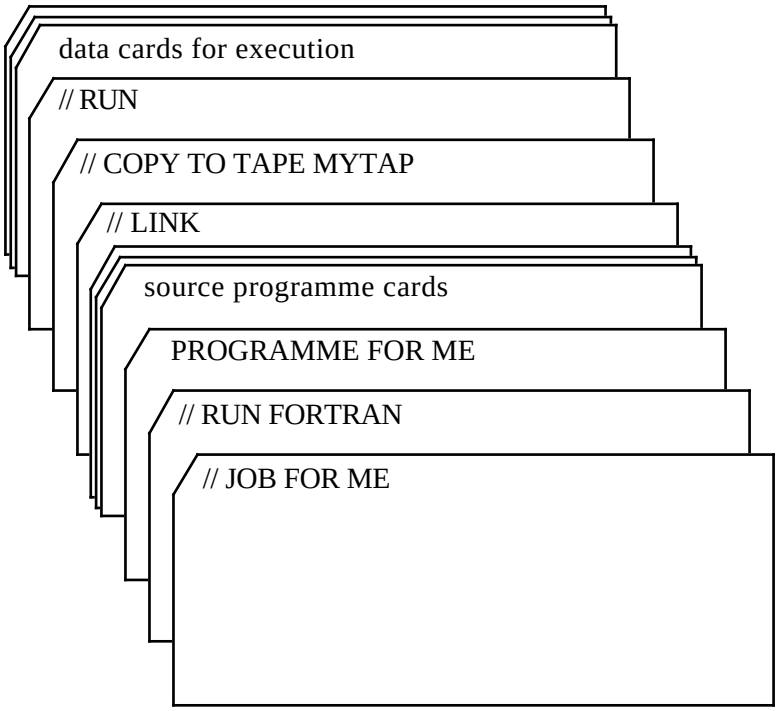
A MONITOR SYSTEM

WHAT IT DOES.

The object of the exercise is to take all the information needed to define and run a computer job – as in the sample card deck shown below – and have the computer complete the job with as little human intervention as possible.



HOW IT WORKS.

The only way to make a computer work for you is to give it a programme to run; so we need a programme to read the card deck, to find the operating instructions, and to act accordingly. This programme is the *monitor system*.

OBSERVE that we shall come across another sort of monitor later in the course : there is absolutely no connection between the two ideas. In a conversation about computing jargon some time ago, someone brought up the double meaning of "monitor" as a bad example. It was some time before we realised that she had never heard of the other operating system sort of monitor – she meant monitor system, and monitor meaning terminal – and both of us had missed the performance monitor, useful in measuring the work done by a system.

The monitor has to be ready to take over as soon as a programme stops – so some of it at least must live permanently in memory. We remarked earlier that this was the *resident monitor*. It has to be able to find and load common programmes (like the compilers, linker, tape utilities etc.) – so it must have some sort of secondary memory in which these programmes can be kept. Magnetic tapes will do, but discs are a lot better.

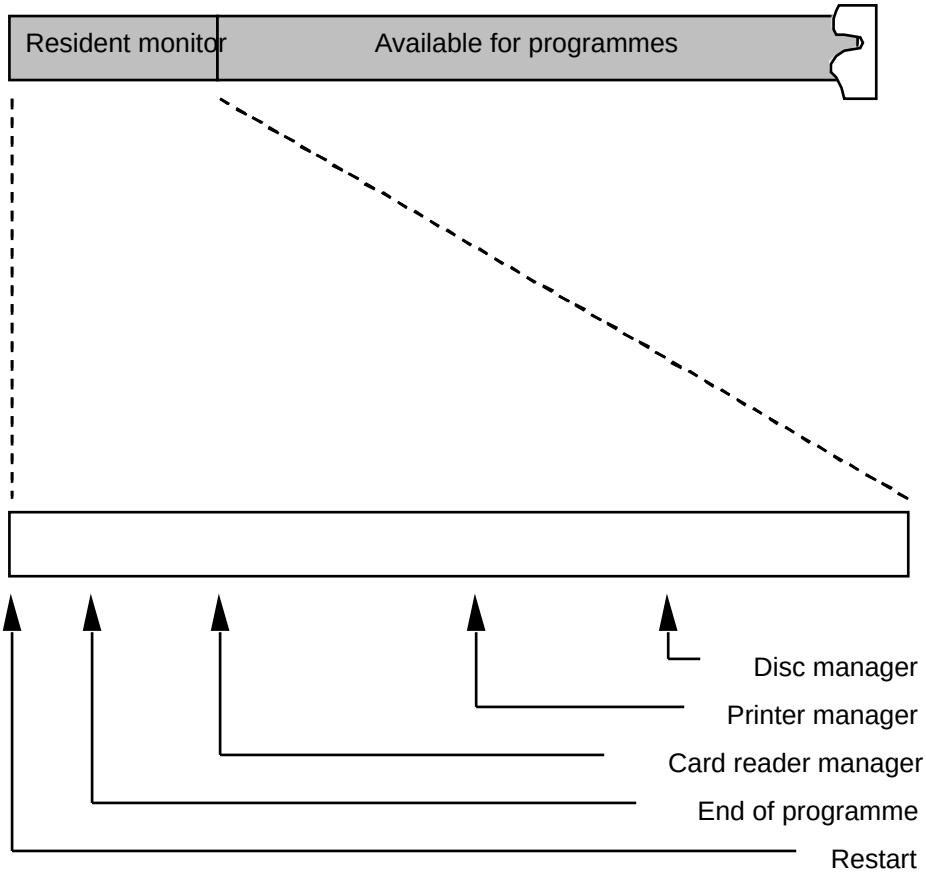
What constitutes the resident monitor ? There's no cut-and-dried answer to that question, as the decision is a matter of compromise. We want to use as little memory as possible, because there isn't much available. (A monitor system would run in a computer with eight kilobytes of memory; that was small, but not surprisingly small.) At the same time, the resident monitor provides useful services, and the more of those we want the more memory it will require. One could argue something like this :

1 : We *must* have something which can pick up the processing when a programme stops.
2 : Whatever that does, it must be able to get more monitor system, or other programmes, from the disc, so there must be procedures to read disc files.

