(12) UK Patent Application (19) GB (11) 2 162 406 A

(43) Application published 29 Jan 1986

(21) Application No 8515318

(22) Date of filing 17 Jun 1985

(30) Priority data
(31) 8415474 (32) 18 Jun 1984 (33) GB

(51) INT CL⁴
H04L 11/16 G06F 1/04 12/10 15/20 G06K 9/00

(52) Domestic classification
H4P PC PEC
G4A FT MP NX
G4R 11A 11D 11E 1X 8G RP

(56) Documents cited
GB 1518565

(58) Field of search
H4P

(71) Applicants
Logica (UK) Limited (United Kingdom),
Cobham Park, Downside Road, Cobham, Surrey KT11 3LX.
The Secretary of State for Defence (United Kingdom),
Whitehall, London

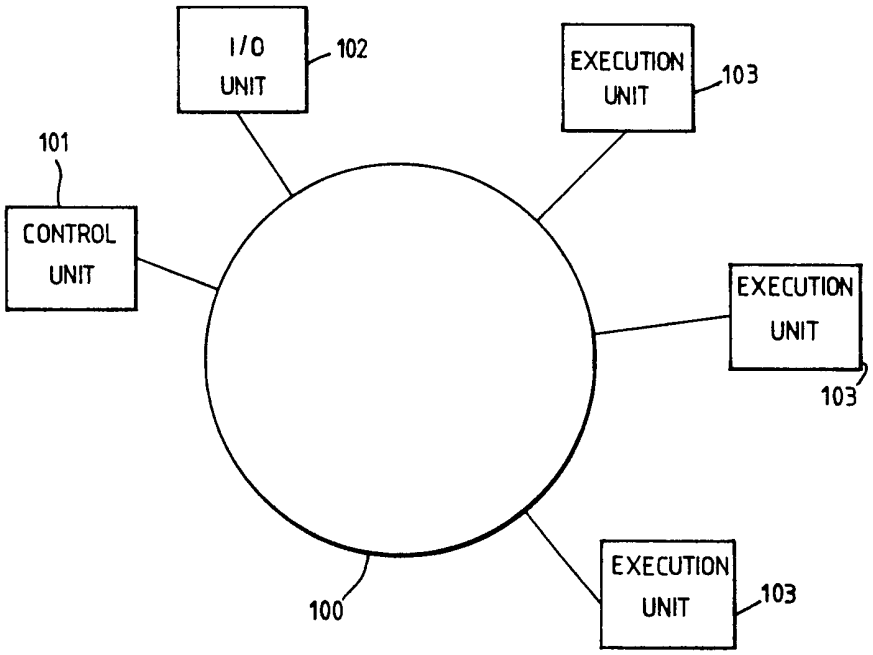
(72) Inventors
Peter Kenneth Bailey
Peter Jonathan Brumfitt
Andrew Crofton Sleigh
Neil Francis Trevett
Nicholas Maxwell Trier

(74) Agent and/or Address for Service
Mewburn Ellis & Co,
2/3 Cursitor Street, London EC4A 1BQ

(54) Computer system

(57) A computer system has a control unit 101, an input/output unit 102, and plurality of execution units 103, connected to a network bus 100. The network bus 100 has data lines corresponding to the number of units connected to it, and each execution unit has a data line associated with it in which it signals its status. These status signals may be used to control the sequence of operation of the execution units. Details of the structure of the execution units are disclosed, as is the use of the computer system in an image recognition device. Each execution unit includes a bank switched memory (306, Fig. 4) which permits bidirectional transmission between a transmission control processor (301) and an execution processor (302). A program memory for the 16 bit execution processor uses 24 bit words. The excess bits are used to address 2 megabytes of RAM via intermediate translation (Fig. 4). The processor clock rate is automatically adjusted in accordance with operations to be performed.

Fig.1.



The drawings originally filed were informal and the print here reproduced is taken from a later filed formal copy.

GB 2 162 406 A

2162406

Fig.1.

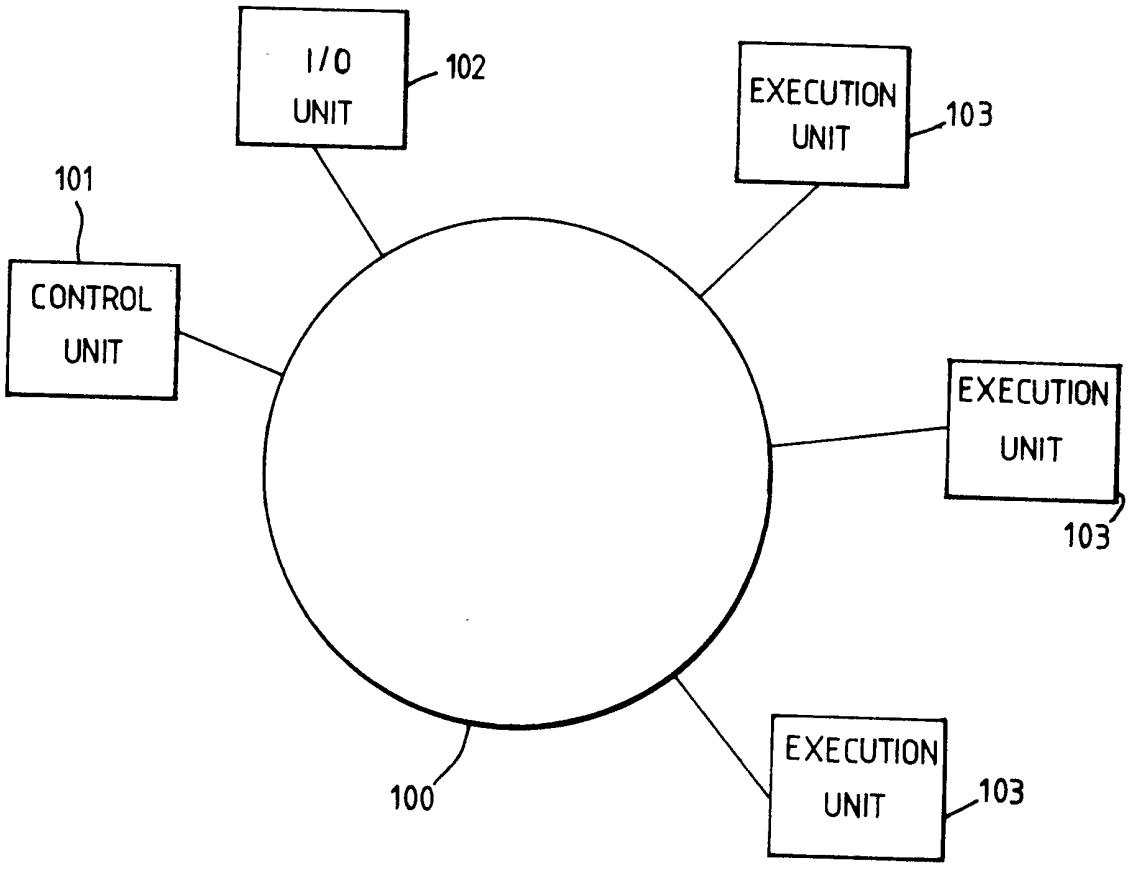
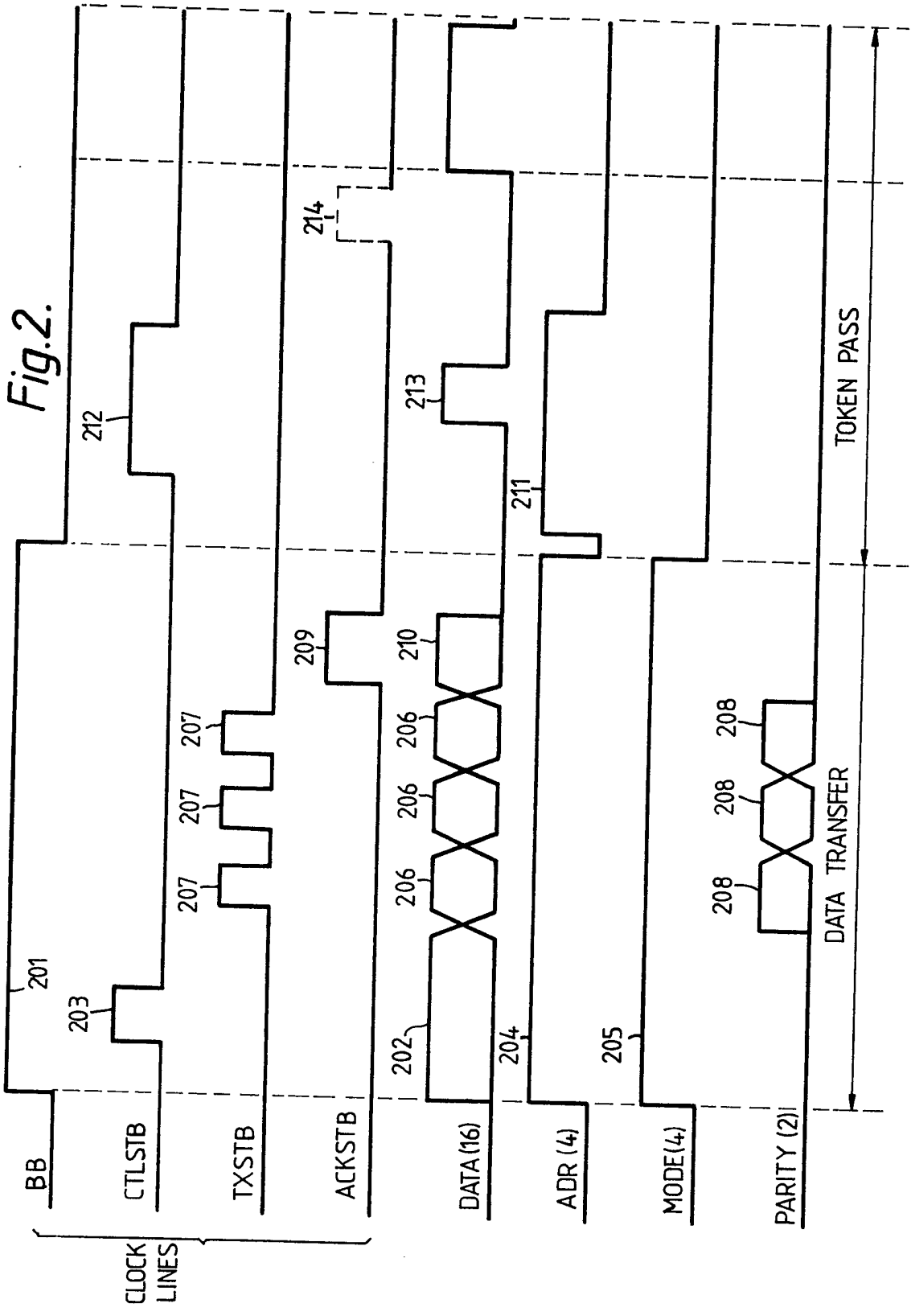
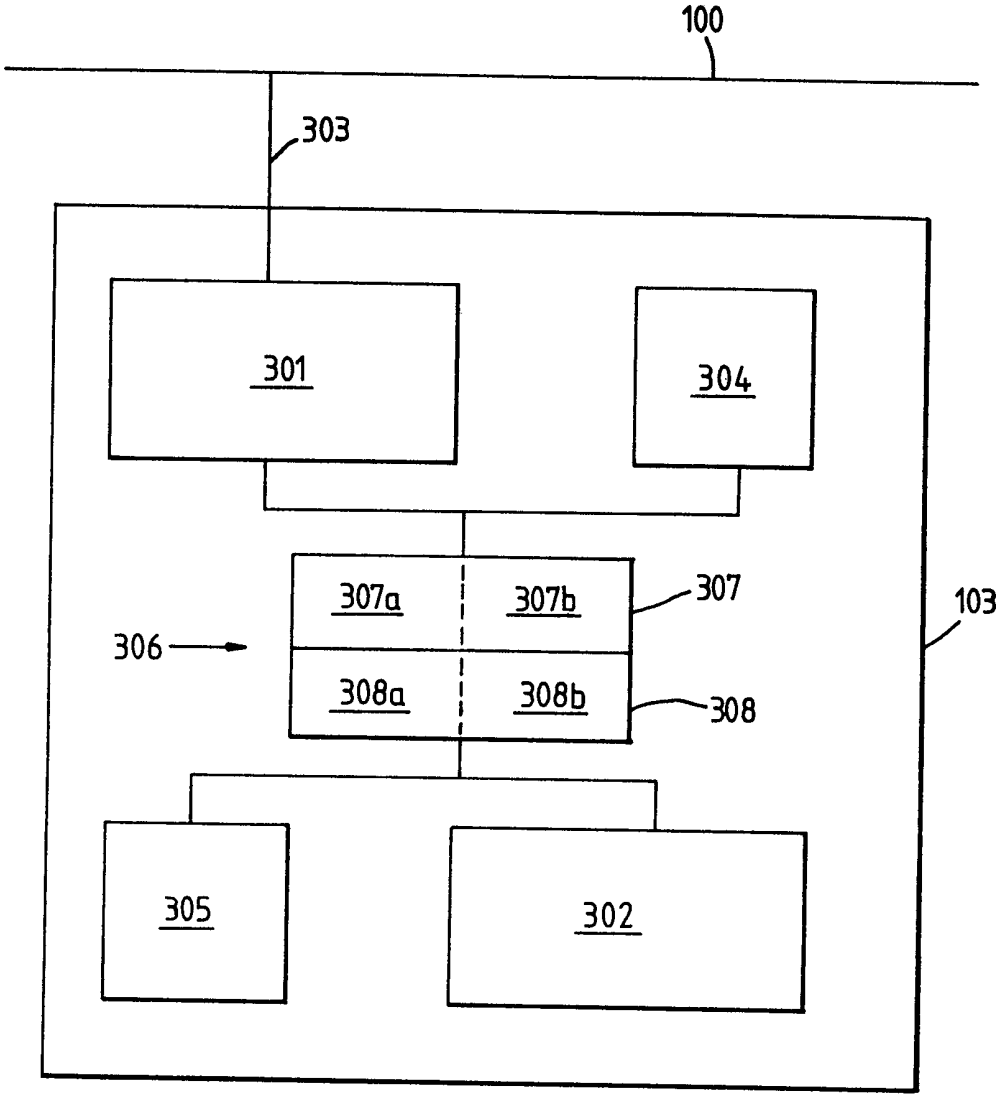


