MULTI-PROCESSING SYSTEMS

Professor B. W. Arden

Computing Center,
University of Michigan,
North University Building,
Ann Arbor,
Michigan, 48104.

Abstract:

This paper outlines the computing system available at the University of Michigan concentrating primarily on the M.T.S. hardware and software structure and the measurement of system performance. In this context the computer science courses offered at Michigan are described in some detail.

Rapporteurs:

Mrs. N. Newman
Mr. J. S. Clowes

Introduction

Professor Arden began by saying that he felt it would be meaningless to talk about the teaching of systems design in Michigan without describing briefly the environment in which the teaching took place. He therefore proposed to outline the computing system available at Michigan, and to go on to talk about the educational process, which was closely connected with the system environment.

---Historical Development

The system in use at Michigan is commonly known by the name of one of its sub-systems, the Michigan Terminal System. The Michigan System involves the use of a multi-processing supervisor controlling a number of sub-systems, and currently operates on a two-processor I.B.M. 360/67 computer. The use of multi-processing came about for two main reasons; firstly at the time when it was introduced at Michigan it seemed likely that the development of single-processor systems was nearing a technological barrier, and secondly there are considerable advantages in centralization, notably increased reliability, the sharing of data and programs, possibilities of load adjustment and improvements in facilities for system development.

Up to 1964 Michigan had had a series of I.B.M. machines, a 650, a 704 and then a 7090, all of which were sequential buffered I/O systems of increasing complexity. In that year Michigan's users gave the University a mandate to install a multi-processing, multi-programming system with batch and terminal support. After discussions with manufacturers it was decided to buy an unsupported I.B.M. machine, the 360/66M, which eventually became the Model 67. It was originally hoped that a suitable system for meeting Michigan's commitments would be available for the 360. When it became apparent that this would not be the case, work was started on the development of a terminal system derived from a small program written at Lincoln Laboratories for multi-programming of I/O units and graphical I/O devices. This system, not originally intended for the 360/67, was developed for this purpose and in mid-1966 it was extended to include a file handling, command and accounting sub-system for terminal support, the Michigan Terminal System. To begin with this system operated with real memory, and had a maximum capacity of eight to ten terminals. Late in 1966 the change was made to virtual memory and paging drum management was added. This immediately resulted in an improvement to 25-30 terminals.

---M.T.S. Hardware Structure

The structure is symmetric, in that all the I/O devices, disk drives, etc. can be linked to a single processor by means of configuration switches. A diagrammatic representation of the M.T.S. hardware structure is given in Figure 1.

---M.T.S. Software Structure

The software structure consists of a resident multi-programming supervisor, which controls a system of jobs which are instances (usually multiple) of job programs currently being multi-programmed. Examples of job programs are M.T.S. itself with its file, command and accounting facilities; unit check recovery routines; disk file support routines; terminal support routines; the paging drum processor; the monitoring statistics collection routine; the batch job scheduler and unit operation routines. These programs are normally re-entrant. Currently, the supervisor and job programs occupy segment zero of the virtual address range, whereas user programs occupy segment one. Scheduling is basically sequential with two exceptions; jobs returning from I/O operations are given preference in the CPU queue, and the jobs with an excessive page requirement receive special treatment, so that, where a job needs, for example, more than 48 pages, its time slice in each cycle is increased and only three such jobs are given a time-slice in each cycle.

A diagrammatic representation of the M.T.S. software structure is given in Figure 2.

DIAGRAMMATIC REPRESENTATION OF M.T.S. SOFTWARE STRUCTURE

Job list table;
This contains an inventory of resident program with lettering requirements etc.

Device List:
This contains an inventory of devices

Saved I/O Queue:
This takes care of the 'pushing down' of I/O requests because e.g. of recovery from errors.

