Input/Output Macros

The subject of this lecture is the Input/Output system of the IBM 370 series.

> To quote the textbook "Input/output on the 370–series computers is significantly complex."

The reason for this complexity is a desire to achieve acceptable performance from the computing system, especially when it is running multiple jobs.

Basically there are two forms of Input/Output processing available for use in the design of a computer.

1. The Central Processing Unit can manage all input/output operations.
   This has the trouble of idling the CPU during long I/O operations.
2. The I/O can be separately managed by independent hardware that is directed by the CPU and communicates with the CPU. IBM calls this an **"I/O Channel"**.
   This structure frees the CPU to do other tasks while the I/O is in process.

Those who have taken computer architecture will remember that:

1. The I/O Channel is a special form of Direct Memory Access I/O device, which is a device that interacts directly with the computer's main memory.
2. There are several classes of I/O channels: some for faster devices (such as disks), and some for slower devices (such as keyboard input).

The Input/Output Process

At the most fundamental level, all input and output is a cooperative process between the Central Processing Unit and the various I/O Channels of the computer.

Input:

1. The CPU signals the I/O Channel with a request for input.
2. The I/O Channel reads its "channel program", which is a sequence of "channel commands" that indicate what to do.
3. The I/O Channel deposits data in a buffer in the main memory of the computer. This buffer exists in system memory and is accessed by the OS.
4. The I/O Channel signals the CPU that the input has been completed. If needed, it can send an error code.

Output:

1. The OS has received data from a user program and deposited the data in a buffer in the main system memory of the computer.
2. The OS signals the I/O Channel with a request for output.
3. The I/O Channel reads its channel program, which includes the address of the memory buffer containing the data.
4. The I/O Channel completes the output and signals the CPU on completion.

Input/Output in a Shared Computer

On a large mainframe system, the input and output for each program is managed within the requirement to share the resources of the computer among a number of programs. This process is called **“Time Sharing”**; it allows each user of the computer to believe that he or she is the only one accessing the computer at that time.

In the Time Sharing model, we have

1. A single computer with its CPU, memory, and sharable I/O resources,
2. A number of computer terminals attached to the CPU, and
3. A number of users, each of whom wants to use the computer.

In order to share this expensive computer more fairly, we establish two rules.

1. Each user process is allocated a **“time slice”**, during which it can be run.
   At the end of this time, it must give up the CPU, go to the “back of the line”
   and await its turn for another time slice.

2. When a process is blocked and waiting on completion of either input or output,
   it must give up the CPU and cannot run until the I/O has been completed.

With this convention, each user typically gets reasonably efficient service from the computer. Thus the computer is **“time shared”**.

Users and Supervisors

In a modern computer system, users are commonly restricted from direct access to the input and output devices. There are a number of good reasons for this.

TRUE STORY: While I was a Systems Analyst at a company in Cambridge, I was one of two programmers who had **“superuser access**” to a PDP–11 computer. This meant that each of us could write and run programs with what IBM calls **“supervisor privilege”**. As an experiment, each of us wrote a program that output directly to a shared printer. When run at the same time, the two programs produced almost nothing but illegible junk: some of his output followed by some of mine, etc.

Neither of us was able to produce an entire line of uninterrupted text output.

Modern computers (including all variants of the PDP–11) solve this problem by a process called SPOOLing, (**S**imultaneous **P**eripheral **O**peration **O**n **L**ine). The user program attempting to write to a shared printer actually writes to a disk file.

After the program closes its output, a SPOOL program operating with supervisor privilege, queues up the disk file for output to the printer.

IBM terminology for the two states is	**supervisor** and **problem**.
The UNIX terminology is	**superuser** and **user**.

Your User Program Requests Input

As noted above, your user program cannot communicate directly with an I/O device. Thus it has to request I/O service by the O/S on its behalf.

1. The user program creates a work area of size sufficient to hold the input data.

2. The user program passes a request, including the address of this work area, to the operating system. This might be called a **SVC** (Supervisor Call).