Fig.2.


BB
CTLSTB
TXSTB
ACKSTB
DATA (16)
ADR (4)
MODE (4)
PARITY (2)
CLOCK LINES
DATA TRANSFER
TOKEN PASS
201
202
203
204
205
206
207
208
209
210
211
212
213
214

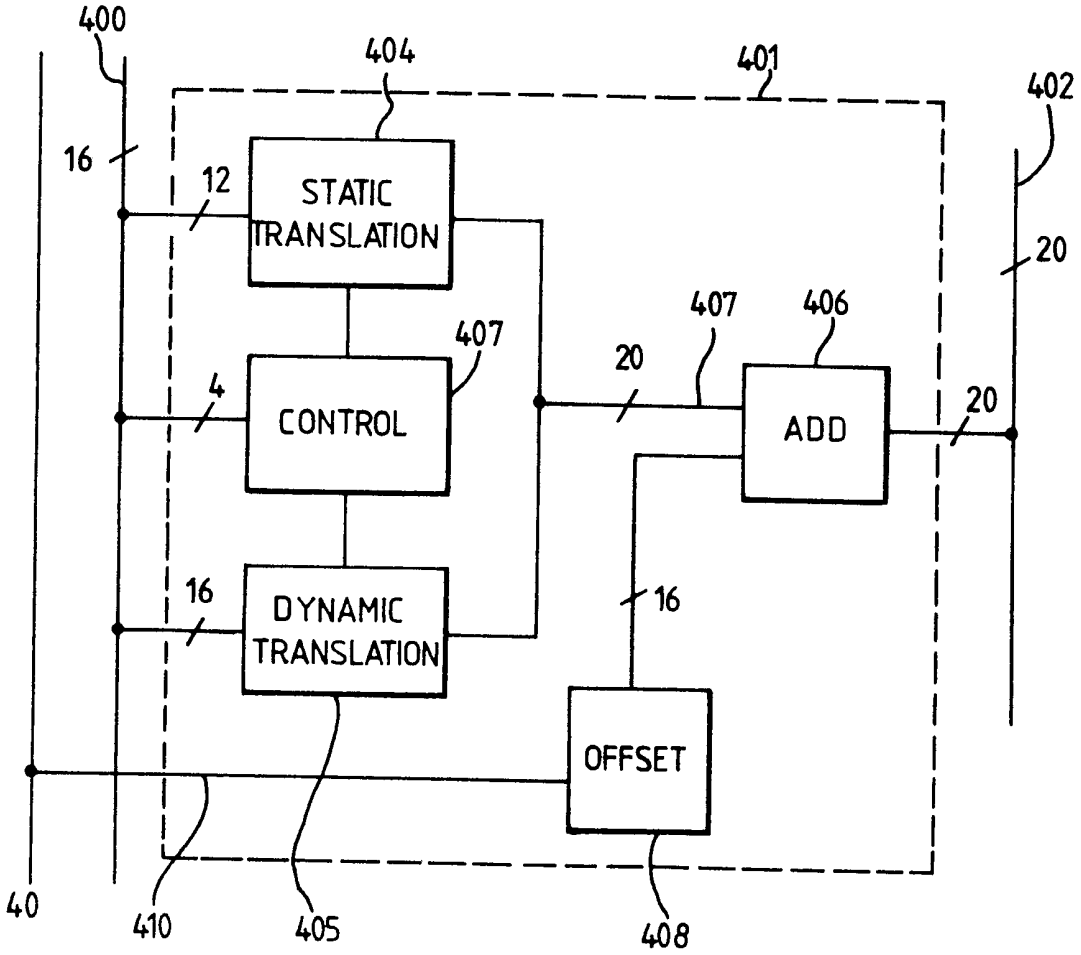
2162406

Fig.3.



2162406

Fig.4.



2162406

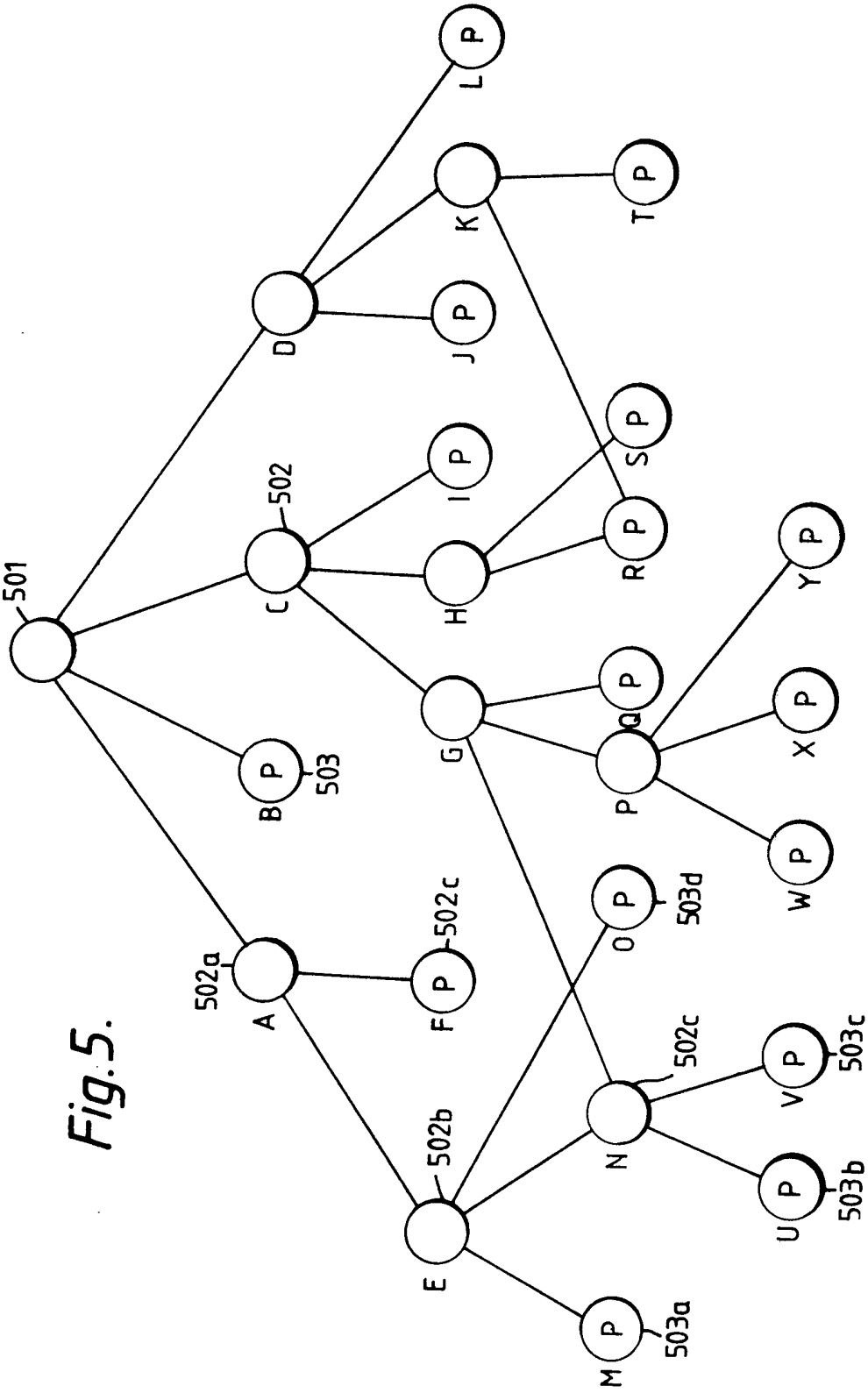


Fig.5.

