7 Instructions

The instruction set gives the programmer the ability to carry out all the functions necessary to program the system efficiently and may be divided into ten basic groups:

Load/Store Instructions
Arithmetic Instructions
Logical Instructions
Character Instructions
Branch Instructions
Shift Instructions
Control Instructions
Input/Output Instructions
External Transfer Instructions
Move Table Instructions

Within these groups efficiency is ensured by the possible use of up to eight different methods of forming one of the instruction's operands, the method to be used being chosen by the programmer with reference to the memory and timing requirements of any particular program.

Two formats for instruction layouts are used and where necessary two words are used to define an instruction.

INSTRUCTION FORMATS

Two instruction formats are possible and these are defined within the instruction by the most significant bit, bit 0, of the instruction word. Where instructions consist of two words the format bit is the most significant bit of the first word only.

Format 0 instructions are always short, that is one word. Format 1 instructions may be short or long, one or two words.

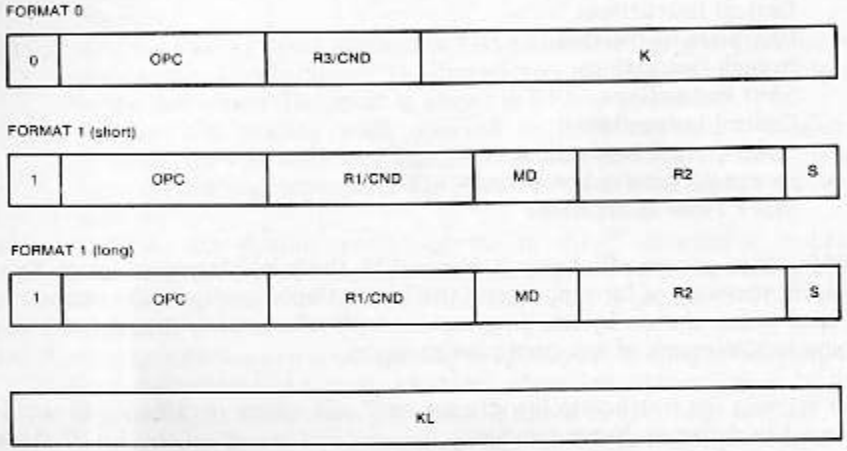


Figure 7.1 Layout of instruction formats.

OPC 4 bits, the pattern of which defines the instruction to be carried out.

R1 4 bits, specifies the working register to be used by the instruction, A0 - A15. It may contain one of the operands to be used and may also be used to hold the result of the instruction. In certain cases with R1 = 0 the addressed register, the P register, will not be used and in these cases R1 = 0 will qualify the operation code and define a different instruction than when R1 = 0.

R2 4 bits, specifies the second working register to be used by the instruction A1 - A15. It may contain the second operand or hold an address to be used in forming this operand. If R2 is made zero, no second working register is specified but this condition is used in deciding the method of forming the second operand.

R3 3 bits, specifies the working register to be used by the instruction A0 - A7. It may contain one of the operands to be used and may also be used to hold the result of the instruction. In certain cases with R3 = 0 the addressed register, the P register, will not be used and in these cases R3 = 0 will qualify the operation code and define a different instruction than when R3 = 0.

CND 3 bits, specifies the condition which must exist for a particular instruction to be carried out. Used to qualify conditional branch instructions and replaces R3 or the most significant 3 bits of R1.

MD 2 bits, specifies the mode of addressing to be used when forming the second operand of an instruction where this is applicable.

S 1 bit, applicable to certain instructions using memory. When present it specifies that the result of the instruction concerned is to be stored in the memory address specified by the instruction. When this bit is not present the result is placed into the working register specified by R1.

K 8 bits, these bits are used to specify the operand in format 0 instructions, and include short constant operands (k) and short displacements (m-for relative branch instructions). This field is also used to specify counts for shift instructions (n) and device addresses to I/O instructions (dev), in these cases a part of the field may be used to qualify the operation code.

KL 16 bits, this field is made up of the complete second word of a double length instruction and may specify a long constant (KL) or an address (m).

FORMING THE OPERAND

Many of the instructions may use various methods of forming one of the operands to be used. In all, eight methods of forming an operand are available governed by the values of the Format, Mode, and R2 fields of the instruction layout.

Figure 7.2 lists the eight methods of forming an operand and a brief description of each method is given following the figure.

Type	Format	Mode	R2	
T1	1	00	–	Reg/Reg.
T2	1	01	R2 = 0	Long Constant
T3	1	01	R2 ≠ 0	Address in Reg R2
T4	1	10	R2 = 0	Address in next word
T5	1	10	R2 ≠ 0	Indexed
T6	1	11	R2 = 0	Indirect
T7	1	–	R2 ≠ 0	Indexed Indirect
T8	0	–	–	Short Constant

Figure 7.2

T1. Register/Register - Format 1 (short)
The operand is the value in the register specified by R2 of the instruction format.

T2. Long Constant - Format 1 (long)
The operand is the value in the least significant word, all sixteen bits, of the double length instruction format.

T3. Address in Register - Format 1 (short)
The operand is held in memory. The memory address of the operand is the value in the register specified by R2 of the instruction format.

T4. Address in Next Word - Format 1 (long)
The operand is held in memory. The memory address of the operand is the value in the least significant word of the double length instruction.

T5. Indexed Address in Next Word - Format 1 (long)
The operand is held in memory. The memory address of the operand is found by adding the value in the register specified by R2 of the instruction format to the value in the least significant word of the double length instruction.

T6. Indirect Address in Next Word - Format 1 (long)
The operand is held in memory. The memory address of the operand is also held in memory. This indirect address is the value in the least significant word of the double length instruction.

T7. Indexed Indirect Address in Next Word - Format 1 (long)
The operand is held in memory. The memory address of the operand is also held in memory. This indirect address is found by adding the value in the register specified by R2 of the instruction format to the value in the least significant word of the double length instruction.

T8. Short Constant - Format 0
The operand is the value in the least significant eight bits of the instruction format.

INSTRUCTION TIMING

The timing of the instructions depends on various factors: the type of instruction itself, the memory, the method of forming the operand and the number of memory cycles required.
The instruction set offers the possibility of very rapid execution times where single word register/register or short constant operations are employed whilst the more complex register/memory instructions save execution time when compared with the routines they may replace.
Execution time is also reduced in the case of conditional instructions by carrying out the conditional check immediately after accessing the instruction and then only continuing if the required conditions are satisfied.