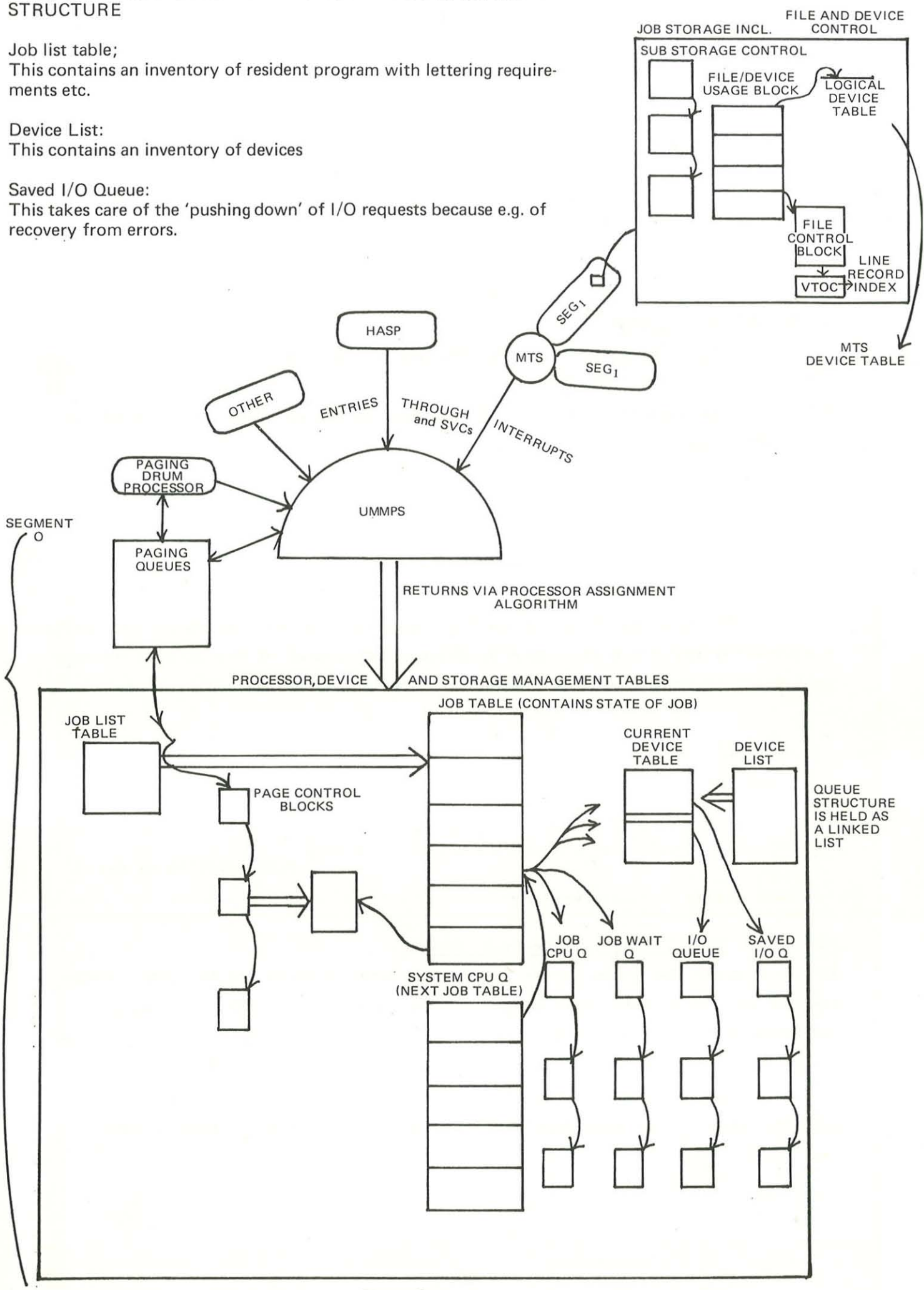

Figure 2

Performance Measurement

Professor Arden believed that it was essential that any complicated system should have facilities for monitoring its performance. Data collected by monitoring a system while it is running enables one to assess the efficiency of different parts of the system and thus to identify areas which require further development. Analysis of such data is also an important and educationally useful student activity.

Two monitors are in use at Michigan collecting information from different levels of activity of the system.

---The Job-level monitor

This monitor provides information about jobs, its software is an integral part of the M.T.S. system and it is always active. Data recorded by this monitor forms the basis for the accounting system. Items recorded for each job include:

CPU time;

Elapsed time;

CPU space-time;

Waiting space-time;

File storage.