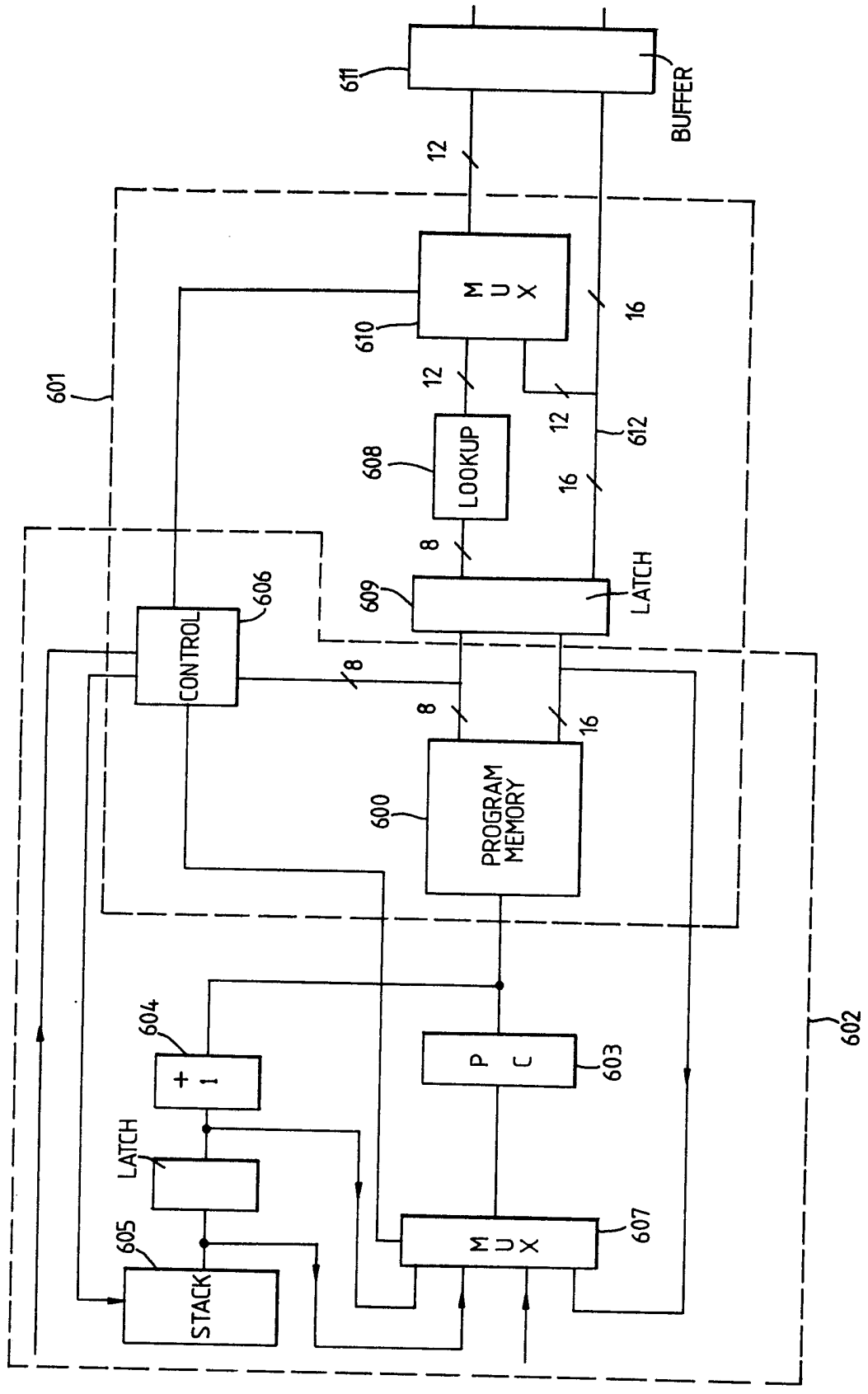

611
BUFFER
12
MUX
610
16
601
12
12
612
608
LOOKUP
16
8
609
LATCH
606
CONTROL
8
8
16
600
PROGRAM MEMORY
604
+ 1
P C
603
602
LATCH
605
STACK
MUX
607

Fig.6.

Fig.7.

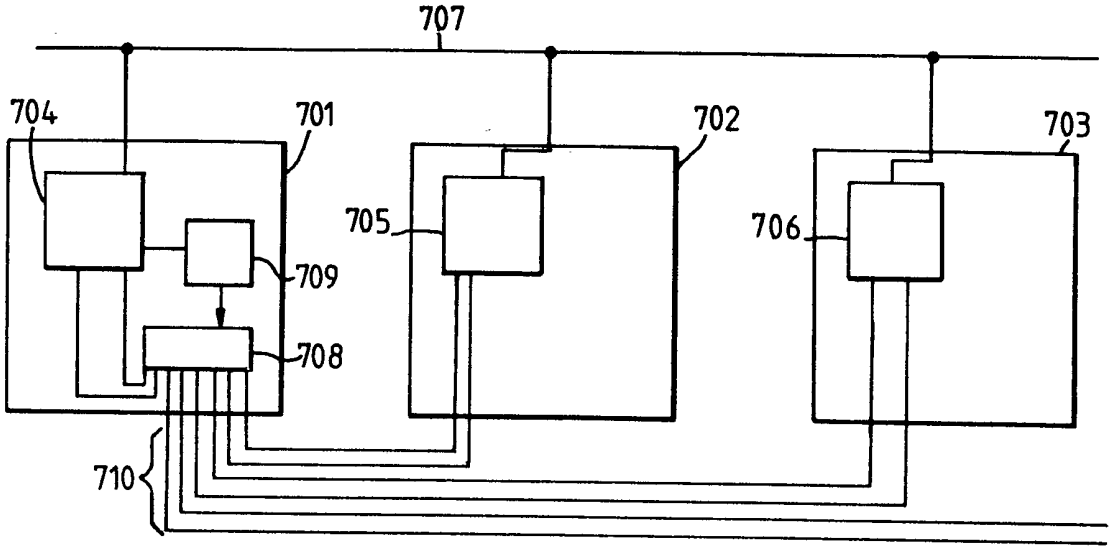

707
704
701
702
703
705
706
709
708
710

Fig.8.

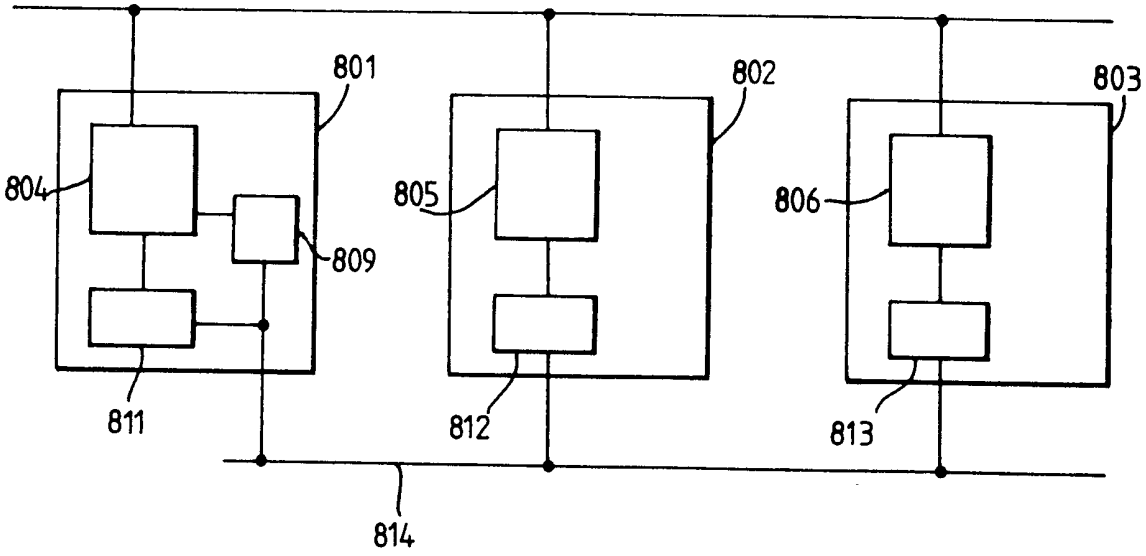

801
802
803
804
805
806
809
811
812
813
814

Fig.9.

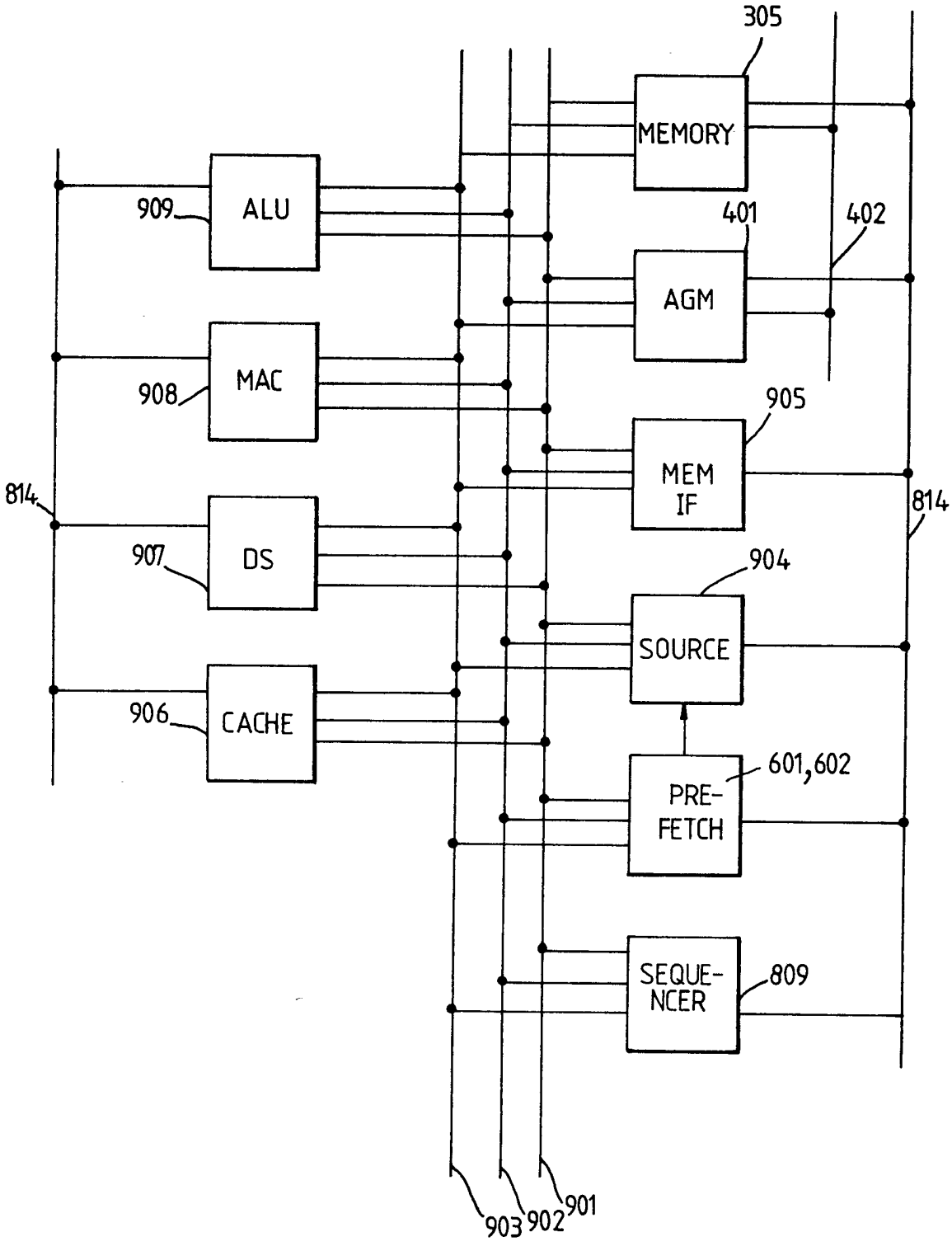
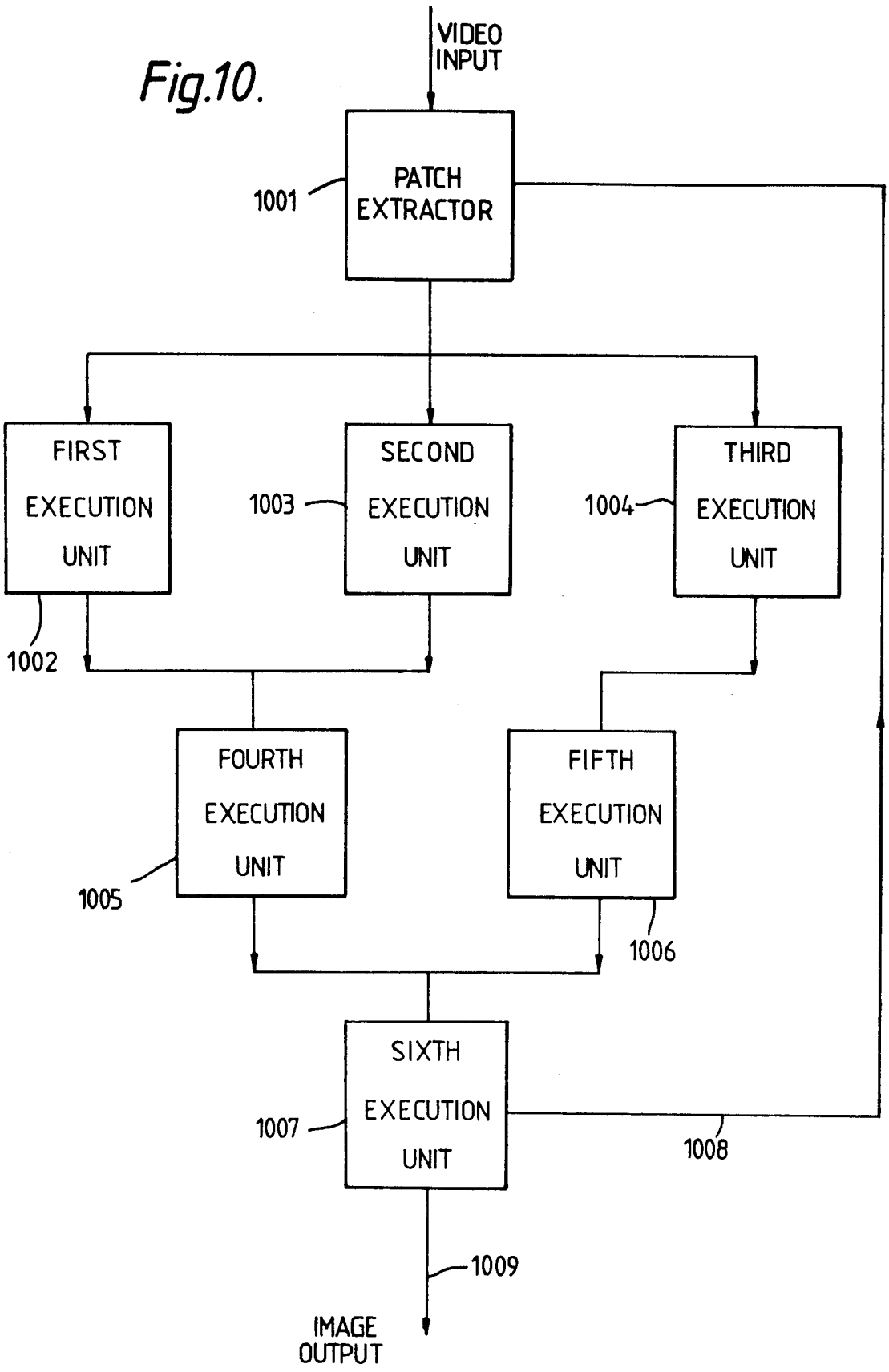