TRAP ACTION

The use of any invalid instruction causes the activation of the Trap action which consists of the following basic actions:
- the CPU does not attempt to carry out the instruction
- information with reference to the instruction address and processor status is saved in the stack
- interrupts are inhibited
- a user mode flag is reset when working in user mode
- an indirect branch is made to address /7E for a trap routine.

no RIT, automatically deactivated

THE INSTRUCTION SET

The instructions within the basic groups, together with their mnemonic, addressing type(s) and the execution time for the different types of memory are listed here:

Load/Store Instructions		Addressing types	Execution times in µs	
			1.2 µs memory	0.7 µs memory
LD	Load	T4 - T7	3.7 - 5.0 µs	2.2 - 3.0 µs
LDR	Load Register	T1, T3	1.4 - 2.5 µs	1.2 - 1.8 µs
LDK	Load Constant	T8, T2	1.3 - 2.5 µs	0.9 - 1.5 µs
ST	Store	T4 - T7	3.8 - 5 µs	2.4 - 3.3 µs
STR	Store Register	T3	2.8 µs	2.1 µs
ML	Multiple Load	T4 - T7	2.8 - 4.1 + nx1.3 µs	2.6 - 3.5 + nx0.8 µs
MLR	Multiple Load Register	T3	2.0 - 2.4 + nx1.3 µs	1.9 - 2.3 + nx0.8 µs
MLK	Multiple Load Constant	T2	2.9 + nx1.3 µs	2.7 + nx0.8 µs
MS	Multiple Store	T4 - T7	2.8 - 4.1 + nx1.3 µs	2.6 - 3.5 + nx0.8 µs
MSR	Multiple Store Register	T3	2.5 - 3.1 + nx1.3 µs	2.3 - 2.9 + nx0.8 µs
EL	Extended Load (MMU)	T4 - T7	3 - 4.1 µs	2.4 - 3.3 µs
ELR	Extended Load Register (MMU)	T3	2.5 µs	2.1 µs
ES	Extended Store (MMU)	T4 - T7	3 - 4.1 µs	2.4 - 3.3 µs
ESR	Extended Store Register (MMU)	T3	2.5 µs	2.1 µs
Arithmetic Instructions				
AD	Add	T4 - T7	3.8 - 6.3 µs	2.2 - 3.9 µs
ADR	Add Register	T1, T3	1.4 - 3.8 µs	1.2 - 2.6 µs
ADK	Add Constant	T8, T2	1.3 - 2.5 µs	0.9 - 1.5 µs
SU	Subtract	T4 - T7	3.8 - 6.3 µs	2.2 - 3.9 µs
SUR	Subtract Register	T1, T3	1.4 - 3.8 µs	1.2 - 2.6 µs
SUK	Subtract Constant	T8, T2	1.3 - 2.5 µs	0.9 - 1.5 µs
MU	Multiply	T4 - T7	9.7 - 11 µs	8.6 - 9.5 µs
MUR	Multiply Register	T1, T3	7.8 - 8.5 µs	7.6 - 8.9 µs
MUK	Multiply Constant	T2	8.5 µs	7.9 µs
DV	Divide	T4 - T7	10 - 11.3 µs	8.8 - 9.5 µs
DVR	Divide Register	T1, T3	7.8 - 8.8 µs	7.6 - 8.9 µs
DVK	Divide Constant	T2	8.8 µs	8.2 µs

		Addressing types	Execution times in µs	
			1.2 µs memory	0.7 µs memory
DA	Double Add	T4 - T7	5.6 - 6.9 µs	3.9 - 4.8 µs
DAR	Double Add Register	T1, T3	3.1 - 4.5 µs	3.0 - 3.6 µs
DAK	Double Add Constant	T2	4.4 µs	3.2 µs
DS	Double Subtract	T4 - T7	5.6 - 6.9 µs	3.9 - 4.8 µs
DSR	Double Subtract Register	T1, T3	3.1 - 4.5 µs	3.0 - 3.6 µs
DSK	Double Subtract Constant	T2	4.4 µs	3.2 µs
C2	Two's Complement	T4 - T7	5.3 - 6.5 µs	3.5 - 4.4 µs
C2R	Two's Complement Register	T3	4.0 µs	3.1 µs
IM	Increment Memory	T4 - T7	5.0 - 6.3 µs	3.0 - 4.0 µs
IMR	Increment Memory Register	T3	3.8 µs	2.6 µs
NGR	Negate Register	T1	2.0 µs	1.9 µs
CM	Clear Memory	T4 - T7	3.8 - 5.0 µs	2.4 - 3.3 µs
CMR	Clear Memory Register	T3	2.8 µs	2.1 µs
CW	Compare Word	T4 - T7	3.8 - 5.0 µs	2.2 - 3.0 µs
CWR	Compare Word Register	T1, T3	1.4 - 2.5 µs	1.2 - 1.8 µs
CWK	Compare Word Constant	T2	2.5 µs	1.5 µs
FFL	Integer to Floating Point	T1		3.7 µs
FFX	Floating Point to Integer	T1		5.1 µs
FAD	Floating Point Addition	T4-T7		6.2 - 9.6 µs
FADR	Floating Point Addition/Register	T3		5.9 - 8.4 µs
FSU	Floating Point Subtract	T4-T7		6.2 - 9.6 µs
FSUR	Floating Point Subtract/Register	T3		5.9 - 8.4 µs
FMU	Floating Point Multiply	T4-T7		8.8 - 12.2 µs
FMUR	Floating Point Multiply/Register	T3		8.4 - 11.0 µs