The meaning of the terms CPU space-time and waiting space-time is best explained by reference to figure 3, which shows a plot of the virtual storage assigned to some job at each instant of time during its period of activity.

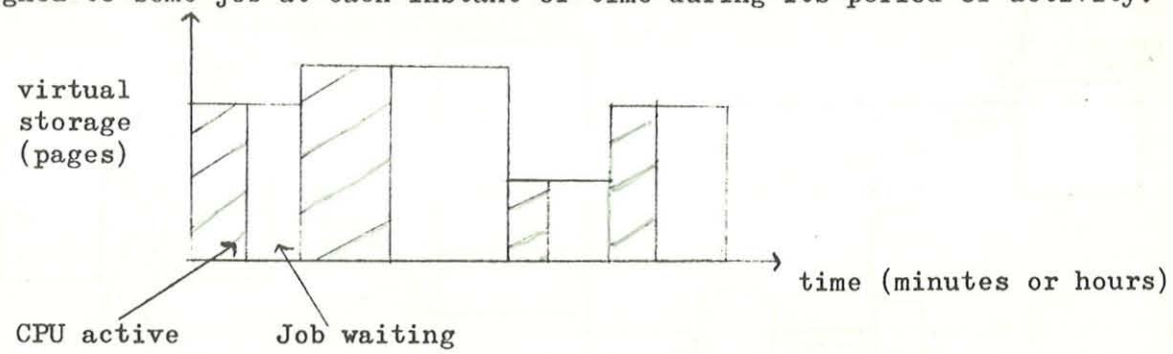

Figure 3. To illustrate space-time measure.

The area under the graph is the total space-time measure for the job. While the job is active there will be periods during which the CPU is actually executing the job, such periods are hatched in the diagram. During the rest of the time the job will be waiting for some reason. The sum of the areas of the hatched portions of the diagram is the CPU space-time measure for the job, the area of the remaining portions is the waiting space-time measure.

---The Event Recording Monitor

The second monitor operates at a lower level in the system and records events, such as, when a job is assigned a CPU, or, when a job joins one of the queues internal to the M.T.S. system. This monitor is implemented as a job program and is activated when required, like any other job program. The data collected is used for fine tuning of the system.

---Examples of the use of monitor information

As an example of an application of the second monitor Professor Arden described its use to record a history of interrupts during a period of 15 hours continuous running of the system. In this time there were approximately 19×10^6 interrupts (including SVC's) evenly distributed between the two CPU's. Analysis of the data yielded the following statistics:

Breakdown of interrupts by type.

External (i.e. timer)	5%
SVC	70%
Program (e.g. overflow)	3%
I/O	22%

(Each I/O interrupt is matched by an SVC which is included in the 70%)

average time per interrupt	775 micro-seconds
average number of interrupts per second	370
average time between interrupts	2.7 milli-seconds
CPU utilization (averaged)	44%

Professor Arden remarked that, since the proportion of timer interrupts was so small and the average time between interrupts was also small, one might be led by these results to consider whether time-slicing hardware was really necessary. Instead, one might compute the time for which a job had been running whenever any interrupt occurred. Such a technique would only be effective for suitable hardware configurations (more than two CPU's would be required) and an appropriate mix of jobs, but this was an example of the way in which performance measurement could suggest developments in system design.

As an example of the use of the job-level monitor for performance analysis, Professor Arden presented a load point analysis of the Michigan System. The data base in this case was the output of the monitor for all jobs run in a period of one month, a total of about 56,000 jobs with a ratio of batch to terminal jobs of 2:1. The hardware configuration was two CPU's and 300 pages of core available to the jobs (the remaining 84

pages were used by the system programs). From the data collected the following means were computed:

Mean:	Batch:	Terminal:
CPU time	0.5 min.	0.5 min.
Holding time (admit-complete)	2 min.	14 min.
Wait space-time (page mins.)	55	208
CPU space-time (page mins.)	18	12
Storage (derived)	~36	~16

These figures were used to obtain the CPU saturation line and the paging line shown in figure 4.