   NOTE: This is not the same as a subroutine call. A subroutine operates under the control of the user program. These I/O routines are controlled by the O/S.

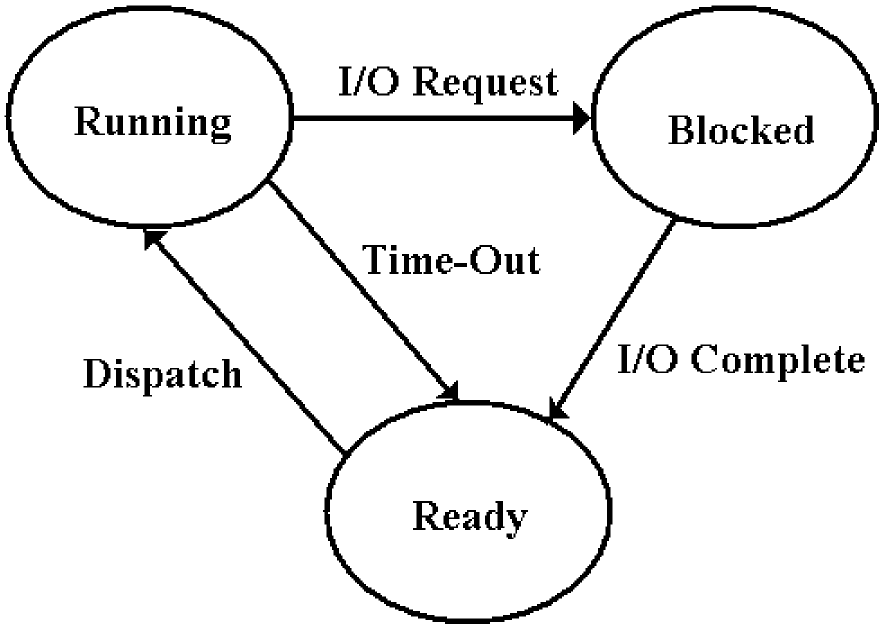
3. If the Operating System determines that the user request is appropriate and can be granted within the security guidelines, it will start an Input/Output Control Program.

4. The I/O Control Program generates sets aside a buffer in system space for the data that will be input. It then generates a sequence of channel commands, which are directions to the I/O Channel of what it to be done. It then sends a signal to the I/O Channel, which begins the appropriate processing.

5. The I/O Control Program suspends itself and the Operating System marks the user program as “blocked, waiting for I/O”. It then suspends the user program and grants the CPU to another user program that is ready to run.

The Classis Process Diagram

Here is the standard process state diagram associated with modern operating systems.



When a process (think “user program”) executes an I/O trap instruction (remember that it cannot execute the I/O directly), the O/S suspends its operation & starts I/O on its behalf.

The job is then marked as “blocked”, awaiting completion of the I/O. Another job is run.

When the I/O is complete, the O/S marks the process as “ready to run”. It will be assigned to the CPU when it next becomes available.

Levels of I/O Commands

The goal of the I/O Macros is to isolate the programmer from the detailed control of the Input/Output devices. Much of the detailed code is tedious and difficult.

The **Channel Command Level** can be seen as the lowest level.
Commands at this level are all privileged and not executable by user programs.

- Start I/O
- Test I/O
- Halt I/O
- Test Channel

The **Physical I/O Level** is the next lowest level. This level initiates the channel commands and handles I/O interrupts, all still low–level stuff. Commands include:

EXCP	Execute Channel Program
WAIT	Wait for completion of the channel program
CCB	Channel Control Block

The **Logical I/O Level** is the one that most user programs access to define the layout and start both input (GET) and output (PUT).

Double Buffering

Double buffering refers to a process that is built on the fact that the use of I/O Channels allows for simultaneous processing and input/output of data.

A double buffering scheme calls for two data buffers; call them B1 and B2. This assumes that each buffer can contain enough data to allow processing based only on the data in that buffer.

The process basically has two phases: a startup and a processing. Here is an example, supposing that buffer B1 is used first.