		Addressing types	Execution times in µs	
			1.2 µs memory	0.7 µs memory
FDV	Floating Point Division	T4-T7		8.8 - 12.2 µs
FDVR	Floating Point Division/Register	T3		8.4 - 11.0 µs
Logical Instructions				
AN	Log. AND	T4 - T7	3.8 - 6.3 µs	2.2 - 3.9 µs
ANR	Log. AND Register	T1, T3	1.4 - 3.8 µs	1.2 - 2.6 µs
ANK	Log. AND Constant	T8, T2	1.3 - 2.5 µs	0.9 - 1.5 µs
OR	Log. OR	T4 - T7	3.8 - 6.3 µs	2.2 - 3.9 µs
ORR	Log. OR Register	T1, T3	1.4 - 3.8 µs	1.2 - 2.6 µs
ORK	Log. OR Constant	T8, T2	1.3 - 2.5 µs	0.9 - 1.5 µs
XR	Exclusive OR	T4 - T7	3.8 - 6.3 µs	2.2 - 3.9 µs
XRR	Ex. OR Register	T1, T3	1.4 - 3.8 µs	1.2 - 2.6 µs
XRK	Ex. OR Constant	T8, T2	1.3 - 2.5 µs	0.9 - 1.5 µs
TM	Test Mask	T1	1.4 µs	1.2 µs
TNM	Test Not Mask	T1	1.4 µs	1.2 µs
Cl	One's Complement	T4 - T7	3.8 - 6.3 µs	2.2 - 3.9 µs
ClR	One's Complement Register	T1, T3	1.4 - 3.8 µs	1.2 - 2.6 µs
Character Handling Instructions				
LC	Load Character	T4 - T7	3.8 - 5.0 µs	2.7 - 3.6 µs
LCR	Load Character Register	T3	2.8 µs	2.4 µs
LCK	Load Character Constant	T2	2.8 µs	2.4 µs
SC	Store Character	T4 - T7	3.8 - 5.0 µs	2.4 - 3.3 µs
SCR	Store Character Register	T3	2.8 µs	2.1 µs
CC	Compare Character	T4 - T7	3.8 - 5.0 µs	2.7 - 3.6 µs
CCR	Compare Char. Register	T3	2.8 µs	2.3 µs
CCK	Compare Char. Constant	T2	2.8 µs	2.3 µs
ECR	Exchange Char. Register	T1	1.4 µs	1.2 µs
Branch Instructions				
AB	Absolute Branch	T8, T2	1.3 - 2.1 µs	0.9 - 2.0 µs
ABR	Absolute Branch Register	T1, T3	1.6 - 2.6 µs	1.2 - 2.4 µs
ABI	Absolute Branch	T4 - T7	4.0 - 5.2 µs	1.1 - 3.2 µs

		Addressing types	Execution times in µs	
			1.2 µs memory	0.7 µs memory
RB	Relative Backward Branch	T8	1.3 µs	1.1 µs
RF	Relative Forward Branch	T8	1.3 µs	1.1 µs
CF	Call Function	T2	4.8 µs	4.0 µs
CFR	Call Function Register	T1, T3	4.2 - 4.9 µs	3.6 - 4.1 µs
CFI	Call Function	T4 - T7	5.5 - 6.6 µs	4.5 - 5.4 µs
RTN	Return	T3	3 - 4.4 µs	2.7 - 4.1 µs
EX	Execute	T4 - T7	depends on instruction in operand	
EXR	Execute Register	T1, T3		
EXK	Execute Constant	T2		
Shift Instructions				
SLA	Left Arithmetic Shift	T8	2.0 + nx0.3 µs	1.9 + nx0.3 µs
SRA	Right Arithmetic Shift	T8	1.9 + nx0.3 µs	1.7 + nx0.3 µs
SLL	Left Logical Shift	T8	1.9 + nx0.3 µs	1.7 + nx0.3 µs
SRL	Right Logical Shift	T8	1.8 + nx0.3 µs	1.6 + nx0.3 µs
SLC	Left Circular Shift	T8	1.9 + nx0.3 µs	1.7 + nx0.3 µs
SRC	Right Circular Shift	T8	1.8 + nx0.3 µs	1.6 + nx0.3 µs
SLN	Left Shift and Normalize	T8	4.2 + nx0.5 µs	4.0 + nx0.5 µs
SRN	Right Shift and Normalize	T8	4.1 + nx0.5 µs	3.9 + nx0.5 µs
DLA	Double Left Arith Shift	T8	3.1 + nx0.3 µs	3.0 + nx0.3 µs
DRA	Double Right Arith Shift	T8	3.1 + nx0.3 µs	3.0 + nx0.3 µs
DLL	Double Left Log. Shift	T8	2.4 + nx0.3 µs	2.2 + nx0.3 µs
DRL	Double Right Log. Shift	T8	2.4 + nx0.3 µs	2.2 + nx0.3 µs
DLC	Double Left Circular Shift	T8	2.4 + nx0.3 µs	2.2 + nx0.3 µs
DRC	Double Right Circ. Shift	T8	2.4 + nx0.3 µs	2.2 + nx0.3 µs
DLN	Double Left and Norm Shift	T8	4.5 + nx0.5 µs	4.3 + nx0.5 µs
DRN	Double Right and Norm Shift	T8	4.9 + nx0.5 µs	4.7 + nx0.5 µs
Control Instructions				
ENB	Enable	T8	3.5 µs	3.4 µs
HLT	Halt	T8	1.7 µs	1.6 µs
RIT	Reset Internal Interrupt	T8	1.7 µs	1.6 µs

		Addressing types	Execution times in μs	
			1.2 μs memory	0.7 μs memory
INH	Inhibit Interrupt	T8	1.7 μs	1.6 μs
LKM	Link To Monitor	T8	3.5 μs	3.4 μs
SMD	Set Mode	T8	1.7 μs	1.6 μs
Input/Output Instructions				
CIO	Control Input/Output	T8	4.4 μs	4.3 μs
INR	Input to Register	T8	5.3 μs	5.2 μs
OTR	Output from Register	T8	4.4 μs	4.3 μs
SST	Send Status	T8	5.3 μs	5.2 μs
TST	Test Status	T8	5.3 μs	5.2 μs
External Transfer Instructions				
WER	Write External Register	T8	4.6 μs	4.5 μs
RER	Read External Register	T8	5.1 μs	5.0 μs
TL	Segment Table Load (MMU)	T4 - T7	15.4 - 16.8 μs	12 - 13 μs
TLR	Segment Table Load Register (MMU)	T3	15.1 μ	11.7 μs
TS	Segment Table Store (MMU)	T4 - T7	15.4 - 16.8 μs	12 - 13 μs
TSR	Segment Table Store Register (MMU)	T3	15.1 μs	11.7 μs
FLD	Floating Point Load	T4-T7		4.4 - 5.3 µ
FLDR	Floating Point Load/Register	T3		4.1 µs
FST	Floating Point Store	T4-T7		3.7 - 4.6 µs
FSTR	Floating Point Store/Register	T3		3.4 µs
Move Table Instructions				
MVF	Move Table Forward (P857 standard)	T8	4.7 + nx2 μs	4.5 + nx1.8 μs
MVB	Move Table Backward (P857 standard)	T8	4.3 + nx2 μs	4.1 + nx1.8 μs
MVUS	Move Table from User to System (MMU)	T8	4.3 + nx2 μs	4.1 + nx1.8 μs
MVSU	Move Table from System to User (MMU)	T8	4.7 + nx2 μs	4.5 + nx1.8 μs

8 Data Flow and Control

The control of data flow within the system is governed by the action which is being carried out at the time. The main sources of control being the instruction set, the input/output processors, the interrupt system, and the bus control system. Data flow within the system is carried out via the general purpose bus. The input/output processors use conventional control circuitry whilst the control exercised by the instruction set, the interrupt system and the bus controller are via a microprogram held permanently within the control ROM of the CPU.