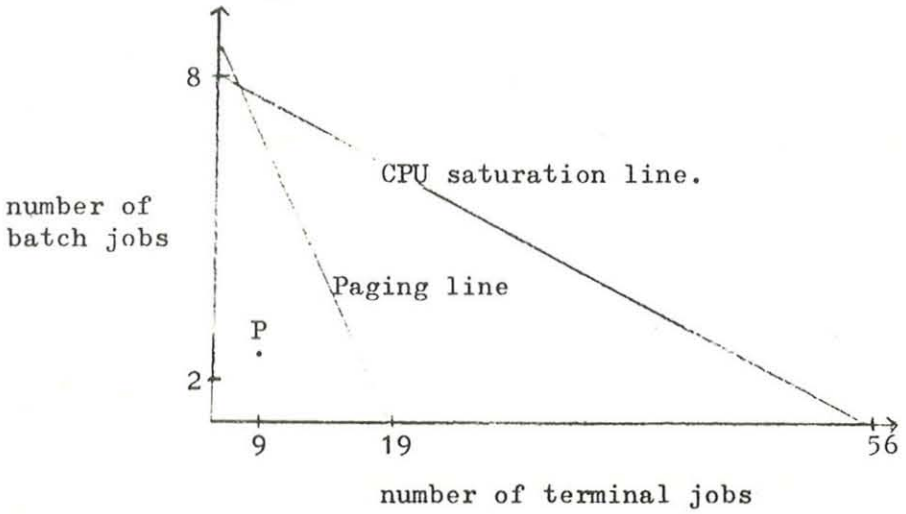

Figure 4. Load point analysis.

In figure 4, the number of terminal jobs is plotted as abscissa and the number of active batch jobs as ordinate. Since an average batch job requires 0.5 mins. of CPU time and is active for two minutes, two CPU's could handle eight batch jobs if there were no terminal jobs present. This gives the intercept of the CPU saturation line on the batch job axis. The intercept on the terminal job axis is computed similarly and hence the CPU saturation line. The paging line is obtained in an analogous manner, remembering that there are 300 pages of core available. Its significance is that no paging is necessary for any combination of average batch and terminal jobs represented by a point below this line.

The average load, obtained by considering the measured load to be distributed uniformly over all days of the month, is approximately two batch and nine terminal jobs. This load, represented by P, lies well below both the paging and CPU saturation lines. In practice the load varies with time and the actual load point makes excursions within some region determined by the loading algorithm.

If the load consisted of the maximum number of 56 terminal jobs approximately 900 pages of store would be required, giving a ratio of virtual to real store of 3:1. Loads corresponding to other points on the CPU saturation line require lower paging ratios. Other experiments indicate that the system can run with such a paging ratio without significant degradation. Thus the 44% CPU utilization observed during the previously described experiment suggests that the system was underloaded.

---Bench Mark Tests

Another method of examining system performance is by bench mark runs. Professor Arden presented the results of eight test runs involving 15 standard Fortran programs using different compilers and different system configurations. The results of these tests and specifications of the system configurations used are given in Table 1 together with Professor Arden's comments. The threshold and decrement referred to in runs five and six are scheduling algorithm parameters. If the store requirement of a job exceeds the threshold then it becomes a candidate for consideration as a 'privileged' job receiving larger time slices; as additional larger page-demand jobs are encountered, the threshold is reduced by the decrement.

Computer Science Education at Michigan

The University of Michigan has two degree programmes involving computer science. One of these, entitled Computer, Information and Control Engineering, is run by an engineering department, the other is the responsibility of Professor Arden's own Department of Computer and Communication Sciences (CCS).

As its name implies, the interests of the CCS faculty are not confined to what may strictly be called computer science. Principle subject areas are:

1. Computing (primarily systems design).
2. Automata Theory.
3. Linguistics (formal and natural).
4. Natural Systems (the main interest here is in modelling, e.g. there are physiologists concerned with cell models).

Students intending to major in CCS take courses in all these areas as a 'core' syllabus; however, Professor Arden described only the 'computing' courses. Although some of the courses offered by the CCS department include instruction in numerical methods, formal numerical analysis courses are given by the Mathematics Department.

TABLE I

Results of Fortran bench Mark run in M.T.S.