1. Start reading data into buffer B1.
   Wait on the buffer to be filled.
2. Start reading data into buffer B2.
   While reading into that buffer, process the data in buffer B1.
3. Wait on buffer B2 to be filled. Start reading data into buffer B1.
   While reading into B1, process the data in buffer B2.
4. Go to step 2.

One needs to be careful in order to process all of the data in the last buffer. When the I/O completes, there is still more to do.

The IOCS and Its Macros

Here are the common macros used by the IBM IOCS (Input/Output Control System).

We have mentioned a few of these in previous lectures.

DCB	Data Control Block, used to define files.
OPEN	This makes a file available to a program, for either input or output.
CLOSE	This terminates access to a file in an orderly way. For a buffered output approach, this ensures that all data have been output properly.
GET	This makes a record available for processing.
PUT	This writes a record to an output file. In a buffered output, this might write only to an output buffer for later writing to the file.

Usage of the General Purpose Registers

The operating system has special purposes for a number of these registers.

0 and 1	Logical IOCS macros, supervisor macros, and other IBM macros use these registers to pass addresses.
2	The TRT (Translate and Test) instruction uses this to store a generated value. This instruction is often used to translate other character code sets into EBCDIC.
13	Used by logical IOCS and other supervisory routines to hold the address of a save area. This area holds the contents of the user program's general purpose registers and restores them on return.
14 and 15	Logical IOCS uses these registers for linkage. A GET or PUT will load the address of the following instruction into register 14 and will load the address of the actual I/O routine into register 15.
	This use of registers 13, 14, and 15 follows the IBM standard for subroutine linkage, which will be discussed in a later chapter.

In general, only registers 3 through 12 can be considered to be truly general purpose, with register 2 useable only if the TRT is not used.

Magnetic Tape Units and Record Blocking

In order to understand the standard forms of record organization, one must recall that magnetic tape was often used to store data.

The standard magnetic tape was 0.5 inches wide and either 1200 or 2400 feet in length. The tape was wound on a removable reel that was about 10.5 inches in diameter. The IBM 727 and 729 were two early models.

The IBM 727 was officially announced on September 25, 1963 and marketed until May 12, 1971. The figure at left was taken from the IBM archives, and is used by permission.

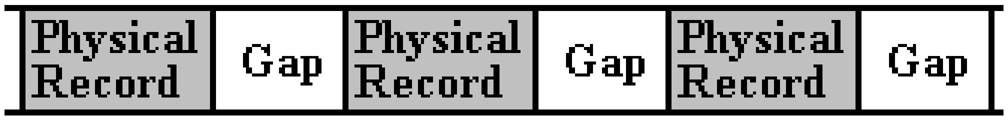
It is important to remember that the tape drive is an electro–mechanical unit. Specifically, the tape cannot be read unless it is moving across the read/write heads. This implies a certain amount of inertia; physical movement can be started and stopped quickly, but not instantaneously.

The tape must have blank gaps between records.

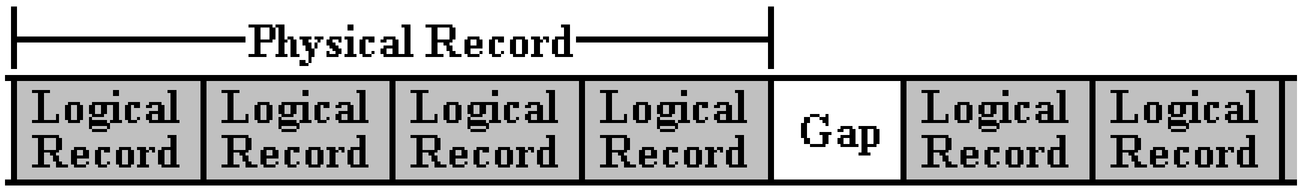
Record Blocking

The problem with inter–record gaps is that they consume space that cannot be used to write data. This is an inefficient use of magnetic tape.

Consider a sequence of small records, written one per physical block.



Space on tape can be used more efficiently if the tape is written in blocks, each block comprising a number of independent logical records.



If the last physical record is not completely filled with logical records, it will be filled with dummy records to achieve its full size.