The following examples of data flow cover only the instruction set. The data flow and control of the input/output processors is covered later in chapter 10. As all the instructions are controlled in a generally similar manner only one instruction, an add instruction, is shown.

Figures 8.1 and 8.2 show a flowchart of the microprogram actions carried out during an add instruction which places the result in a register. The required microprogram instruction words would be accessed in sequence from the address generated by the ROM address generator. Three separate actions take place to carry out the complete operation:

1. The instruction is accessed from memory using the address in S REG and P REG are then incremented by 2 in preparation for the next action.
2. The method of forming the operand is decided. The operand is accessed and placed into REG M and Q.
3. The arithmetic action is carried out and the result placed into the specified register. The Condition Register is updated.
   At the same time the next instruction is fetched and the registers P and S are incremented by 2.

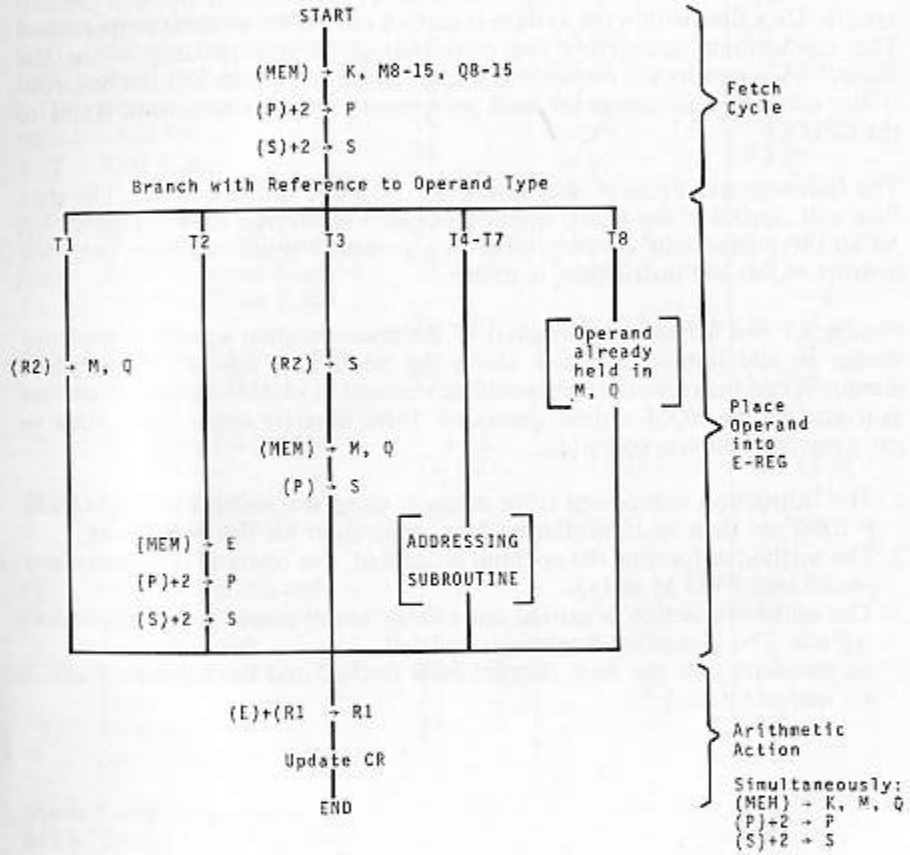


Figure 8.1 Instruction Microprogram

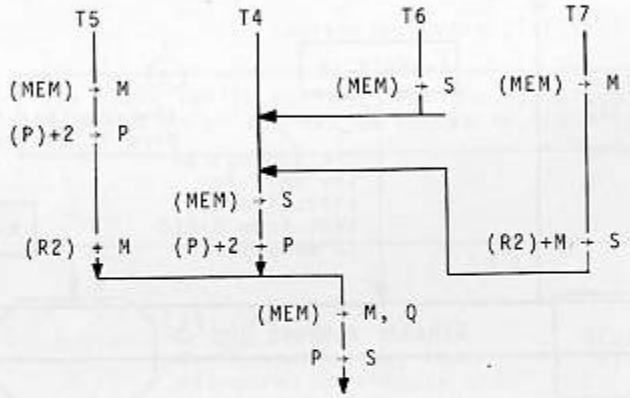


Figure 8.2 Microprogram Addressing Routine

Figures 8.3 to 8.6 show dia matically the data flow involved in the basic arithmetic operations, with r ect to the overall system block diagram on page 2-3.

