9 December 1964 EWD113 - 0

Segment Control.

(This report replaces EWD84)

Layout of core store and drum store.

In order to avoid confusion we introduce some new terminology. With the term "segment" we indicate an information unit of 512 words. With the term "page" we indicate a memory unit of 512 consecutive words. Core store and drum store are subdivided into pages, a segment may be stored in a core page, in a drum page or even in both (as we shall see below).

If the complete drum were available for subdivision into drum pages the number of drumpages accomodated would equal 1023. Drum pages are identified by either drum page number ($0 \leq$ drum page number ≤ 1022) or by their drum page starting address. The relation between these two quantities is given by (following a suggestion from A.N. Habermann):

drum page starting address =(1025 * drum page number)(mod 2^{19})

and

drum page number = drum page starting address (mod 1024).

From this it follows that the first words of drum pages pass the heads in order of (cyclically) increasing drum page number.

Within core store there will be a Drum Page Table "DPT", which contains one bit per drum page, indicating whether the corresponding drum page is occupied or not. The bits in DPT are arranged in such a way that a simple algorithm (using normalizing orders) can discover the first unoccupied drum page with a number (cyclically) following upon a given value; this algorithm will be used to reduce wating times for segment transfers from core to drum, whenever this is possible. (In DPT the bits corresponding to non-available drum pages will be turned permanently in the position "occupied"; non-availability might result from different sources: parts of the drum may be reserved for standard system routines or a drum page may be defect!)

The available part of core store will be subdivided into core pages of 512 words. A Core Page Base "CPB" (i.e. its starting address) will be an integral multiple of 512.

The contents of the core pages are kept track of in the core table with one Core Table Entry "CTE" for each core page. Each CTE will occupy four consecutive words in the core table; the whole of the core table will be permanently in core store in fixed locations. Denoting the base address of a CTE with :CTE the relation between :CTE and CPB is given by

:CTE = CPB / 128 + core table positionings constant.

The meaning of the constituent parts of a CTE will be described at a later stage (see below).

Up till now we have described the identity of pages; now we must describe the identity of segments. With each segment we associate a so-called Segment-Variable An SV is a one word variable which, during the life time of the associated segment, will be stored permanently on the same core location. During the life time of the segment, the segment can therefore be identified by the physical core address of the SV (denoted below by ":SV").

A segment variable is a <u>variable</u> because its value will give a clue where the segment associated with it can be found. The meaning of the different SV-values will also be described at a later stage.

States of a core page.

The mutually exclusive primary states of a core page are
a) Free
b) the Victim
c) Designated
d) Occupied.

When a core page is Free, its contents are of no interest, no drum transfers from or towards this page are pending and the page is immediately available for any other purpose. The number of Free core pages may be zero.

At every moment exactly one core page is designated as "the Victim". Whenever a program wants access to a segment which happens to be on the drum only, a drum to core instruction will be given with the Victim as the destination; at this moment a new Victim will be selected.

Remark 1. At the moment that the Victim is used as destination for a drum to core transport instruction, its contents may still be of vital importance, viz. for the completion of a pending drum transfer given to dump a segment from core to drum. The difference between the Victim and a Free page is that a Free core page is instantaneously available for any other purpose, whereas the cores of the Victim will become available in due time, i.e. certainly after completion of the pending drum transports.

Remark 2. Essential use is made of the fact that the drum transports are performed in order in which they are given. If a machine were equipped with two independent drums, say drum A and drum B, and simultaneous transfer possibilities (and two independent chains of transport instructions for the two drums) then analogous techniques seem possible, provided one introduces two victims, Victim A and Victim B. This situation has also been investigated, it did not seem to present grave additional difficulties. In this report the multidrum situation will not bee pursued any further.

A core page is Designated when it has been decided which segment it is going to contain, while the availability of the segment in this core page will only come into effect after the completion of one ore more pending drum transports. As soon as these are completed the page becomes Occupied.

A core page is Occupied when it contains a segment, when the information of this segment is instantaneously available and when the page is not involved in a pending drum transport. Occupied core pages fall into two classes: those that contain an Original, or that contain a Copy. In the first case the segment in question is regarded as being stored in the core page only; if this core page is neede for other purposes, this means that a suitable non-occupied drum page has to be selected, that a core to drum transport has to be given and that care must be taken that this core page will not be used effectively before the completion of this dumping transport. If the core page, however, contains a Copy, this means, that the segment in question is stored on a drum page as well. In that case no dumping transport order has to be given in order to free the core page for other purposes.