305
MEMORY
909
ALU
401
402
AGM
MAC
908
905
MEM
IF
814
814
907
DS
904
SOURCE
906
CACHE
601,602
PRE-
FETCH
SEQUE-
NCER
809
903
902
901

Fig.10.


VIDEO INPUT
PATCH EXTRACTOR
1001
FIRST EXECUTION UNIT
SECOND EXECUTION UNIT
1003
THIRD EXECUTION UNIT
1004
1002
FOURTH EXECUTION UNIT
1005
FIFTH EXECUTION UNIT
1006
SIXTH EXECUTION UNIT
1007
1008
1009
IMAGE OUTPUT

SPECIFICATION

Computer system

The present invention relates to a computer system, and to various features of that system.

In many image recognition devices in which image recognition is carried out by computer, the main problem is that the amount of processing required to make sense of the image being processed is very high. If the amount of time for the device to recognise the image is not to be too great, the device must operate very rapidly and be capable of being adapted to various recognition strategies, and it is not possible to obtain the necessary speed simply by suitable programming.

The present invention originated in a desire to obtain a computer system which would permit very rapid processing of image information. It was found that significant improvements in processing time could be made by suitable design of both the overall architecture of the system, the structure of various components of the system, and the interaction of those components with the programming language to be run on the system. The present invention therefore has a number of aspects directed to the various features that have been found to be desirable. The computer system is organised as a plurality of units (hereinafter execution units) each of determined by the program stored in that unit, the units being connected together via a network bus. The majority of the aspect of the present invention concern the structure and processing operations of the execution units, but we will first deal with the aspect of the present invention concerning the system as a whole.

The first aspect relates to the way the execution units signal on the network bus. The network bus has a plurality of data lines, via which data may be transferred between the execution units, but in addition each execution unit has assigned to it a specific one of the data lines. The assigned data line then becomes a signalling line for the corresponding execution unit. The signalling line may then signal that the execution unit is ready to receive data, ready to transmit data, has validly received data, or has validly received the right to transmit.

Thus an execution unit which is ready to transmit data may detect that the execution units to which that data is to be sent are ready to receive that data, by checking the signals on the data lines which are signal lines of those execution units. The transmitting execution unit may then apply a signal to those signalling lines to signal to the appropriate other execution units that it is about to transmit data to them, then the data is transferred via the data lines. At the end of reception of the data the execution units receiving that data may signal on their signal lines that they have validly received the data, and this will be detected by the execution unit which has transmitted the data.

The present invention is thus applicable to a "token passing" ring in which a notional "token" is passed between the various execution units, the execution unit with the token being the one that is to transmit data, and that the end of transmission of e.g. a packet of data from that execution unit the token is passed to another execution unit. The passing of the token may be monitored by a master unit which monitors the signals on the various data lines, and also may initiate the token passing.

A computer system according to the first aspect of the present invention may be used in an image recognition device. Each execution unit may be programmed to carry out a particular part of the analysis of an image, with the processing being carried out in both a parallel and a pipeline way. Thus a number of the execution units operate in parallel to perform a first set of analysis operations on data representing a part of the image to be processed, and then pass their results to one or more other execution units, which again work in parallel to process the data further. This may be repeated for an appropriate number of stages until the desired result is achieved. Although the image recognition operations carried out by the various executions units for data analysis are in themselves known, the use of the arrangement of the first aspect permits very rapid interchange between the various execution units, and so improves the speed of image processing. The last stage of the image recognition device (i.e. the last execution unit in the sequence) may generate a signal which causes another part of the image to be fed to the first stage, so that the image recognition device operates sequentially on various selected parts of the image with each part being fully processed before the next part is analysed. This is very important because it permits the next part to be analysed to be chosen in dependence upon the results of the first analysis, which is desirable in many image recognition strategies.

The other aspects of the present invention relate to features of the execution units.

In a second aspect of the present invention an execution unit has two processors and a memory accessable by both processors. The memory is divided into two areas and at any time, one area is accessable by one processor and the other area is accessable by the other processor. At some suitable time access is switched so that one processor gains access to the area formerly accessed by the other processor and vica versa. The effect of this may be considered as an exchange of data between the two memory areas so that data can be transferred from one processor to the

other, but at no time is there competition between the two processors for memory access.

Thus, suppose that data to be processed by the first processor is stored in a first part of one memory area, and data to be processed by the second processor is stored in a second part of the other memory area. The first processor processes the data in the first part of the one memory area with the processed data being stored in a second part of that memory area, and similarly the second processor processes the data in the second part of the other memory area, and stores the processed data in a first part of the other memory area. Then data can be transferred between the processors simply by switching the access to the memory area, so that data processed by the first processor and stored in a second part of the one memory area may be accessed for processing by the second processor, and similarly data processed by the second processor and stored in the second part of the other memory area may be accessed for processing by the first processor.

This second aspect is particularly useful in conjunction with the computer system of the first aspect. One processor controls the transfer of data to and from the network bus, whilst the other processor processes the information received. Incoming data to an execution unit is stored in the second part of one memory area and is transferred to the first part of the other memory area of the second for subsequent processing. The processed information is then stored in the second part of the other memory area for subsequent transfer to the first part of the memory of the first processor from where it is then distributed to other execution units. This arrangement allows the two processors to operate simultaneously in a series of "frames". Within each frame, the first processor receives information from, and passes information to, other execution units, while simultaneously the second processor processes data received during the previous frame. The frame ends with the transfer of fresh data from the first to the second processor, and the transfer of processed data from the second processor to the first, simply by giving access to the opposite memory areas. The control circuitry and programming for recognising correct transmission and reception of data via the network bus may all be contained within the first processor, so that the second processor is entirely free to process the information received, and this may occur in parallel with transmission of data from the previous frame and reception of data for the next frame.

The third aspect of the present invention relates to address generation. The aim here is to convert a data signal (which is a number not corresponding to an address) to an address signal of an increased number of bits. If the most-significant-bit of the data signal is spaced by less than or equal to a predetermined number of bits from the least significant bit, that predetermined number of bits, starting at the least significant bit, is fed to a conversion memory, which converts that signal to an address signal of a greater number of bits. If the most significant bit of the data signal is spaced from the least significant bit by more than the predetermined number of bits, the conversion memory is inhibited, and the entire data signal is converted to an address signal by shifting it by 0, 1, or more bits, with the bits of the address signal not corresponding to a bit of the data signal being set to zero.

Thus, suppose the data signal has 16 bits. If the four most significant bits are all zero, the twelve least significant bits are fed directly to a conversion memory to be converted to 20 bit address. It is convenient if the address signals thus generated are used to provide the addresses of static data objects in the memory (static data objects being data objects which do not change position during the running of a program). If any of the four most significant bits of the data signal is non-zero, those 4 bits are fed to a control unit which inhibits the conversion memory, and which enables a transmission unit, which transmission unit shifts the entire 16 bit data signal by 0, 1, 2, or 3 bits to form a 20 bit signal.

In this way a 16 bit signal can be converted into a very large number (2 MByte) of memory addresses. 4K addresses are generated by direct conversion of the bottom 12 significant bits of the data signal, whilst the other addresses are generated by shifting of the entire data signal.

The fourth aspect of the present invention relates to hardware structure. When a computer system is to be implemented on more than one board, control signals must be transferred from the microprogram memory to the various elements of the processor. In the prior art, the system has a single microprogram memory on the board, and control signals are generated from that memory and transmitted via an interconnecting "backplane" to the other boards. This system has the problem that the total number of signals transmitted via the backplane is limited, as is the microprogram memory of the system. Therefore, if it is to be possible to increase the capacity of the system, by increasing the number of boards, spare capacity must be built into the system. This is wasteful, and the seventh aspect of the present invention seeks to solve this.