	Description of run	CPU Time	Elapsed Time	Drum Reads	Cost	Total Time	% Billable CPU Time.
1.	1-15 H, 4 CB, 2 CPU	1339.659	4269.424	9026	146.60	783	85.6
2.	1-10 G, 11-15 H 6 CB, 2 CPU	1189.989	3279.848	814	118.97	666	89.3
3.	1-8, 10 G 6 CB, 2 CPU	1123.349	3238.326	1104	116.97	629	89.3
4.	1-8, 10 G 6 CB, 1 CPU	1094.129	6152.908	1044	114.28	1180	92.7
5.	1-8, 10 G 3 CB, 1 CPU Threshold = 48, Decrement = 16.	1085.572	6577.185	25358	114.65	1319	82.3
6.	1-8, 10 G 3 CB, 1 CPU Threshold = 16, Decrement = 0.	1065.555	6183.435	12088	111.92	1190	89.5
7.	1,2,3,5,6,7,10 in Watfor, rest H, 4 CB, 1 CPU, 3 streams	920.346	2131.782	1690	100.60	948	96.4
8.	1,2,3,5,6,7,10 in Watfor; 4,8 in G; 9,11-15 in H; 2 CPU, 6 CB, 9 streams.	996.365	2570.124	1920	107.29	540	92.2

Comments:

1. Tests 1 to 6 were run with two input streams. H stands for the FORTRAN H compiler, G for FORTRAN G.

2. Comparison of 3 and 4 indicates an interference between CPU's of at most 2.5%.

3. Since all jobs were small by current standards (<100k), the benefits of pooling storage between multiple CPU's did not appear strongly.

4. Comparison of 4 and 6 shows that the size of main storage is not a critical factor, since all jobs are small.

5. Comparison of 5 and 6 shows the critical importance of the parameters that control the paging algorithm.

6. Comparison of 4 and 6 shows that neither the cost nor the total time needs to increase when paging increases. Paging can be completely overlapped, even for this small set of jobs.

7. Tests 1 through 4 were run with 2 drums, the rest with 1 drum.

---Courses in Computer Science offered by CCS

The following two courses CCS 273 and CCS 274 are intended to provide an introduction to computing for second year undergraduates.

CCS 273. Elementary Numerical Computing 3 hours per week.

A course in calculus is a pre-requisite for this. Students are taught to program in FORTRAN and some elementary numerical methods. Exercises involve writing programs to solve numerical problems.

CCS 274. Elementary Computer Methods 3 hours per week.

A knowledge of mathematics is not a pre-requisite for this course which is primarily a service course for students from areas where formal mathematics is not heavily used. At least one procedure-oriented language (usually FORTRAN) is taught and through this students are introduced to algorithms, programs, subroutines, various data structures and some elementary statistical and symbol manipulation techniques. Exercises run on the computer involve searching, sorting and file handling. A special purpose language usually SNOBOL) is introduced and students learn to use terminal facilities through some language such as PIL, BASIC or SPC (a subset of PL/1).

The two courses CCS 473 and CCS 475, are designed for senior undergraduates or first year graduate students.

CCS 473. Introduction to Digital Computing 3 hours per week.

This course serves both as an introductory course to the computing area and also as a terminal course for students interested only in specific applications. Some mathematics is assumed (first year college calculus) but no computing.

Students learn at least three high-level languages. Assembly language is introduced but students are not required to do programs in this language although they are examined on the material presented.

A major portion of the lecture time is devoted to numerical techniques. Beginning with a consideration of round-off errors and the effect of approximate arithmetic operations, methods for function approximation are introduced and this leads on to quadrature and integration of differential equations. This last topic is of particular importance to students interested in modelling natural systems.

The remainder of the lecture material is devoted to non-numerical aspects. Topics covered here include symbol manipulation, tree structures and sentence parsing. Recent problems include parsing, line justification, maze traversing and flow chart production.

The practical work of the course consists of four or five problems, at least two of them are numerical and at least one non-numerical.

CCS 475 Digital Computers and Computation 3 hours per week.

This course is intended only for CCS students, prerequisites are: course 274 or 273 and one year of college calculus. The course has two main objectives. One is to broaden the student's knowledge in the computer science area, the other is to draw together material presented in more formal CCS courses (Automata Theory, Linguistics etc.) and relate it to practical aspects of computing.