Before describing the possible transitions between these four states of a core page, we must mention the "empty segment". Segments can be introduced, i.e. SV's are introduced; at the moment of introduction, however, they are made to point to "an empty segment". Actual reservation of 512 words of storage is postponed till the moment that the segment is actually used for the first time. (This allows us to "overdeclare" arrays without any appreciable loss.)

State transitions of a core page.

Between the four states six directed transitions are possible; these transitions are instantaneous in the sense that they will be accomplished by an uninterrupted sequence of X8 instructions.

1. Free → Occupied.

When one of the programs desires access to an empty segment at a moment that the number of Free core pages is greater than zero, then one of the Free core pages is supplied to store the segment. This is the only case where a program request for a new segment is granted immediately, bypassing the Segment Controller (see below). With "a new segment" is meant here: a segment not present in core store.

2. Occupied → Free.

Whenever a program decides that one of its segments ceases to exist (e.g. segments of local arrays at block exit or at stack collapse) and the segment in question occupies a core page, then this core page will become free.

3. Free → the Victim.

When the old Victim is used and a new Victim must be chosen, one of the Free core pages (if present) may be selected.

4. Occupied → the Victim.

When the old Victim is used and a new Victim must be chosen, one of the Occupied pages may be selected. If the Occupied page selected was an Original, this choice implies a dumping transport from core to drum.

5. The Victim → Designated.

When a new segment request cannot be granted instantaneously, and there is not already a transport order pending that will bring the segment required into core store, then the Victim will act as destination for the new segment. When this new segment has to come from the drum (i.e. was non-empty) then this implies a drum to core transport order.

6. Designated → Occupied.

The previous five transitions take place under direct control of the program that needs or kills a segment. The sixth transition takes place under control of a special purpose abstract machine, the so-called "Segment Controller". In all cases that a segment request cannot be granted instantaneously, this is due to the fact that one or more pending drum transfers must first be completed. It is the task of the so-called "Segment Controller" -the progress of which is synchronysed with respect to the interrupt signals of the drum transfers- to keep track of these (and other) completions.
The main tasks of the controller are
a) to effect the transition Designated → Occupied
b) to notify the program waiting for this completion
c) to do the above at the proper moment.
Its structure will be described in more detail at a later stage.

The possible values of an SV.

An SV, SVi say, can be in one of foue different states; the first two are applicable when no core page is associated with the segment, the last two when there is a core page associated with the segment.

No core page association arises in two cases, either because the segment is empty, or because the segment can be found on the drum only.

1. Segment empty.

    SVi = - 0

2. Segment on drum.

    SVi = { d[26] = 1
    { d[25] = 0
    { d[24].....d[0] = DPB, to the left supplied with zeros.

Legenda. In this description the bits of SV are numbered in the usual manner with d[0].......d[26] in order from right to left, from least to most significant; X d[26] is thus the sign bit. With DPB (Drum Page Base) is meant the drum page starting address. (At present it will extend over the 19 bits d[18]...d[0]; this coding leaves room for 63 additional drums!)

When a segment is associated with a core page we distinguis between XXX two different cases: either the core page is Designated because the segment has not arrived yet in the oore page, or its presence is indeed well established in the core page, whibh is then Occupied. This difference will be indicated in the SVi; the coupling to a oore page nr.k will be made by storing in the SVi the :CTEk, i.e. the base address of the corresponding core table entry.

3. Segment in core.

X SVi = :CTEk, to the left supplied with zeros.

This will indicate that the segment is to be found in the Occupied core page nr.k.

4. Segment coming.

    SVi = - (:CTEk, to the left supplied with zeros)

This will indicate that the XXXXXXX segment in question is due to become available in core page nr.k, but that its arrival has not been confirmed yet by the Segment Controller.

State 4 can be distinguished from state 1 because XXXXX SV $\neq$ 0, and from state 2 because d[25] = 1. The above rules imply that the definite presence of a segment in core)state 3) can be established by inspection of (the sign bit of) SVi only.