The fourth aspect therefore provides a computer system fabricated on a number of boards, in which each board has a microprogram memory. The microprogram memory may store the entire microprogram or simply the relevant part of the microprogram for that

board. One board generates microprogram addresses (e.g. from a sequencer) which are transmitted in parallel to the microprogram memories of each board, for extraction of the appropriate microcode. The various components on the board then respond to that extracted microcode. The advantage of this system is that it is unnecessary to built in any spare capacity, because additional boards may be added simply by connecting the microprogram of that board to the microprogram address generator. This means that the system is extremely flexible, and the size of the microprogram memory does not have to be excessively large.

The fifth aspect of the present invention relates to the cycle time of each processor. It is normal for the operations of a processor to be controlled by a fixed frequency clock sending out a regular stream of pulses which cause the other components of the processor to operate in a regular way. The time taken for each operation of each component may depend on that operation, but with a fixed frequency clock the total processing time is determined entirely by the clock rate. It is also known to provide clocks of variable frequency, so that the length of each clock cycle can be set to the length of the longest operation to be performed that cycle and time can be saved when all the operations carried out by the various components are short. However such variable frequency clocks depend on a manual determination of the lengths of the various operations, and in a complex program this is virtually impossible. Therefore the fifth aspect concerns a way of achieving variation in the clock length automatically in dependence upon the length of the operations that must be performed during that cycle.

It is achieved by using an assembler to calculate the longest operation in a microword. An assembler "assembles" a microword in response to an input command from a plurality of "fields", each of which may represent an instruction for a part of the processor. In known devices one of the fields is a "clock" field which controls the length of the clock cycle, and in the prior art this clock field must be pre-calculated for each microword. In the fifth aspect of the present invention there are no pre-programmed clock fields, but all the other fields each have information relating to the duration of that field, and the assembler calculates a clock field from the other fields. The clock field then is added to the microword which is assembled by the assembler and controls the clock cycle in dependence on the longest operation of each microword. Thus the clock fields are calculated automatically, unlike in the prior art, and this permits automatic regulation of the clock rate.

The sixth aspect of the present invention relates to the relative length of data signals and the length of words in the program memory. In known systems these are the same, so that if an instruction needs both an address and a parameter field, these have to be formed by two separate words from the program memory. This is wasteful of memory space and is inefficient and so the sixth aspect of the present invention provides a program memory with each word being longer than the word length of the processor to utilize that program, and with the words in the memory being divided into a tag, with a length corresponding to the number of additional bits of memory word relative to the processor word, and a parameter field of the same length as the processor word. This use of a program memory with word length exceeding processor word length by a plurality of bits which form a tag for that word has two advantages.

Firstly, it permits a simple way of converting the program memory word to an address and parameter field by deriving an address for the parameter field either from the tag or from the parameter field itself. Thus all but one of the possible tags may be converted to a corresponding address with a larger number of bits via a conversion memory. The remaining one tag, however, desirably inhibits the conversion memory, and instead an appropriate number of bits are extracted from the parameter field signal to form an address. Thus, imagine the program memory stores words of 24 bits, corresponding to a parameter field of 16 bits and a tag of 8 bits. All but one (i.e. 255) tags may be converted into a 12 bit address via a "look-up" RAM forming the conversion memory. When the one remaining tag occurs, the look-up RAM is inhibited, e.g. by a control unit detecting this one tag and inhibiting the RAM accordingly and the 12 significant bits of the 16 bit parameter field are used to form the address. In this way a 12 bit address and a 16 bit parameter field may be extracted from one 24 bit word.

The second advantage of dividing words in the program memory into a tag and a parameter field is that it makes possible the extraction of instruction to be executed from a tree-structure code, i.e. code which is divided into a number of branches, with each branch terminating either in an instruction to be executed (known as a "primitive") or in a junction to other branches (known as a "secondary") from which depends one or more subprograms which may themselves contain several branches. In prior art processors the primitives are extracted from the tree by software known as an "inner interpreter". The division of a program instruction into a tag and a parameter field has make it possible to achieve the function of the inner interpreter in a hardware form, which operates much more rapidly than a software system. The extraction of primitives in the present invention is

achieved by a prefetch unit which is independently an aspect (the seventh aspect) of the present invention.

In essence the prefetch unit consists of the program memory, a control unit (which may be the same control unit which detects the tag which inhibits the look-up RAM as discussed above) and a stack, in which certain addresses of program instructions are stored. Normally the program computes the instruction in a sequence determined by the sequence of their addresses. This will usually be controlled by a program counter. However, when an instruction represents a secondary in the program, this is detected by the control unit and a suitable address, e.g. the address of the secondary itself, or the address of next instruction in the sequence, is stored in the stack. The program jumps to the first subprogram from the secondary and processing continues at that subprogram. If the first node in that subprogram is another secondary then again the address of the secondary or the address plus 1, is stored on the stack. When all the sub-programs depending from a secondary (whether only primitives, or including secondaries with further sub-programs) has been executed, this may be detected by a suitable tag on the last (primitive) instruction and the topmost address in the stack extracted and fed (adding one if this has not already been done) to the program memory. In this way the program may be made to return to an earlier position in the program, and the detection of all the primitives and secondaries is entirely achieved by the tag signal fed to the control unit. It makes use of the fact that each instruction contains a tag and a parameter field, because if they were separate instructions, as in the prior art, it would be impossible to determine which address to store in the stack without investigating several instructions, which would be slow.

An embodiment of the present invention will now be described in detail, by way of example, with reference to the accompanying drawings in which:

Figure 1 is a block diagram of a computer system according to the first aspect of the present invention;

Figure 2 is a timing diagram of various signals on the network bus of Figure 1;

Figure 3 shows the structure of an execution unit of the system of Figure 1;

Figure 4 shows a block diagram of the address generator module of an execution processor of the execution unit of Figure 3;

Figure 5 shows a schematic drawing of a treestructure program;

Figure 6 shows a block diagram of the preset unit of the execution processor;

Figure 7 shows a known system for distributing microcodes to a plurality of boards; microcodes to a plurality of boards;

Figure 8 shows a block diagram of a system according to the seventh aspect of the present invention for distributing microcodes to various boards;

Figure 9 shows a block diagram of the execution processor of the execution unit of Figure 3; and

Figure 10 shows a block diagram of an image recognition device according to the second aspect of the present invention;

BUS STRUCTURE.

Referring first to Figure 1, a computer system has a logical ring structure comprising a network bus 100 to which is connected a master unit 101, an input/output (I/O unit) 102 and a plurality of execution units 103. Three execution units are shown in Figure 1, but additional execution units may be provided up to a maximum of N-2 where N is the number of data lines in the network bus 100. Normally the network bus will have 16 data lines, so that a maximum of 14 execution units 103 may be provided.

A master unit 101, which may be a standard minicomputer e.g. plexus 35, is the unit which provides user input such as the programs to be performed by the various execution units 103 to the computer system. It also provides a start signal for initiating the processing by the other units, and may provide a monitor to check that the execution units 103 are processing data in the correct sequence, but otherwise plays no part in the processing of data by the rest of the computer system. The input/output unit is the point of entry of data to the system for processing by the execution units 103, and the point of exit of processed data.

The execution units 103 will be described in more detail later.

One aspect of the present invention concerns the bus structure of the computer system of Fig. 1. The network bus 100 comprises 30 lines, each connected in parallel to the master unit 101, the input/output unit 102, and the execution units 103. Sixteen of the lines of the network bus 100 are data lines, which in a conventional system would be used to transmit data from one unit to another. In the present invention the data lines are used for transmitting data, but also are used for indicating the status of each unit to which the network bus is connected. Each unit is assigned a corresponding data line which is to carry status signals from that unit. In the absence of other information on the data lines, each unit applies a signal to its assigned data line to indicate whether or not it is ready to receive data. Suppose that one unit is to transmit data to some of the other units. The transmitting unit checks the data lines of the other units to confirm that they are ready to receive data. Since each unit has a corresponding data line, the transmitting unit can perform this check simultaneously

(i.e. in parallel) for all the units to which it is to transmit. The unit transmitting the data then applies signals to the data lines of the units to receive the data, which signals enable the receiving units in parallel, and then the data is transmitted in parallel to all the enabled units. After the receiving units have received the data, each receiving unit signals on its assigned data line that it has received the data correctly, and this permits the transmitting unit to check that the data it has transmitted has been received by all the units to which that data was to be transmitted, this check again being in parallel for all the receiving units. Thus status checks, enabling signals, validation checks, and data transfer are all achieved in parallel, so that the checks take up as little time as possible, increasing the time available for data transfer, and hence increasing the efficiency of the bus.

This bus structure facilitates the use of the system as a token-passing ring. Token-passing rings are known and comprise a logical ring of interconnected processing units with a (notional) token which is passed between them. The processing unit with the token is enabled to transmit data to other processing units, and when it has finished transmitting, it passes the token to the next processing unit in the logical ring. That unit then transmits any data it has to transmit, and the token is again passed on. This continues until the token has been passed around the ring, completing a cycle for the system. Using the bus structure discussed above, the computer system of the present invention is particularly suitable for a token-passing system, because when the token is passed from one unit to another, the receiving unit can signal on its assigned data line that it has received the token, so that the control unit 101 can monitor that the token is being passed correctly between the units and that the token is not "dropped" (when a token is passed from one unit but is not received correctly by another unit). In this way the amount of time during which the signals on the bus represent token passing signals (during which time the bus is not available for data transmission) may be reduced relative to prior art arrangements.

In addition to the sixteen data lines, the network bus has 14 control lines divided into four address lines, four mode lines, four clock lines and two parity lines. The signals on these lines co-operate with the signals on the data lines to control the various steps of data transmission and token passing. The functions of the control lines are as follows:

address lines—during data transfer, the transmitting unit applies a signal to the data line corresponding to its address so that receiving units know which unit is transmitting;
—during token passing, the transmitting unit gives the address of the unit to receive the token;

mode lines—signals are applied to these lines by the transmitting unit only during data transfer, to give a signal indicating a characteristic of the data being transferred;

clock lines

a) bus busy—indicates data being transferred;
b) TK/strobe—provides clock pulses for data transfer;
c) control/strobe—during data transfer, a signal on this line from the transmitting unit enables the receiving units;
—during token passing, a signal on this line indicates the token is to be passed;
d) ack/strobe—during data transfer, a signal on this line from the transmitting unit indicates that all the data has been transferred and that the receiving units are to acknowledge data receipt;
—during token passing, a signal is applied to this line by the master unit if token transfer has failed;

parity lines—during data transfer, provide a check from transmitting to receiving unit for assisting validation.

Referring now to Fig. 2, the pattern of signals on the network bus 100 can be seen. Assume that one execution unit has the token and all other units are signalling on their assigned data lines that they are ready to receive (checked by the unit with the token). Assuming that it has data to transmit, the execution unit with the token applies a signal 201 to the bus busy (BB) line indicating that it is about to transmit data, applies a signal 202 to the data lines of the units to receive the data, which, together with an enabling signal 203 on the control/strobe (CTLSTB) line enables the receiving units, applies a signal 204 to the address (ADR) line corresponding to the address of the unit with the token so that the receiving units know which unit is transmitting, and applies a signal 205 to the mode lines indicating a characteristic of the data to be sent. If the unit has no data to transmit, it immediately starts the token passing sequence.

When data is to be transmitted, the data 206 is transmitted across the data lines accompanied by clock pulses 207 on the TX/strobe (TXSTB) line and parity pulses 208 on the parity lines. At the end of the data transfer the unit with the token applies a signal 209 to the ack/strobe (ACKSTB) line requesting acknowledgement of valid data receipt by the receiving units, which is achieved as described above by the receiving units each applying a signal 210 to their assigned data line. This completes data transfer and the bus busy (BB), address (ADR) and mode lines are cleared.