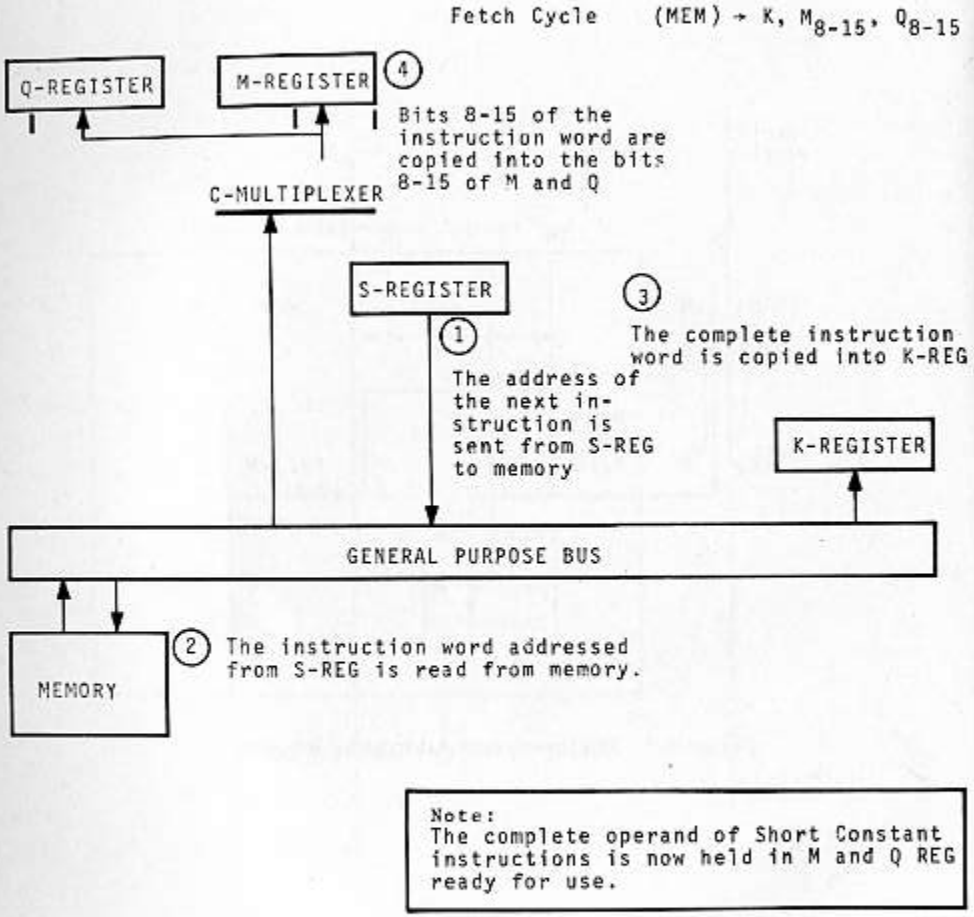


Figure 8.3 Accessing an Instruction

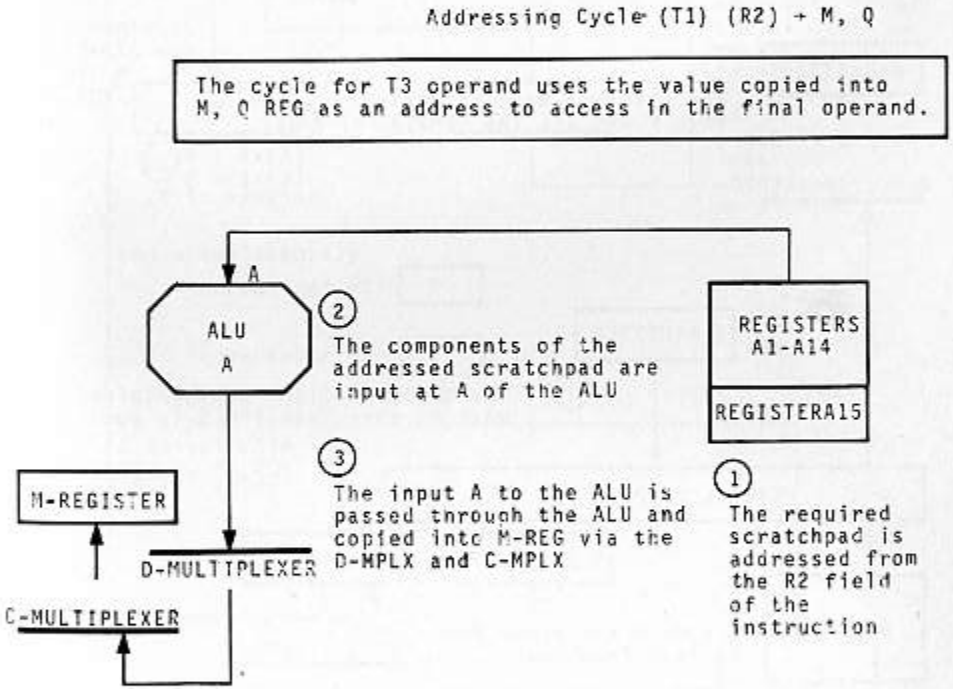


Figure 8.4 Addressing cycle (T1)

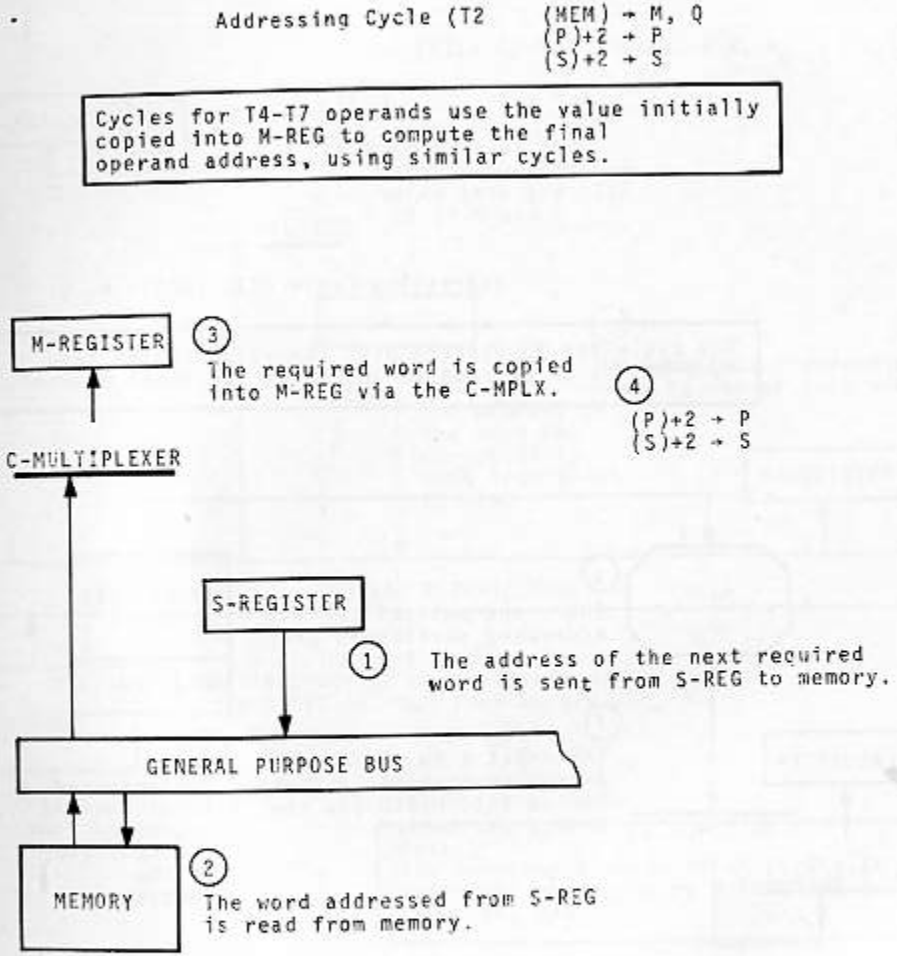


Figure 8.5 Addressing cycle (T2)