The first purpose is accomplished by having the students implement a variety of algorithms using several different programming languages. These exercises are intended to illustrate the following topics:

Recursive formulation
Logical operations
String processing
Symbol manipulation
Simulation of a machine
Function approximation
Integration of differential equations

The second objective is achieved mainly through appropriate choice of example problems.

Examples

1. In the lecture course a simple minimal machine, called a k-machine, is introduced. This is equivalent to a Wang machine. Students are required to simulate this and to write and run programs for it.
2. The relationship between string manipulation algorithms and Markov normal algorithms is covered. Students are required to do an exercise in theorem proving.

The two courses CCS 573 and CCS 673, are for fifth and sixth year postgraduate students. Students taking these courses usually intend to take employment in the computer science field and are seeking experience of system

programming. These courses occupy about one third of a student's time.

CCS 573 Automatic Programming

Students taking this course must previously have taken either course 473 or 475. The object of the course is to introduce students to systems program construction as distinct from application programming. As course work they are required to write and develop typical systems program components in assembly language. The content of the course varies as equipment changes and techniques evolve; the current course includes the following topics:

1. Machine Organization and Assembly Language.
   - Functional machine description (System 360);
   - Data types and storage representations;
   - Instruction types and assembly format;

   and Assembler Language.
2. Data and Storage Structures.
   - Distinction between data and storage structures;
   - Standard compiler structures;
   - Accessing techniques;
   - Allocation time;
   - Access tables;
   - Types of references;
   - Storage management;

   and Comparison of list processing languages.
3. Compiler techniques.
   - General word problem;
   - Trees and stacks as natural elements;
   - Parsing and roll back;
   - Functional compiler structure;
   - Formal definitions;
   - Language types;
   - Specialized grammars;
   - Grammar transformation;

   and Optimization algorithms.
4. Internal M.T.S.

Problems set to students require then to write macros and, for example, the parsing part of a compiler and a storage management routine.

CCS 673 Advanced System Programming

This is effectively a continuation of the previous course into the hardware dependent level and covers such topics as, dynamic storage allocation and relocation, interaction between central and peripheral units, design of system debugging and measurement tools and the design of new sub-systems capable of operating under or in parallel with the current system. Additionally, the course is intended to give students experience in functioning as part of a group in the specification and implementation of a substantial sub-system, including not only the programming aspect but also the management of the group itself.

These two objectives are achieved by getting the class to implement a group project. In order to keep the task of supervising the group within manageable proportions the class is limited to about 20 students. Some formal lectures on relevant parts of the system are given during the first two or three weeks of the term and during this time the students select their project. Thereafter, the function of the responsible faculty member is to act as advisor to the class, and to ensure that continuous progress is made with the project and that group objectives remain reasonable.

The students themselves choose the subject of their project. Usually, the staff suggest two or three possible subjects but the students may suggest other subjects as they wish. To date, CCS 673 has been presented twice and the projects chosen were MOMS (Michigan On-line Mathematical System) and BASIC II. MOMS is a display driven graphical mathematics system similar to the Culler-Fried system and BASIC II is the BASIC terminal system. BASIC II was implemented and debugged under M.T.S. but the final product runs external and in parallel with M.T.S.

When the project has been chosen the class draw up a precise system specification. After this specification has received the approval of the staff the class partitions the project into sub-projects with specification of interfaces and common internal tables. The class is then broken into sub-groups and each sub-group assigned a sub-project. The projects mentioned above were broken down into components as follows:

MOMS

Numerical routines
Command scanning
Sub-routine definition

Light pen processing

Display buffer pushdown and management

Initialization

BASIC II

Lexical scan to postfix

Symbol table

Command Language

File routines

Interface to system

Numerical routines

Compiler

System de-bugging

User de-bugging

In addition to the groups assigned to specific sub-projects there is also a group responsible for the integration of these components into a complete system. A work-book is maintained in which the current status, and specification, of each sub-project is recorded. Usually, the integration group is responsible for the accuracy of this book.

At the end of the course each group must submit full documentation and current source decks for its components. Each student must also submit answers to a set of questions which ensure that he understands not only his own component but also the total system.