Then the unit with the token passes that token to another by applying a signal to the address lines indicating the address of the unit which is sent to receive the token, and applying a signal 212 to the control/strobe

(CTLSTB) line to clock the passage of the token. The unit receiving the token then applies a signal 213 to its assigned data line indicating that it has validly received the token, and token passing has been completed. If token passing is not performed correctly, and the token is dropped, the master unit 101 may apply a signal 214 to the ack/-strobe (ACKSTB) line to reset the token passing system. Once the token has been transferred, the unit now with the token waits until all units have indicated by a signal 215 on their assigned data lines that they are ready to receive data, then data transfer from the unit now with the token may commence.

As described above a network bus 100 with 16 data lines permits 14 execution units 103 since each unit must have a corresponding data line. If more were needed it would be feasible to link network bus rings to create an extra level of processing within the system, i.e. one of the execution units would be replaced by an interface to another network ring, which itself could have up to 14 execution units. The system is very efficient because the data lines are used for several purposes, decreasing the number of lines which would otherwise be required in the network bus 100. Furthermore, during data transfer, the signals on the bus are controlled by the unit with the token, i.e. the unit with the token acts as a bus controller whilst it has the token, and control of the bus is passed to another unit when the token is passed.

EXECUTION UNIT STRUCTURE.

The structure of an execution unit 103 will now be discussed in more detail with reference to Figure 3. As can be seen from that figure, the execution unit 103 consists of two processors, a control processor 310 (which may be a standard Motorola 68000 microcomputer) and a execution processor 302. The control processor 301 is connected by a transmission bus 303 to the network bus 100 discussed with reference to Fig. 1. The function of the control processor 301 is to control the transfer of data between the network bus 100 and the execution processor 302. It is the control processor 301 which:

(i) signals to the appropriate data line of the network bus 100 that the execution unit 103 is ready to receive information;

(ii) signals to other units that it is about to transmit and that they are to receive data;

(iii) transmits the data;

(iv) checks that data has been received correctly, and applies a suitable signal to the appropriate data line of the network bus 100;

(v) receives and passes the token of the logical ring.

Thus it is the control processor 301 of each execution unit 103 which interacts with the network bus 100 and with the rest of the computer system to achieve the advantages of efficient data transmission and signalling discussed above in connection with Fig. 1.

The control processor 301 also controls the transmission of data to and from the execution processor 302.

The control processor 301 and the execution processor 302 each have their own "working" memmories 304, 305 respectively, but in addition there is a "bank switch" memory 306 connected between them. The memory space of the bank switch memory 306 is divided into two areas 307, 308, each of which is (notionally) subdivided into two parts 307a, 307b, 308a, 308b during each processing operation. The use of a bank switch memory 306 with such subdivision permits simultaneous transmission of data from the control processor 301 to the execution processor 302 and vice versa. Of course, there need be no physical division of the bank switch memory 306, and the division may be a purely logical one of division of memory addresses in a single memory component. Normally, the addresses of the two areas 307, 308 will not change, but the addresses of each part of the area may be changed by the appropriate processor unit 301, 302 depending on the operations to be performed.

The program to be used on the computer system may be divided into a series of "frames" corresponding to the processing of a batch of information by the execution processor 302 of each execution unit 103. Assume that data to be processed by the execution processor 302 is stored in the right half 308b of the memory area 308 and that data to be transmitted by the control processor 302 to other parts of the computer system is stored in the left half 307a of the memory area 307. The "frame" begins with the execution processor 302 commencing to process the data in the right half 308b of memory area 308 and this continues until the data is fully processed and can be stored in the left half 308a of the memory area 308. Simultaneously with this processing by the execution processor 302, the control processor 301 transmits data from the left half 307a of memory area 307 to other units, and receives data from appropriate other units which is stored in the right half 307b of memory area 307. At the end of the processing by both processors 301, 302, the memory areas 307, 308 are "switched" (again a logical operation rather than actual movement of data) so that the control processor 301 has access to the memory area 308 and the execution processor has access to the memory area 307.

As described above the system may operate as a token-passing ring and during a token cycle the token is passed once around the ring. Consider now the operations of one particular execution unit 103 during a token cycle. At the start of the token cycle the unit

signals on its assigned data line that it is ready to receive data. When the token arrives at a unit (e.g. another execution unit 103) which is to transmit data to the execution unit 103 under consideration, that other unit signals on the appropriate data lines that it is about to transmit data thereby enabling the execution unit under consideration. A data packet (which may be all or only a part of the data that unit has to transmit) is then transmitted via the network bus 100, is received by the execution unit 103 under consideration via the bus 303 and is stored in the right half 307b of the memory area 307. Storing of data packets in the right half 307b of the memory area 307 continues as the token is passed around the ring. Thus during a token cycle the right half 307b of the memory area 307 receives packets of data which are to be processed by the execution processor 302 during the next "frame".

At some time during the token cycle the execution unit 103 under consideration will receive the token. It signals that it is to transmit data, thereby enabling the units to receive that data and then transmits a data packet from the left half 307a of the memory area 307 onto the network bus 100 and hence to other appropriate units. At the end of the transmission of the data packet it checks that the data packet has been received correctly by monitoring the signals on the data lines assigned to the receiving units and then it signals via the appropriate data line of network bus 100 that it has finished transmitting the data packet and the token is then passed on. Thus during the token cycle the control unit transmits a packet of data processed by the execution processor 302 during the previous "frame". The token is passed round and round the ring, and each execution unit with data to transmit will transmit a packet of that data each time the unit has the token. If the execution unit has no data to transmit, it simply passes on the token. After a sufficient number of token cycles, a control processor of an execution unit will have passed all the data processed by the execution processor during the previous "frame" of that execution unit and data is switched between the control processor and execution processor as will now be described.

The control unit 301 has suitable means for recognising when it has received all the input data and successfully transmitted all its output data. When the execution processor 302 finishes processing the data of that frame (from the right half 308b of memory area 308) it signals to the control unit 301 that it has finished, and requests more data. However, the control processor 301 will only respond to this request when it has received all input data and transmitted all output data. When this happens the memory areas are switched so that the control processor 301 has access to the memory area 308 and the excecution processor 302 has access to the memory area 307. Although this is, in fact, merely a change of addresses, it may be considered as a transfer of the data in the right half 308b by the memory area 308 (to form the input data for the execution processor 302 for the next frame) and of the data in the left half 308a of the memory area 308 to the left half 307a of memory area 307 (to form the output data to be transmitted by control processor 301). The execution unit 103 has then completed one frame and the control unit can signal that it is ready to receive data (i.e. is ready for the next frame to begin). The frame then begins when all the execution units 103 are ready to receive data. The division of the bank switch memory 306 into the memory areas 307, 308 means that neither processor 301, 302 contends with the other for access to the memory 306.

If the execution processor 302 is to handle particularly complex processes, it may be necessary for there to be more interaction between the control processor 301 and the execution processor 302 than described above. For example, the signal from the execution processor 302 to the control processor 301 to switch access to the memory areas 307 and 308 need not be at the end of a processing cycle by the execution processor 302, but the execution processor 302 may continue processing data after the data access has been switched.

EXECUTION PROCESSOR COMPONENTS

The execution processor 302 consists of a plurality of components all connected in parallel to a plurality of data lines. Many of the components are conventional, but some relate to various aspects of the invention and will now be described in detail. The architecture of the execution processor as a whole will be described later.

ADDRESS GENERATOR MODULE

An address generator module converts a data signal appearing as a data line of the processor to an address. However, if the memory has more memory addresses than possible signals on the data line, it is necessary to have a conversion system which effectively multiplies the number of data signals, and the problem is to convert data signals, which are random sequences of n bits to an ordered sequence of memory addresses of n + x bits (i.e. the memory is object addressable). So far as is known, there is no prior art system to do this. There are systems that can convert an address of n bits to an address of n + x bits, i.e. convert one ordered sequence to another, but none that can translate from the random sequence of data signals. One way of achieving this will now be described with reference to Fig. 4. Assume that a 16 bit

address signal on the data lines 400 is to be converted to a 20 bit memory address which is transmitted from the address generator module 401 via an address bus 402 to the memory 305 (see Fig. 3). If this conversion were not done, the total address space of the memory 305 and the area of the bank switch memory 306 to which the execution processor has access would be limited to 64K, but by increasing the number of bits in the address signals, a memory address base of 2 megabytes can be achieved. The address generator module consists of two translation units 404, 405 connected in parallel, via an adder 406, to the address bus 402. Two translation units 404, 405 are required because there are two different types of addresses with which the address generator module 402 must deal. One type of addresses are the addresses of static data objects i.e. those data with predetermined positions in the main memory 305, and the addresses of which are therefore known before a program is executed. Since the addresses of these data objects are known it is relatively easy to generate a 20 bit address for each data object. One of the translation units 404 acts as a static translation unit, and consists of a random access memory (RAM) which stores the addresses of the static data objects as 20 bit addresses and acts as a "look-up" table to convert each 16 bit address to a corresponding 20 bit address. Suppose that there are a maximum of 4K static data objects. A 12 bit signal fed from the data bus 400 can then be used to generate a complete set of unique addresses for each of the static data objects, and the RAM of the static translation unit 404 converts the 12 bit signals into 20 bit addresses for transmission to the address bus 402.

The other type of data objects stored in the memory 305 and all the data objects stored in the bank switch memory are dynamic data objects, i.e. data objects which are not predetermined and which may change during the program. practical programs require a large number of dynamic data objects, so that it is not practicable to use a RAM to store the addresses of all the dynamic data objects. Therefore the dynamic memory addresses must be generated directly from the data on the data bus 400. This is achieved by feeding a number of bits of the data signal to a control unit 407 which inhibits the static translation unit 404 and enables the other translation unit (the dynamic translation unit) 405. The dynamic translation unit 405 shifts the signal on the data line by up to 4 bits, to form a 20 bit address with the bits of the address not coresponding to a bit of the data signal being set to zero. The shifted signal, now being a 20 bit signal is fed to the address bus 402, via adder 406.

There is a difficulty with this however. As the most significant bit (MSB) of the signal on the data line 400 is shifted towards the MSB of the memory address, the memory that can be addressed increases, but the memory has to be allocated in larger blocks. In order that the available dynamic address space is used efficiently, it is desirable that the program controlling the dynamic memory allocation knows by how many bits the dynamic addresses are being shifted.

It is convenient if the addresses of the static data objects correspond to the bottom 4K of the 16 bit address line 402. The top 60K can then be used for the addresses of dynamic data objects. Assume a signal appears on the data bus 400 which is to be converted to an address on the address bus 402. The top 4 bits of the 16 bit signal are fed to the control unit 406. If these top 4 bits are all zero, i.e. the signal is in the bottom 4K, the control unit 406 enables the static translation unit 404 which receives the bottom 12 bits of the signal from the data bus 400. The RAM of the static translation unit 404 converts this 12 bit signal to a 20 bit address which is fed via line 407, and the adder 403 to the address bus 402. If, on the other hand, any one of the 4 top bits of the signal on data bus 400 is non-zero, the control unit 407 inhibits the static translation unit 404 and enables the dynamic translation unit 405 to receive the 16 bit signal on the data line 400. The dynamic translation unit 405 then shifts the 16 bits upwards to create a 20 bit signal which is again fed via line 407 and adder 406 to the address bus 310.