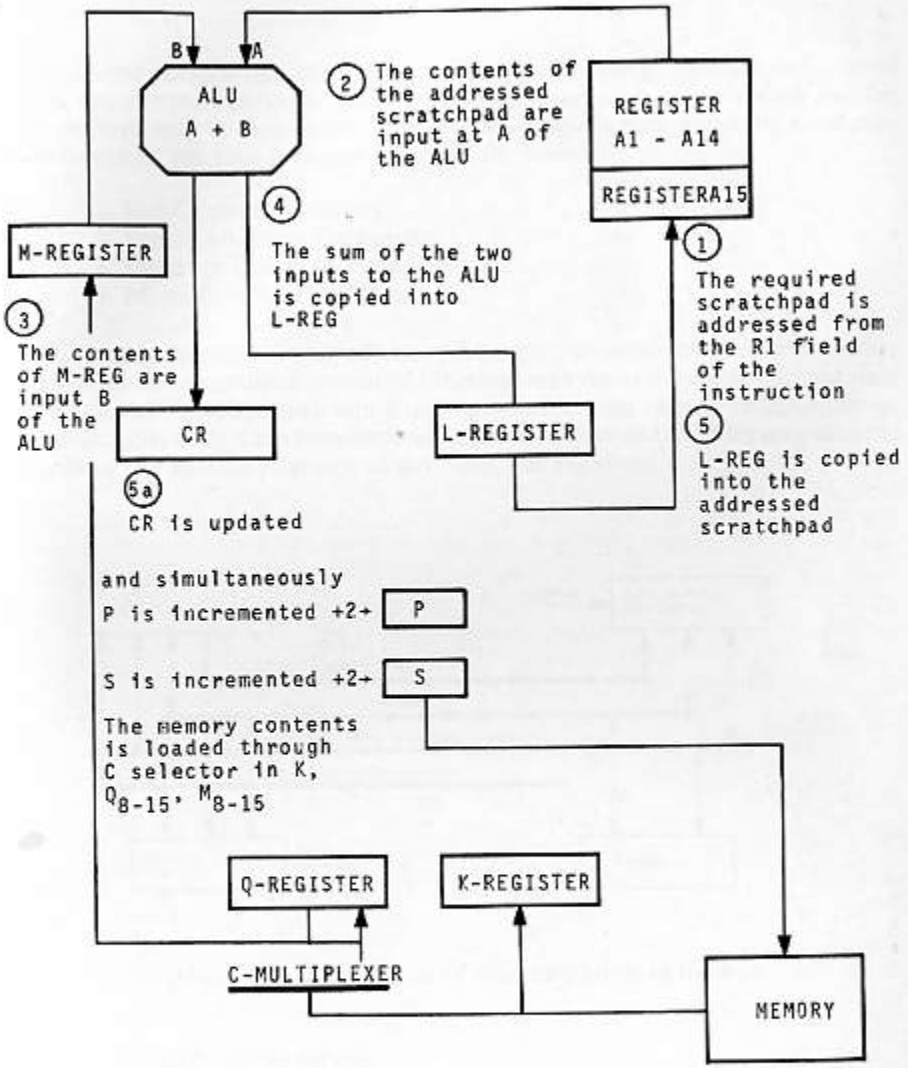


Figure 8.6 Execute cycle

The general purpose bus, consisting of 57 data, addressing, interrupt and control lines, handles the exchanges made between the main units of the system and for this purpose may be divided into four groups of signals each providing a separate bus function. The four function groups of the bus are:

1. Bus Control Functions
2. Data/Command Exchanges
3. Interrupt Handling Functions
4. Miscellaneous Functions

In order to gain the maximum efficiency from the bus certain of the bus functions may occur concurrently. Bus control functions may occur during the current data or command exchange, interrupt handling is carried out entirely independant of other facilities once it has been initiated, and miscellaneous functions may occur at any time and without reference to any other bus function.

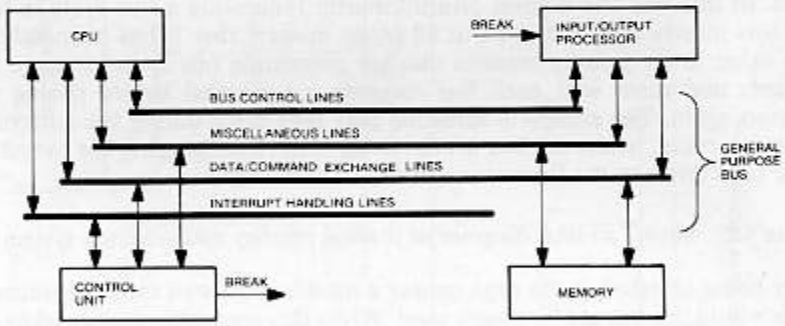


Figure 9.1 Connection of Standard Units to the Bus

BUS CONTROL FUNCTIONS

Efficient use of the bus for data or command exchanges is organized by a bus controller within the CPU. This controller allocates bus cycles for data or command exchanges one at a time, on a priority basis, to units which are able to request such cycles.

All units connected to the bus are known as master or slave units, and masters may act as either a master or slave depending on the type of exchange to be carried out. Units known as masters are those units which are capable of requesting a bus cycle for data or command exchanges and then, when the cycle is granted, controlling an exchange with either another master or slave, or controlling an exchange between two separate slaves. Units known as slave units are not able to request such bus cycles. To overcome clash conditions which may occur when two or more masters request a cycle simultaneously, normal bus allocations are made on a hardware wired priority basis to the masters, selection of the next master being carried out during the current exchange. Apart from this priority system the bus is allocated directly to the CPU in the event of a power failure being detected.