Record Blocking Example

Consider a set of 17 logical records written to a tape with a blocking factor of 5.
There would be four physical records on the tape.

Physical record 1 would contain logical records 1 – 5.

Physical record 2 would contain logical records 6 – 10.

Physical record 3 would contain logical records 11 – 15.

Physical record 4 would contain logical records 16 and 17,
as well as three dummy logical records to fill out the count to five.

Magnetic tape is little used today for large data storage.

The idea of record blocking persists, and has been adapted to use
on large disk structures.

Use of the I/O Facilities

In order to use the data management facilities offered by the I/O system, a few steps are necessary.

The program must do the following:

1. Describe the physical characteristics of the data to be read or written with respect to data set organization, record sizes, record blocking, and buffering to be used.
2. Logically connect the data set to the program.
3. Access the records in the data set using the correct macros.
4. Properly terminate access to the data set so that buffered data (if any) can be properly handled before the connection is broken.

If the program accesses a **magnetic tape**, the programmer must do the following.

1. Either submit the magnetic tape with the job or tell the computer operator where to find the tape. The operator must mount the tape.
2. Give the operator any special instructions for mounting the tape on the tape drive.

While some of these steps might be handled automatically by the run–time system of a modern high–level language, each must be executed explicitly in an assembler program.

Standard Style for Writing Invocations of I/O Macros

Programmers have evolved a standard style for writing macro invocations in order to make them easier to read. Here is an example written in standard style.

```
FILEIN   DCB   DDNAME=FILEIN,                                          X
               DSORG=PS,                                               X
               DEVD=DA,                                                X
               RECFM=FB,                                               X
               LRECL=80,                                               X
               EODAD=A90END,                                           X
               MACRF=(GM)
```

Note the “**X**” in column 72 of each of the lines except the last one.

This is a continuation character indicating that the next physical line is a continuation of the present physical line.

Note that these seven physical lines of text form one logical line and should be read as a single logical statement.

The File Definition Macro

The **DCB** (**D**ata **C**ontrol **B**lock) is the file definition macro that is most commonly used in the programs that we shall encounter.

This is a keyword macro, meaning that every argument is passed in the form

`PARAMETER=VALUE`

While the parameters can be passed in any order, it is good practice to adopt a standard order and use that exclusively. Some other programmer might have to read your work.

The example above shows a DCB invocation that has been shown to work on the particular mainframe system now being used by Columbus State University.

It has the form:

```
Filename  DCB  DDNAME=Symbolic_Name,                                  X
               DSORG=Organization,                                    X
               DEVD=Device_Type,                                      X
               RECFM=Format_Type,                                     X
               LRECL=Record_Size,                                     X
               EODAD=EOF_Address,                                     X
               MACRF=(Input_Operation)
```

The DCB Macro Label

The name used as the label for the DCB is used by the other macros in order to identify the file that is being accessed. Consider the following pair of lines.

```
         OPEN  (FILEIN,(INPUT))

FILEIN   DCB   FILEIN,
```

Here we see that the file being opened for input is that defined by this specific DCB macro.

As we shall see later, this macro expands into a large number of assembly language instructions.

Parameters for the DCB Macro

```
Filename  DCB  DDNAME=Symbolic_Name,                                   X
               DSORG=Organization,                                     X
               DEVD=Device_Type,                                       X
               RECFM=Format_Type,                                      X
               LRECL=Record_Size,                                      X
               EODAD=EOF_Address,                                      X
               MACRF=(Input_Operation)
```

`DDNAME` identifies the file's symbolic name, such as **`SYSIN`** for the primary system input device and **`SYSPRINT`** for the primary listing device.

`DSORG` identifies the data set organization. The most common value is

`PS`	Physical sequential, as in a set of cards with one record per card.

`DEVD` defines a particular I/O unit. The only value we shall use is **`DA`**, which indicates a **d**irect **a**ccess device, such as a disk.

All of our I/O will be disk oriented; even our print copy will be sent to disk and not actually placed on paper.