That would be sufficient if we really wanted to minimise the memory demand, as there's enough to make sure that the resident monitor can get help from the disc whenever anything happens. But we also want to speed things up if we can, so :

3 : If we put just enough code in there to drive the card reader and the printer as well, and to load and start a programme, we could often handle the transition between programmes without reading any monitor system from the disc, which would save some time.

That is pushing the bounds of the amount of memory we'd want to spend; typically, code for that much performance, with some other useful data, would fit into about one kilobyte. (That's still true. Why do operating systems now occupy megabytes ?) We can imagine the result something like this :



Now we've packed quite a lot of useful software into our resident monitor, but that leaves us with another question : how do the programmes which want to use the useful software know where to find it ? This introduces a part of the operating system which is surprisingly ignored : the set of conventions which must be known by people who want, or software which wants, to use the system. This information has changed in detail through the years, but is always with us, and we shall return to it later.

WHAT HAPPENS WHEN YOU RUN A JOB, with questions.

The ACTIONS in the table below show what happens in the computer, stage by stage, as a simple job is executed In each case, the running programme – monitor, compiler, etc. – is identified, with a brief description of what it does.

ACTION	REMARK
The monitor reads the identification card, saves the name to identify printed material.	*Why not do the accounts too ?*
The monitor reads the "// RUN FORTRAN" card, loads the compiler, transfers control to it.	*Why do we need the "// " ?*
The compiler reads the programme name, then the source programme, then finishes, transferring control back to the resident monitor.	*Why couldn't the programme name go on the RUN FORTRAN card ?*
The monitor reads the "// LINK" card, loads the linker, transfers control to it.	*Why do we need the explicit instruction ? Under what circumstances wouldn't we want to link the code ?* *Why not run a separate programme called LINK ?*

The linker runs, reads any system subroutines from disc, leaves the runnable programme in memory, transfers control back to the resident monitor.	
The monitor reads the "// COPY TO TAPE MYTAP" card; requests the operator to load tape MYTAP; waits until the tape is loaded; copies the programme to tape MYTAP.	*The file will be named FORME on the tape. We still can't eliminate the operator.*
The monitor reads the "// RUN" card; transfers control to the resident programme.	*How does it know where to find the programme ?*
The programme runs, reading data cards, probably printing, reading and writing disc or tape files, and finally transferring control to the resident monitor.	*How does it know where to find the monitor ?*
The monitor waits until more cards are supplied – then, presumably, starts a new job for someone else.	

SOME ANSWERS.

Most of the "REMARKS" in the table were questions. We've put them in that form to point out several other features of the system, and to show how the features arise from the monitor system's job. Here are some brief answers to the questions. They are all taken up again somewhere later in the notes, but you might like to explore their implications now.

Why not do the accounts too ? Because there is nowhere safe to put them. There's little point in keeping a record of bills which people have to pay if the people in question can change them at will !

Why do we need the "// " ? As a safety feature, so that there's less chance of accidentally executing an "instruction" which is really an ordinary source programme or data card.

Why couldn't the programme name go on the RUN FORTRAN card ? Well, it could if that's what the system designer wanted, but it would mean a more complicated system because there would have to be provision for getting the programme name from the monitor system which read it to the Fortran compiler which would use it. It isn't hard to do if you want to (one way is to save the last system instruction in a buffer somewhere), but every additional special case like this means a little more memory which has to be set aside, and there is absolutely no memory to waste.

Why do we need the explicit instruction ? Because we might not wish to link the code immediately; we might wish to save it unlinked so that we can use it again without recompiling.

Why not run a programme called LINK ? In practice, the system might have done exactly that, but a boundary was perceived between "system code" and "programmes". Generally, the early systems were thought of as single entities – which at least sometimes made communication between their parts fairly easy – and it was a long time before the virtues of more modular systems were understood.

How does it know where to find the programme ? This is another example of a convention which must be established as part of the operating system; the required memory address must be laid down by the operating system and observed by the people who write the linker which makes this possible. Perhaps the first instruction

of the programme is always placed at some predetermined address – or, more flexibly, every programme must be accompanied by some information which identifies the starting address to be used.

How does it know where to find the monitor ? As in the previous answer, there must be some convention laid down by the operating system, this time about the address for return from a programme. This is the function of the "End of programme" label on the diagram earlier in this chapter.

QUESTIONS.

Are there any defects in the system ?

How can we do better ?

Is it reasonable to consider the programme to be a subroutine of the monitor system ?

Is it appropriate to regard systems such as CP/M or MS-DOS as monitor systems ? What about a Macintosh system ? (Try to separate the interface from the underlying system software.)

Define a monitor system.