Priority Chain

The priority chain is hardware wired at installation time and within a standard system the CPU is given the lowest priority in the chain, other masters, such as the input/output processors will be given priority according to the system's requirements. Once a bus request has been made the bus controller initiates a master selection cycle to determine the highest priority master requesting the bus for a data or command exchange, as any number of masters may make a request at the same time. In response to a request a scan signal is initiated and is routed to all masters, via the masters, and in strict order of priority. The signal is only retransmitted from a master if the master is not requesting a bus cycle. In this way the highest priority master requesting a bus cycle is found and this master then indicates to all other masters that it has been selected. Any other lower priority masters that are requesting bus cycles remove their requests and must wait until bus requests are allowed before raising their requests again. The complete selection may take place during the current bus exchange cycle, whilst the bus is effectively busy, thus keeping the overall bus cycle time down to the time required for an exchange.

Figure 9.2 Shows a block diagram of the bus priority and selection system.

After being selected as the next master a master must wait until the exchange paths within the bus are no longer used. When this occurs the master takes control of the bus exchange paths, and removes its master selected signal from the bus. This final action allows a new master selection cycle to be carried out whilst the exchange cycle takes place.

DATA OR COMMAND EXCHANGES

These exchanges comprise data transfers with memory, command and response transfers to control units, and transfers with external registers.

As has been previously mentioned, before any such data or command exchange

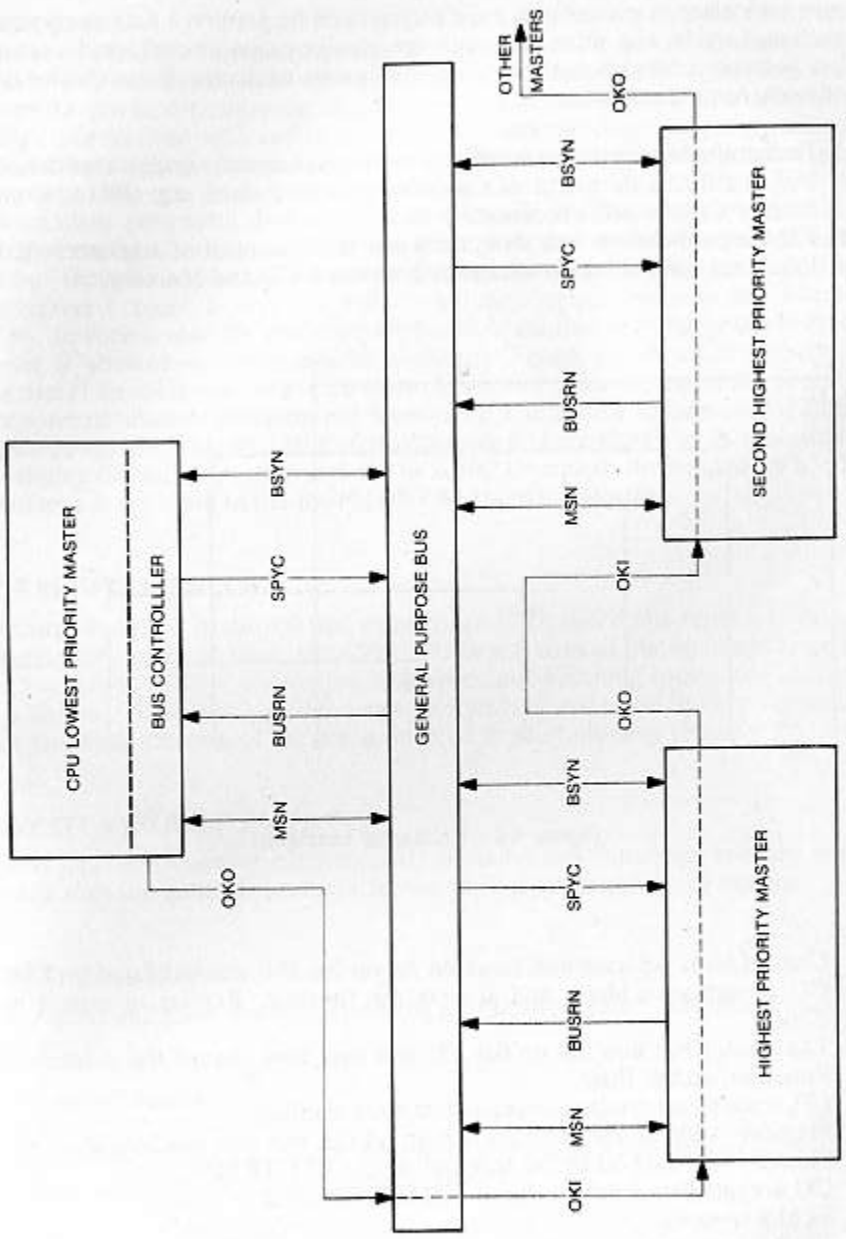


Figure 9.2 Bus Priority and Selection System

may take place, a master unit must request and be granted a data or command exchange cycle, and must wait until the previous data or command exchange has been completed before commencing its own exchange. Basically two types of exchange are possible:

1. Exchanges between the controlling master and another unit, where the other unit is either a slave unit or a master acting as a slave. e.g. CPU to Control Unit or CPU to I/O Processor.
2. Exchanges between two slave units under the control of a master, e.g. I/O Processor controlling an exchange between a CU and Memory.

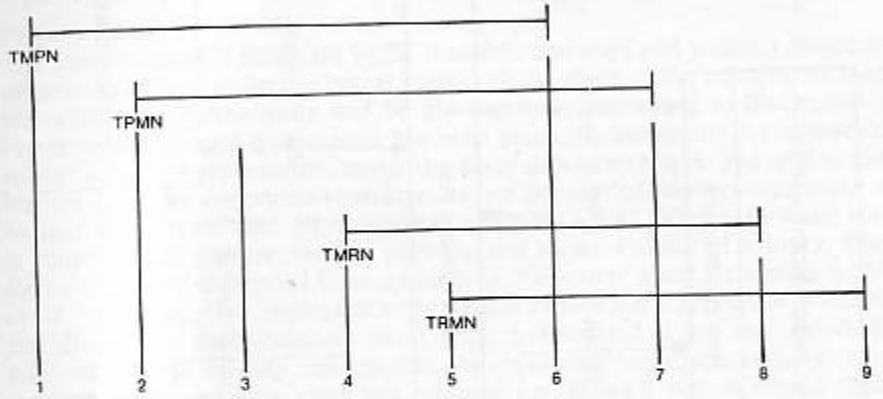


Figure 9.3 Exchange example

1 Control Unit Address and function set on the Bus and validated by TMPN
2 CU recognises address and accepts the function. Replies to master with TPMN
3 The master has now set up the CU and may now change the Address and Function, on the Bus.
CU remains set ready to receive data from the Bus
4 Memory Address and function set up on the Bus and validated by TMRN
5 Memory sets data on to the Bus and replies with TRMN
6 CU accepts data from the Bus TMPN removed
7 TPMN removed
8 TMRN removed
9 TRMN removed