`RECFM` specifies the format of the records. The two common values are:

`F`	Fixed length and unblocked
`FB`	Fixed length and blocked.

More Parameters for the DCB Macro

```
Filename  DCB  DDNAME=Symbolic_Name,                                   X
               DSORG=Organization,                                     X
               DEVD=Device_Type,                                       X
               RECFM=Format_Type,                                      X
               LRECL=Record_Size,                                      X
               EODAD=EOF_Address,                                      X
               MACRF=(Input_Operation)
```

`LRECL` specified the length (in bytes) of the logical record.

A typical value would be a positive decimal number. Our programs will all assume the use of 80–column punched cards for input, so that we set **`LRECL=80`**.

`EODAD` is a parameter that is specified only for input operations. It specifies the symbolic address of the line of code to be executed when an end–of–file condition is encountered.

`MACRF` specifies the macros to be used to access the records in the data set. In the case of **`GET`** and **`PUT`**, it also specifies whether a work area is to be used for processing the data. The work area is a block of memory set aside by the user program and used by the program to manipulate the data. We use **`MACRF=(GM)`** to select the work area option.

There are many other options.

Expansion of the DCB Macro

Here is a complete assembly language expansion of a single DCB. It is long.

```
                           163 FILEIN   DCB   DSORG=PS,
                                              MACRF=(GM),
                                              DEVD=DA,
                                              DDNAME=FILEIN,
                                              EODAD=A90END,
                                              RECFM=FB,
                                              LRECL=80
                           166+*                     DATA CONTROL BLOCK
                           167+*
0000F8                     168+FILEIN   DC    0F'0'          ORIGIN ON
                           169+*                       DIRECT ACCESS DE
0000F8 0000000000000000    170+         DC    BL16'0'        FDAD, DVTB
000108 00000000            171+         DC    A(0)           KEYLEN, DE
                           172+*                       COMMON ACCESS ME
00010C 00                  173+         DC    AL1(0)         BUFNO, NUM
00010D 000001              174+         DC    AL3(1)         BUFCB, BUF
000110 0000                175+         DC    AL2(0)         BUFL, BUFF
000112 4000                176+         DC    BL2'0100000000000000' DSO
000114 00000001            177+         DC    A(1)           IOBAD FOR
                           178+*                       FOUNDATION EXTEN
000118 00                  179+         DC    BL1'00000000'  BFTEK, BFA
000119 000074              180+         DC    AL3(A90END)    EODAD (END
00011C 90                  181+         DC    BL1'10010000'  RECFM (REC
00011D 000000              182+         DC    AL3(0)         EXLST (EXI
                           183+*                                FOUNDATION BLOCK
000120 C6C9D3C5C9D54040    184+             DC    CL8'FILEIN'    DDNAME
000128 02                  185+             DC    BL1'00000010'  OFLGS (OPE
000129 00                  186+             DC    BL1'00000000'  IFLGS (IOS
00012A 5000                187+             DC    BL2'0101000000000000' MAC
                           188+*                                BSAM-BPAM-QSAM I
00012C 00                  189+             DC    BL1'00000000'  OPTCD, OPT
00012D 000001              190+             DC    AL3(1)         CHECK OR I
000130 00000001            191+             DC    A(1)           SYNAD, SYN
000134 0000                192+             DC    H'0'           INTERNAL A
000136 0000                193+             DC    AL2(0)         BLKSIZE, B
000138 00000000            194+             DC    F'0'           INTERNAL A
00013C 00000001            195+             DC    A(1)           INTERNAL A
                           196+*                                     QSAM INTERF
000140 00000001            197+             DC    A(1)           EOBAD
000144 00000001            198+             DC    A(1)           RECAD
000148 0000                199+             DC    H'0'           QSWS (FLAG
00014A 0050                200+             DC    AL2(80)        LRECL
00014C 00                  201+             DC    BL1'00000000'  EROPT, ERR
00014D 000001              202+             DC    AL3(1)         CNTRL
000150 00000000            203+             DC    H'0,0'         RESERVED A
000154 00000001            204+             DC    A(1)           EOB, INTER
```