As shown in Figure 4, the adder 403 may combine the signal from the static or dynamic translation units 401, 402, with a signal from an offset unit 408. The purpose of the offset unit 408 is to permit the generation of the addresses of vectors, i.e. quantities for which more than one parameter, and thus more than one address is necessary to define the quantity. When the addresses of a vector are to be generated, first one address is generated as described above then a signal is applied to a data bus 409 which is fed via line 410 to the offset unit 408 which calculates the difference between the initial address and subsequent address, and applies that difference to the adder 406, which sums it with the initial address thereby deriving the subsequent address in a simple way.

PROGRAM MEMORY AND PREFETCH UNIT

One way of structuring a computer program is known as tree-structuring. Such a structure is shown in Figure 5 and consists of an entry point 501 which is connected to other points or nodes, which may themselves be branching points, or "secondaries" 502 with depending sub-programs or "subtrees" or may represent a single subroutine of the program. Such a sub-routine is known as an executable primitive. When such a program is run, each node

is scanned in turn and the subtrees (if it is a secondary) from that node investigated, again in turn, until all primitives have been extracted. First for example starting at the entry point 501, the secondary 502a would be scanned, and the first subtree from that secondary leads to another secondary 502b. Investigating the branches from the nodes 502b, the first is primitive 503a which would then be extracted for subsequent execution by the computer system. The next subtree goes to another secondary 502c, and the subtres from that secondary would be investigated. This would lead to the extraction of primitives 503b and 503c. Since all the subtrees from secondary 502c would then have been investigated, processing returns to secondary 502b to extract the primitive 503d. Processing then returns to secondary 502a for the extraction of primitive 503e, and then processing returns to the entry point 501 for processing of another branch from that entry point 501. The sequence of investigating the subtrees from each secondary is normally controlled by software known as an "inner-interpreter". However, in many programs the ratio of secondaries 502 to primitives 503 is high so that a considerable amount of processing time is spent simply in transversing the "tree" of secondaries looking for primitives 503 to extract.

Therefore it is desirable that there is a way for extracting the primitives more rapidly than could be done by programming alone. Normally in a tree-structure code, each word in the program memory may represent either an instruction (known as a parameter field) to be acted upon by other components of the processor or a tag which is associated with one or more parameter fields and indicates the nature of the parameter field e.g. primitive or secondary. Since the parameter field and tag are separate program words, it is necessary to extract from the program memory first the tag then the associated parameter field(s), so that at least two processing steps are needed.

The present invention makes use of a program memory which has a word length longer than the word length of the parameter field. The extra bits of the memory word are then used to form the tag, so that the tag and parameter field are combined in a single memory word.

Referring now to Fig. 6, a memory 600 with a word length of e.g. 24 bits forms the heart of both an instruction format unit 601 and a prefetch unit 602. The prefetch unit 602 acts as a hardware "inner interpreter" and will be described first. On a signal from a program counter 603, the program memory 600 outputs an instruction word, which may correspond either to a secondary of the program, in which case the parameter field is an instruction to obtain another instruction word, or to a primitive in which case the parameter field is an instruction for other parts of the processor.

If the program was not branched, it would consist simply of a string of primitives ordered in the sequence in which they are to be performed. A program counter 603 would send out a sequence of signals instructing the program memory to output the primitives in the correct order. However, in branched code this cannot be done because at a secondary the program must jump to an instruction in one subtree from that secondary, but be capable of returning to the secondary for executing the other subtrees from that secondary. Therefore when secondary occurs, the program jumps to one subtree but remembers the address of the next node (secondary or primitive) to enable the program to return when the one subtree has been completed.

This is achieved by adding one to the address of a secondary in a adder 604 and storing the result in a stack 605. The node is detected by a control unit 606 which receives the tag of the instruction output from the program memory 600 and is capable of distinguishing tags representing primitives, tags representing secondaries, and special tags representing "return" primitives which are the end of a secondary subtree (i.e. are the rightmost instruction at any particular level in any secondary subtree of the tree of Fig. 5). The return primitives may be simply an instruction to return to the next level or may be both a return instruction and an instruction to be transmitted to other parts of the processor. When a subtree has been executed, the end of the subtree is detected by the control unit 606, and the top address in the stack 605 is removed from the stack and output via a multiplexer 607 to the program counter 603 and becomes the next address fed to the program memory 600. It is important that each instruction word consists of both a tag and a parameter field because then the tag and the parameter field are produced in a single output from the memory. If the tag and the parameter field were separate words, as in the prior art, it would not be possible to know which step of the program to return to without investigating several words, which would be slow and inefficient.

Consider the tree of Fig. 5, in which the letters A to Y represent the sequence of instruction words stored in the program memory. The first instruction word to be output from the program memory 600 is secondary A. The control unit detects that it is a secondary, causes the address plus one (i.e. B) to be stored in the stack 605 and the program counter 607 is caused via information from the parameter field of the instruction word to jump to instruction E. As this is also a secondary, its address plus one (F) is stored above address B in the stack 605 and programming jumps to address M. This is a primitive, the

tag of which is detected by the control unit 606, and so the instruction word is fed to the instruction format unit 601. processing then continues with the instruction word at address N, which is a secondary so its address plus one (0) is stored at the top of a stack and the program jumps to address U, which is a primitive so is output to the instruction format unit. Then the next instruction word at address V is output and again this is a primitive and so is output to the instruction format unit. However, it is also the rightmost instruction at that level in that subtree, and therefore has a tag which instructs the control unit to extract the topmost address (i.e. 0) from the stack 605 and this then forms the next address fed to the program memory. Again this is a primitive and the rightmost in that subtree at that level so it is output to the instruction format unit 601 and the next address (F) extracted from the stack 605. This continues until all the instruction words have been output from the memory.

As described above, one (1) is added to the secondary address before it is stored in the stack, so that processing can return to the instruction word immediately after that secondary. It would alternatively be possible to store the address of the secondary itself on the stack, and add one when the address is output to the program memory.

The prefetch unit 602 thus steps through the program and extracts the primitives of the program and feeds them in the sequence in which they are to be performed to the instruction format unit. The prefetch unit operates asynchronously with the rest of the execution processor 302 so that the primitives may be "queued" for use at an appropriate time.

Consider now the instruction format unit 601. This receives the primitives from the program memory and each primitive consists of a tag (of e.g. 8 bits) and a parameter field (of e.g. 16 bits). The 8 bit tag is sufficient to define 256 instructions which can use the parameter fields in any way required. However, 256 instructions are not sufficient for many programs, and therefore it is necessary to derive other instructions. This is achieved by feeding the 8 bit tag both to a look-up memory 608, via a latch 609, and also to the control unit 606 which is common as to both the prefetch unit 602 and the instruction format 601. The look-up memory 608 acts in a similar way to the RAM of the dynamic translation unit 404 so that all but one of the 256 tags are converted to a 12 bit address by the look-up memory 608 and are fed to a multiplexer 610. At the same time the tag fed to the control unit 606 causes the control unit 607 to enable the multiplexer 610 to pass the 12 bit address from the look-up memory 609 direct to an instruction buffer 611. However, the one other tag causes the control unit 606 to prevent any address being fed from the look-up unit 609, but instead obtains the 12 bit address from the 12 least significant bits of the 16 bit parameter fields being fed on line 612 from the program memory 600 to the instruction buffer 611 via the latch 609. These 12 least significant bits then become the address signal fed to the buffer 611.

In this way a 24 bit instruction word in the program memory 600 may be used to specify a 12 bit address and a 16 bit value, and to permit maximum use to be made of the 12 bits of the address, so that 4096 addresses may be obtained.

Thus the output buffer 611 stores a 12 bit address and a 16 bit parameter field for each primitive extracted from the program, and the primitives are queued in the buffer 611 in the order in which they are to be performed.

HARDWARE STRUCTURE

Each of the various components of the execution processor 202 corresponds to a combination of hardware, mounted on a series of circuit boards. It is convenient for simplicity, to think of each component being mounted on a separate board, but in practice this need not be the case, and it has been found that the various components can be fabricated on only three circuit boards.

However, for the sake of simplicity assume for the moment that each part of the execution processor is on a separate board. It is then necessary to distribute the control signals from the microprogram memory of the execution unit 302 to the various other boards. The prior art method of doing this would be to buffer the control signals, then transmit them via a bus which interconnects the various boards and is known as a "backplane". Thus referring to Figure 7 a series of boards 701, 702, 703 contain circuit components, generally indicated at 704, 705, 706 respectively, connected to a data bus 707. One of the boards 701 contains the microprogram memory 708 in which the microprogram to be run on the boards is stored. A sequencer 709 controls the output of microprogram instructions from the microprogram memory 708. Each instruction of the microprogram memory consists of a string of bits divided into a plurality of fields, with each field being to control a component of the processor. thus one field may be used as a control signals to the circuitry 704 of the board 701 containing the microprogram memory 708, whilst the other fields are fed into the backplane 710. One of the fields in the backplane is fed as control signals to the circuitry 705 of the board 702, a further field to the circuitry 706 of the board 703, leaving e.g. one more field for one further board. It can be seen immediately that this limits the number of boards that may be interconnected in this way since the signals transmitted by the backplane 710 is limited by the bit length of the instructions in

the microprogram memory 708. Therefore if the system is to permit increase in the number of boards, "spare" capacity must be included in the word length (to add extra fields), and the size of the microprogram memory 708 must be sufficiently large in order to permit this. It is clearly undesirable to include "spare" capacity initially or to have the number of boards limited, and hence an aspect of the present invention seeks to overcome this problem.

Referring to Figure 8, three boards 801, 802, 803 each have circuitry 804, 805, 806 connected to a data bus 807. One of the boards 801 has a sequencer 809 which generates address signals which are to be fed to the microprogram memory. However, unlike the prior art system, each board 801, 802, 803 has its own microprogram memory 811, 812, 813 which may each contain the full microprogram memory required, or the microprogram memory required for that board only. The addresses from the sequencer 809 are fed to the microprogram memory 811 of that board 810 for controlling the circuitry 804, but also to a microprogram address bus 814. This microprogram address bus 814 is then connected to each microprogram memory 812, 813 of the boards 802, 803 in parallel. Thus, when an address is generated by the sequencer 809 it is fed in parallel to the microprogram memories 811, 812, 813 of each board 801, 802, 803, thus extracting the corresponding instructions from each microprogram memory 811, 812, 813 so that the instructions may then be acted upon by the circuitry 804, 805, 806 of one or more of the boards 801, 802, 803.

It can be seen immediately that it is simple to increase the number of boards, merely by connecting the microprogram memory of that board to the microprogram address bus 814. It is therefore unnecessary to include spare capacity in the existing memory to permit increase in the number of boards, and the size of the microprogram address bus does not increase with the number of boards, because it merely carries the address signals.

VARIABLE CLOCK

Each execution unit 103 has a clock which controls the parallel operation of the various components of the execution processor. It is clear that in some of the operations the execution processor 202 will execute will be shorter than others. However, with a fixed frequency clock, the processing speed is entirely determined by the clock rate, and no time advantage can be gained from operations which are shorter than the clock rate. It is known, in prior art processors, to employ a variable frequency clock the cycle length of which can be changed to suit the operations being performed during any particular cycle. However, the variable frequency clocks which are known cannot react automatically to the periods of the various operations within the processor and thus it is necessary for the programmer to calculate the duration of every operation manually. For processors of any complexity this is virtually impossible. Therefore the present invention seeks to provide automatic variation of the clock frequency independent upon the operations being performed.