Remark. When an Original is chosen as the new Victim the coupling between the SV and the core page is disconnected instantaneously and the SV returns to state 2 "Segment on drum" although in actual fact quite some time (at least one, but maybb many more drum transfers) may elapse before the segment has really arrived in the drum page chosen. The drum transfer start organisation and the segment cobtroller together see to it that

a) the segment will not be read efficively from the drum page before it has effectively arrived there

b) the core page will be left untouched until the dumping transfer has been completed.

The possible values of a CTE.

When a core page is Designated or Occupied the four words of its CTE are:

CTE[0] = :SVi

CTE[1] = DPB if the (coming) segment is a Copy
       = − 0 if the (coming) segment is an Original

CTE[2] = XXXX CPB (= 128 *(:CTE − core table positioaings constant))
         This value is a constant in time; it has been introduced to
         speed up the address arithmetic.

CTE[3] = d[26] = 0,
         d[25] = 1 when Designated
               = 0 when Occupied
         d[24]....d[20] = holyness counter (see below)
         d[19].....d[0] = interest number ($\neq 0$)

When a core page is Free the values are:

CTE[0] = chain element

CTE[1] = − 0

CTE[2] = CPB

CTE[3] = d[26] = 0, d[25] = ...= d[0] = 1

The Free core pages are chained: each CTE[0] contains the :CTE of the next Free core page, if present.

The CTE corresponding to the Victim has the same layout as that one of a Free core page, but for the fact that CTE[0] is meaningless and unused.

The Segment Controller.

As we shall see below a seventh core page state transition will be introduced, viz. the transition Occupied → Occupied (but by different segments) following a suggestion made by P.A.Voorhoeve during a discussion of the first half of this report. Also in this transition a segemtn request will be granted bypassing the Segment Controller.

The program actions we are going to describe now are the request for an empty segment and the request for a non empty segment.

The request for an empty segment.

If there are Free core pages a Free core page is selected.(The Free core pages will be chained by their CTE[0] contents and they will be used in stack fashion.) The Victim remains unaltered and the request is granted bypassing the Segment Controller.

If no Free core pages are present the ruling Victim could be used; this would go via the Segment Controller and as a result the segment would only beX handed over to the requesting program "in due time" (see below). Furthermore the use of the ruling Victim always XXXX implies the selection of a new Victim. Therefore, in the case of empty segment request without Free core pages the new Victim selection is carried out first. If the choice for the new Victim happens to fall on a core page containing a Eopy, then this new Victim will be handed over to the requesting program instantaneously; the Segment Controller is bypassed and the (old) Victim retains its position. If the new Victim choice happened to fall upon an original, however, the effort to bypass the Segment Controller has not succeeded. In that case the XXXX old Victim will be handed back in due time to the requesting program; for further details, see below.

<u>The request for a non empty segment.</u>

If the SV indicates that the segment is non empty, not present in core and not coming, then a drum to core instruction must be gicen, for which the Victim is chosen as the destination page. Then a new Victim must be selected. If there are free core pages, one of them may be chosen; otherwise one of the Occupied core pages with a holyness counter value = 0 (see below) must be selected. This may be a Copy, in which case "uncoupling) of the SV and the CTE is the only action necessary; it may be an Original, in which case a next, but now core to drum, transport instruction must be given. In order to see to it that under no circumstances pages are used effectively before pending transports for such a page are completed each program gives in situations like the above a message to the so-called Segment Controller. The Segment Controller processes these messages in the order in which they have been given to it, but does so in synchronisation with the drum interrupts. Giving a new message to the segment controller is an undividable action; when the message has been given completely the requesting program stops on a semaphore, which will be increased by bhe Segment Controller when the presence of the segment asked for can be guaranteed by the latter.

A message to the Segment Controller consists of two halfs, the first half is concerned with the segment asked for, the ~~segment~~ second half is concerned with the possible drum transfer given to make a new Victim.

The first half of a message to the Segment Controller contains:
a) an identifieation of the program semaphore which has to be increased by 1 by the Segment Controller upon completion of the processing of this first half message.
b) a bit indication whether a drum to core transport was part of this first half message
c) if so, the complete specification of the drum transfer in question (taken up in the chain of drum transfers); if not, whether the request was for an empty core page with standard neutral contents. In all cases it will contain the CPB (or the CTE) of the oore page involved.

