier aeon, and that he had lived for forty years in abject fear of a repetition.

“I think I see,” you said, gravely. “If the number encoded were, say 0111, representing position 7, and the photodetector for the most significant bit were slightly misaligned then the number could become 1111, representing position 15.”

“Exactly,” quoth the Professor, beaming. “You understand perfectly. Now, I believe there’s something called a reflective code that’s been invented to cope with just this sort of situation. Don’t know anything about it, but can you find out, and make up a small test rig to demonstrate how effective a reflective code would be? Thank you so much.” And he leapt up from his chair, and stalked off, communing with what he imagined to be alien thought patterns but were in fact the pattern on the wallpaper.

Well, it turned out that, as so often - and so infuriatingly - Pemberton Partridge was right. A reflective (or Gray\(^2\)) code was just what the situation called for. It’s intended to ensure that the combination of bits used to encode a number only differs in one bit from its neighbours. That is, the bit combination for \(n\) differs by only one bit from the bit combination for \(n-1\), and it also differs by only one bit from the bit combination for \(n+1\). Consider a single bit number. This can have two values

\[
\begin{align*}
0 & \text{ represents } \ 0_{10} \\
1 & \text{ represents } \ 1_{10}
\end{align*}
\]

If we increase the number of bits in the number to two, then, using the conventional numeric coding we get

\[
\begin{align*}
00 & \text{ represents } \ 0_{10} \\
01 & \text{ represents } \ 1_{10} \\
10 & \text{ represents } \ 2_{10} \\
11 & \text{ represents } \ 3_{10}
\end{align*}
\]

To count from 110 to 210, we have to change both bits in the binary number so if, say, the photodetector reading the top bit of the code for 110 (that is, 01) is misplaced, it may generate a 1 instead of a 0, and the resulting number becomes the code for 310, (that is, 11) . Similarly, in a four-bit code, if an error occurs in the 2\(^2\) bit of a conventional four-bit binary representation for 6\(_{10}\) (0110), it can turn into the binary representation for 2\(_{10}\).

However, if we build up our binary numbers in a different way, starting with 0 and 1 as usual, so that

\[
\begin{align*}
0 & \text{ represents } \ 0_{10} \\
1 & \text{ represents } \ 1_{10}
\end{align*}
\]

and add another bit to make a two-bit representation where each number differs by only one bit from its neighbours, then we get:

\[
\begin{align*}
00 & \text{ represents } \ 0_{10} \\
01 & \text{ represents } \ 1_{10} \\
11 & \text{ represents } \ 2_{10} \text{ (different from the conventional binary code)} \\
10 & \text{ represents } \ 3_{10} \text{ (different from the conventional binary code)}
\end{align*}
\]

Let’s see what happens with three bit numbers:

\[
\begin{align*}
000 & \text{ represents } \ 0_{10} \\
001 & \text{ represents } \ 1_{10} \\
011 & \text{ represents } \ 2_{10} \\
010 & \text{ represents } \ 3_{10} \\
110 & \text{ represents } \ 4_{10} \\
111 & \text{ represents } \ 5_{10} \\
101 & \text{ represents } \ 6_{10} \\
100 & \text{ represents } \ 7_{10}
\end{align*}
\]

There’s a pattern here, though it may not be particularly clear yet. Each time the size of our numbers increase by one bit, we double the number of numbers that can be encoded, because all the values in the previous code can be preceded by a 1 in the new bit position, and all the values in the previous code can also be preceded by a 0 in the new bit position. But the trick is that we write out the previous code, preceded by a 0, and when we come to write out the previous code again preceded by a 1, we write it in reverse order. The underline in the list of numbers above represents a mirror, in which the two-bit code is reflected. You can extend this idea to four-bit (and more) numbers. Try it:

- Copy out the column of three-bit numbers shown above.
- Then write a second column of three-bit numbers below the first, but put the entries in the column in reverse order.
- Now put a 0 in front of the first eight entries, and a 1 in front of the second eight entries.

\(^2\) The codes were first patented by Frank Gray, a Bell Labs researcher, in 1953.
You’ve just written out a four-bit Gray code for the numbers 0 to 15. Each number differs from its neighbour in only one bit position. You should be able to see why it’s called a reflective code.

Prof. Partridge’s idea is to encode the position disc so that a reflective code is shown on its various sectors, thus:

The lines separating the sectors, and the zeros and ones shown in the diagram would not, of course, be present; they are only shown to make the diagram clearer. And the centre circle is not used for the coding. As the side view below (the blacked-out sectors have been omitted for clarity) shows, the telescope is mounted on an axle passing through the centre of the disc.

Your job is to
(a) design a digital circuit that, when fed the output of three photodetectors, outputs a three-bit number with the appropriate (conventional) binary representation of the position of the disc. That is, it converts a 3-bit Gray-coded number to a 3-bit number in conventional binary representation.  

(b) design a decoder that takes the output from the three photodetectors and makes one signal out of eight TRUE, according to the position of the disc. This circuit should work directly from the output of the photodetectors. It should not decode the conventional binary number produced by the circuit in part (a)  

(c) calculate the number of bits that would be necessary in the real system (remember that the three bit version you’ve been designing is only a small test rig) to encoded the telescope’s position to a precision equal to or better than one second of arc. The rotational position of the telescope does not need to be represented as degrees, minutes and seconds of arc, but can be represented as a continuous series of integers from 0 up to whatever.  

You do not need to design the circuitry for the full sized system.

---

3 A circle is divided into 360 degrees. A degree is divided into 60 minutes. A minute is divided into 60 seconds.