To explain the way this is achieved, it is necessary to consider the operation of an assembler. An assembler converts an input code into a microprogram word, with the microprogram word being a string of fields, each field being pre-programmed instruction for some part of the processor. In the prior art, the input of a code word generates a plurality of fields, one of which is a field representing the length of the longest operation within that microprogram word. Since all the fields are pre-programmed, this means that the length of longest operation must be pre-calculated.

In the present invention, each field contains information relating to the length of operation of that field. When assembling a microprogram the assembler compares the information of the length of operation of each field, and from that information generates a "clock" field representing the maximum operation length and this forms part of the microprogram word. That clock field is then fed to the clock which regulates the clock time in dependence on the clock field set. Thus the system differs from the prior art in that no clock fields representing the total time of the microword are stored, each field stored contains information relating to the length of the operation represented, and the assembler calculates the clock field from the information from other fields. The calculation may simply involve calculation of the longest time if the operations are all in parallel, but some operations may be serial or may themselves represent subprograms in which operations occur in parallel, in series, or both. In such circumstances it is desirable that the assembler is able to analyse information about the subprograms so that the duration of the microword may be calculated accurately.

In this way the operation which determines the maximum duration of the operations represented by the microword during any one cycle of the processing of the execution processor 302 can be determined, and the clock control set automatically.

EXECUTION PROCESSOR STRUCTURE

Fig. 9 shows the general structure of the execution processor 302. It consists of three data buses 901, 902, 903 connected in parallel to the various components of the processor. Some of these components have already been described and it is asssumed

that each component is fabricated in a separate board so that each component has a separate microprogram memory and the sequencer 809 is connected to the microprogram memory of each component via the microprogram address bus 814 (the two parts of the microprogram address bus 814 shown in Fig. 9 being interconnected). Thus each component corresponds to a board 801, 802, 803 of the hardware of Fig. 8. However as discussed in connection with Fig. 8 it is usually possible to combine several components on one board so that the structure of Fig. 9 may be achieved in three boards. The address generator module (AGM) 401 has already been described in connection with Fig. 4, with two of the buses 901, 902, 903 corresponding to the buses 400, 409 of that figure. Since the AGM 401 is connected to all three buses 901, 902, 903 may the bus 400 may correspond to any one of them, and the corresponding bus may be changed during the operation of the processor.

The component marked PREFETCH has already been described because this corresponds to the instruction format unit 601 and the prefetch unit 602. Thus the component marked PREFETCH contains the program memory and the main data memory 305 is also shown. In addition the data buses 901 will be connected to the bank switch memory 306 but this is not shown.

The other components shown in Fig. 9 are more conventional. The constant source 904 provides constants for the rest of the processor, which constants may be desired directly from the microprogram or via the output of the prefetch unit. The memory I/F (MEM I/F) 905 acts as a short-term memory and buffer to allow byte swapping and byte storage during transactions with the memory 305. The cache 906 acts as a rapid access memory which can be addressed in a number of ways, the data stack (DS) 907 stores a stack of data values, the top two of which can be accessed simultaneously, or can be written into simultaneously using two of the data buses 901, 902, 903. Finally the components 408 and 409 marked MAC and ALU are the multiplier/accumulator and the arithmetic and logic unit respectively. It is not necessary to discuss these components in detail as their function will be understood to those skilled in the art.

IMAGE RECOGNITION An example of the application of a computer system as described above to an image recognition device will now be described with reference to Figure 10. The image recognition device comprises a series of stages, with data being passed in a pipeline operation from one stage to the next. It is assumed that the device has six execution units 103 connected to the network bus 100 of Figure 1.

A video input of a scene containing the image to be recognised is fed to a patch extractor 1001, forming the first stage of the image recognition device, which extracts a 64 × 64 pixel patch. The data in this patch is fed via the I/O unit 102 to three execution units 1002, 1003 and 1004 which each receive the data of the 64 × 64 patch and from the second stage. A first one of these execution units 1002 analyses the patch, looking for regions with a single axis of symmetry (e.g. edges and lines) whilst a second execution unit 1003 looks for regions with no symmetry or with complex symmetry. The program which detects these symmetry features is preprogrammed within the appropriate execution units. The third execution unit 1004 carried out extraction analysis on the patch, again the program for this being stored within the third execution unit 1004. The operations of edge/line extraction, no-symmetry extraction, and texture extraction are all known in image processing, but the structure of the execution units of the computer system of the present invention permit operations to be made more rapidly than by known processors.

Since the operations performed by the first and second execution units 1002, 1003 is simpler than that of the texture extraction operation of the third unit 1004, they are likely to finish first. This gives time for them to compare the results they have found with a previous analysis to look for movement, before all three execution units 1002, 1003 and 1004 output their process data to the next stage of the image recognition device. In that next stage, the output of the first and second execution units 1002, 1003 are fed to a fourth execution unit 1005, which analyses edges, lines and other features of the patch and forms line segments, arcs, and nearness relations. The output of the third processor 1004 is fed to a sixth processor 1006 which carries out "region growing" using the texture information from the unit 1004.

Once the execution units 1005 and 1006 have completed their analysis, their outputs are fed to the fourth stage of the image recognition device which comprises a sixth execution unit 1007. This compares the results obtained with various shape models, using the nearness information to dictate a problem solving strategy. This enables a description of the object to be built in absolute co-ordinates, so that the unit 1007 may then determine which is the next part of the image to be scanned. It then transmits a signal via line 1008 to patch extractor 1001 to extract another patch for further processing. At the same time it outputs via a line 1009 the results of its analysis, and the sequence of outputs from the execution unit 1007 on the line 1009 will build up an image of the objection being viewed by the video system.

Thus the device has both parallel, and pi-

peline features. The processing by the three execution units 1002, 1003 and 1004 is carried out in parallel, as is the processing by the execution units 1005 and 1006. However, the movement of information between each stage represents a pipeline operation, with the transfer of information between the various execution units being achieved in the way described in connection with Figure 1.

CLAIMS

1. A computer system having a plurality of execution units connected to a network bus, the network bus having a plurality of data lines with each execution unit connected to all the data lines, wherein each execution unit is associated with one of the data lines, and each execution unit is adapted to transfer data to one or more of the other execution units via the data lines, and also to signal its status by applying a signal to its associated data line.

2. A computer system according to claim 1 including a master unit connected to each of the data lines for monitoring the status of each execution unit by detecting the signals on the data line associated with each execution unit.

3. A computer system according to claim 1 or claim 2, wherein each execution unit has two processors and a memory connected to each processor, the memory being divided into two areas each of which is accessible to each processor, wherein switching means controls the access of each processor to each memory area such that the access alternates between:

a) a first state in which one processor has access to one memory area and the other processor has access to the other memory area; and

b) a second state in which the said one processor has access to the said other memory area and the said other processor has access to the said one memory area.

4. A computer system according to claim 3, wherein the said one processor is connected to a network bus, during the first state the said one processor is adapted to receive signals via the network bus and store them as data in a first part of the said one memory area, and to transmit data from a second part of said one memory area via the network bus, and the said other processor is adapted to analyse data in a first part of said other memory area, and to store data corresponding to the analysed data in a second part of said other memory, and during said second state the said one processor is adapted to receive signals via the network bus and to store them as data in the first part of the said other memory area, and to transmit data from the second part of said other memory area, and the said other processor is adapted to analyse data in the first part of said one memory area, and to store data corresponding to the analysed data in the second part of said one memory area.

5. A computer system according to any one of claims 1 to 4 having an input/output unit connected to each of the data lines for receiving signals from outside the system and transmitting them to at least one of the execution units, and for receiving signals from at least one of the execution units and transmitting them to the outside of the system.

6. An image recognition device having a computer system according to claim 5, and an image detector connected to the input/output unit, wherein each execution unit of the computer system is adapted to carry out image recognition operation.

7. A method of operating an image recognition device according to claim 6, wherein the input/output unit first transmits an image signal to a first execution unit for processing according to the appropriate image recognition operation of that unit, the first execution unit transmits the processed image signal simultaneously to a plurality of second execution units for further processing according to the appropriate image recognition operations of these units, and the second execution units transmit the further processed signal simultaneously to a plurality of third execution units for processing according to the appropriate image recognition operation of those third units.

8. A method of operating an image recognition device according to claim 7, wherein the output of the third execution units is fed to at least one fourth execution unit for processing according to the appropriate image recognition operation of the fourth execution unit(s), the completion of processing of that input by the fourth execution unit(s) causing another image signal to be transmitted from the input/output unit to the first execution unit.

9. An execution unit having two processors and a memory connected to each processor, the memory being divided into two areas each of which is accessible to each processor, wherein switching means controls the access of each processor to each memory area such that the access alternates between:

a) a first state in which one processor has access to one memory area and the other processor has access to the other memory area; and

b) a second state in which the said one processor has access to the said other memory area and the said other processor has access to the said one memory area.

10. An execution unit according to claim 9, wherein the said one processor is connected to a network bus, during the first state the said one processor is adapted to receive signals via the network bus and store them as data in a first part of the said one memory area, and to transmit data from a second part of said one memory area via the network bus,

and the said other processor is adapted to analyse data in a first part of said other memory area, and to store data corresponding to the analysed data in a second part of said other memory, and during said second state the said one processor is adapted to receive signals via the network bus and to store them as data in the first part of the said other memory area, and to transmit data from the second part of said other memory area, and the said other processor is adapted to analyse data in the first part of said one memory area, and to store data corresponding to the analysed data in the second part of said one memory area.

11. A method of converting a data signal having a first number of bits to an address signal having a second number of bits, comprising:

determining the spacing between the least-significant-bit and the most-significant-bit of the data signal;

when that spacing is less than or equal to a predetermined number of bits, converting the data signal to a corresponding address signal of a greater number of bits; and

when the spacing is greater than the predetermined number of bits, shifting the bits of the data signal away from the least significant bit by n positions (where n is an integer) to form part of the address signal and setting to zero any of the bits of the address signal not corresponding to a bit of the data signal.

12. A method according to claim 11 wherein the determination of the spacing is achieved by detecting if any of the bits of the data signal spaced from the leastsignificant-bit by more than the predetermined number of bits is non-zero.

13. A method according to claim 11 or claim 12 wherein the conversion of a data signal to a corresponding address signal is carried out by storing address signals in a memory with each memory location corresponding to one of the data signals, such that the memory generates one of the address signals on receipt of the corresponding data signal.

14. A computer system fabricated on a plurality of boards, wherein each board has a microprogram memory for controlling the electronic components on the corresponding board, and one board has a microprogram address generator connected in parallel to all the microprogram memories for transmitting microcode addresses to all of the microprogram memories simultaneously.

15. A computer system according to claim 14, wherein the microprogram address generator is a sequencer.

16. A method for controlling the clock rate of a computer system comprising:

supplying a plurality of fields to an assembler for assembly into a microword, each of the fields containing information relating to the duration of that field;

determining, in the assembler, the deviation of the longest field and generating a clock field corresponding to that duration; and

controlling the clock rate on the basis of the clock field so calculated.

Printed in the United Kingdom for
Her Majesty's Stationery Office, Dd 8818935, 1986, 4235.
Published at The Patent Office, 25 Southampton Buildings,
London, WC2A 1AY, from which copies may be obtained.