The second half of a message contains
a) a bit indicating whether a core to drum transfer was part of this second half message; if not the second half is regarded further as empty
b) if so the complete specification of the drum transfer in question (taken up in the chain of drum transfers).

The independent programs put their message to the Segment Controller in a cyclic Segment Control Buffer. Upon completion of the message specification (by such an independent program) it signals the presence of a next message to the Segment Controller by means of a V-operation on SCS (Segment Control Semaphore) which is initialized at the value zero (i.e. "no messages").

Neglecting the tentative eecovery measures by the segment controller in the case of error detection in the drum transfers, the basic structure of the Segment Controller is as follows:

```
"LL:  P(SCS);
      if first half under consideration contains drum transport then
                        P(Interrupt Semaphore Drum);
      clerical processing of the first half;
      V(program semaphore specified in the first half under consideration);
      if second half under consideration contains drum transport then
                        P(Interrupt Semaphore Drum);
      cyclic increase of Segment Control Buffer Read Pointer; goto LL"
```

(We have assumed that offering of messaged to the Segment Controller will take place under supervision of a counterpart: the Segment Control Buffer Write Pointer, also to be increased cyclicly.)

The holyness counter.

In each program certain privileged segments may occur, which during a certain period of time should be permanently stored in the same core page. Such a period is initiated by "asking for a holy segment", such a period is closed by the inverse operation, viz. "profanation of a segment".

In the ALGOL organisation examples of holy segments are the stack segments and the program segment under execution.

We should like to make some remarks.

The holyness concept is also used whenever a program asks for a segment for, maybe, a single reference, say an array page. When the reference is desired, presence in core of the segment is investigated. If it is present, the reference is made instantaneously. (Between the establishing of the presence and the actual reference, the segment is of oourse not allowed to disappear from the core; this is, however, taken care of by disabling the interrupt temporarily and the state of holyness remains unchanged in this situation.) When the segment is not in core, the program asks for it, but "holy". When the is allowed to continue, it performs the reference and profanates the segment. Upon arrival of the segment, the Segment Controller signals to the requesting program that it <u>can</u> proceed; by the holyness we have guaranteed that, when the program actually proceeds, the segment asked for is indeed <u>still</u> there!

The next remark is, that holyness cannot be indicated by a single bit, but by positiveness of a counter. For instance, more different programs may wish access in a holy way to the same segment (say a library segment). In that case the page should continue to contain the segment until all of them have profanated this segment. For that reason we have introduced the holyness counter; profanation means subtraction of 1 from this counter, asking for a holy page means setting the holyness counter to1 if a new core page must be selected for it, it implies an increase by 1 of the holyness counter, if already a core page was coupled to the segment asked for.

The concept "multiply holy" is not confined to library pages (remark made by F.J.Hendriks): in the ALGOL system the implementation of simple parameter subroutines has made it desirable to allow each ALGOL program to declare two program segments holy; if these segments happen to coincide its holyness counter will be equal to 2.

This and the limited number of bits, set aside for the value of the holyness counter in CTE[3] puts an upper limit to the number of concurrently running programs. If they were all general ALGOL programs, this upper limit would be 15 (Remark made by C. Ligtmans). In practice this limit will be higher and is therefore assumed never to be a nuisance.

The proces of Victim selection will be such that never a ~~[illegible]~~ ~~[illegible]~~ core page will be chosen with a holyness counter positive. The packing of the holyness bits and the interest number in one single word is made to facilitate the page choice for the new Victim.

The size of the SEGMENT CONTROL BUFFER

In the description of the segment request we have assumed that the Segment Control Buffer was sufficiently large to accept the new message. It is absolutely mandatory ("unbedingt Notwendig"!) that this condition is satisfied! Earlier efforts have been made, where such a buffer could be full, but this gives rise to nearly prohibitive strategical and clerical complications; for during the period of time that the requesting program waits for room in the buffer, the situation in ~~[illegible]~~ core may have changed!) It can be proved that with N concurrently running programs a Segment Control Buffer with a capacity of N + 1 messages is always sufficient.