Timing Control

The overall timing of exchanges made between units connected to the bus will differ widely and in principle will depend upon the type of device, and to some extent the physical positioning of the units on the bus. Control during an exchange is exercised with reference to timing and response signals raised by the units making the exchange and where necessary timing differences are accepted by the bus. This gives the system the ability to use either standard or non-standard peripheral devices without the need for special timing circuits, provided that the devices meet the overall requirements of the bus. In addition the bus includes a time out facility to unblock the priority system should for any reason a device or unit not reply to a timing signal from a master within 6.4 μs. In such a case the proposed exchange is aborted and the next selected master is allowed to commence its exchange. Figure 9.3 shows a simplified diagram of an exchange between a control unit and memory, under the control of a separate master (input/output processor). The timing signals used during the exchange are described later in this chapter and are shown as an indication of relative timing only, they are not to scale. Complete timing details of all transfers are available in the P856M/P857M Interface Manual.

INTERRUPT HANDLING

Interrupt handling is carried out independantly of other bus functions using separate bus interrupt lines, the CPU initiating a scan of the interrupt lines at the beginning of every instruction if the previous scanning took place at least 2 μs earlier. The operation of the overall interrupt system including the interrupt handling function of the bus is covered in the following chapter.

MISCELLANEOUS FUNCTIONS

These functions operate independantly of other bus functions and are concerned with the general reset and power on sequence within the system.

BUS SIGNAL LINES

The signals and lines associated with the four bus functions, N stands for active low, are:

Bus Control Signals

BUSRN *Bus Request*
The signal is raised by a master whenever it requires a bus data or command exchange cycle and bus requests are allowed.

SPYC *Scan Priority Chain*
This signal is raised by the bus controller in reply to BUSRN and indicates to all masters the commencement of a master selection cycle.

OKO/OKI — *Check Requests*
This signal is generated as OKO (OUTPUT) by the bus controller and received by the highest priority master as OKI (INPUT). It is chained through all the masters in order of priority as OKO/OKI. Onward transmission of the signal is inhibited by the first master which receives the signal and is requesting a bus cycle, this master is then selected as the next master.

MSN — *Master Selected*
This signal is raised by a master which has been selected as the next master to indicate the selection to all other masters. It is removed once the master concerned commences its exchange cycle.

BSYN — *Bus Busy*
This signal is raised by the master which has been selected and is now carrying out an exchange cycle. It is removed on completion of the cycle to allow the next selected master to commence its exchange.

Data or Command Exchange Signals

Timing Signals

TMRN — *Timing Master to Memory*
This signal is raised by a master and is used to validate the data and address, and to control the timing of an exchange with memory.

TMPN — *Timing Master to Peripheral*
This signal is raised by a master and is used to validate the control data and address, and to control the timing of an exchange with a peripheral control unit.

TMEN — *Timing Master to External Register*
This signal is raised by a master and is used to validate the data and address, and to control the timing of an exchange between a master and a unit containing an external register.

TRMN — *Timing Register/Memory to Master*
This signal is raised by memory or a unit controlling a register. It is used together with signals TMEN and TMRN in the controlling of an exchange cycle with a register or memory.

TPMN — *Timing Peripheral to Master*
This signal is raised by a peripheral device control unit and is used together with signal TMPN in the controlling of an exchange cycle with a peripheral device's control unit.

Bus Address Lines

MAD128, MAD64, MAD00 to MAD15 — *18 Address Lines*
These lines carry the memory address, register address, or peripheral address and requested function, during any exchange and are qualified by the timing signals from a master:

1. Memory Exchanges (Qualified by TMRN)
   MAD00 to MAD14 – These lines carry the 15-bit memory address required to access up to 32k of memory.
   MAD15 – This line is only significant in character operations and is used to define the character within the addressed word which is to be used.
   MAD64, MAD128 – These lines enable the extension of memory addressing to 64k and 128k words.

2. External Register Exchanges (Qualified by TMEN)
   MAD08 to MAD15 – These lines carry the 8-bit register address required to access up to 256 registers.
   MAD04 – This line is used to indicate a read or write operation to the addressed register.

3. Peripheral Control Unit Exchanges (Qualified by TMPN)
   MAD10 to MAD15 – These lines carry the 6-bit device address required to access up to 64 control units.
   MAD04 – This line is a function line used to indicate the direction of the exchange.
   MAD08 – This line is a function line used to indicate whether the exchange is a data exchange or a command or status exchange.
   MAD09 – This line is a function line reserved for use by special functions.
   MAD03 – This line is used to indicate whether the current word or character exchange is the last when exchanges are organized in blocks.

Bus Data Lines

BIO 00N to BIO 15N — *Input/Output Lines*
These lines are the 16 input/output lines used to carry data between the units making an exchange.

ACN — *Accept*
This signal is sent from a control unit to indicate that it accepts the request to carry out a designated function.

WRITE — *Write*
This signal is raised by a master controlling an exchange with memory to indicate that the exchange is a write to memory. When the signal is not present a read from memory cycle is indicated.

CHA *Character*
This signal is sent from a master to memory to indicate that the requested exchange is to be carried out in character mode.

Bus Interrupt Lines

SCEIN *Scan External Interrupts*
This line is used to allow units connected to the interrupt system via the bus to raise the bus interrupt lines as required.

BIEC *Bus Interrupt Encode*
These lines are the 6 lines which carry the encoded value, 0 to 62, of the highest priority outstanding interrupt request to the interrupt system.

Miscellaneous Signals

CLEARN *Clear*
This signal is sent from the CPU to all units connected to the bus and initiates a general reset of all such units.

RSLN *Reset Line*
This signal is raised during the power on or power restoration sequence and is used within the system to ensure an orderly commencement or resumption of operation without loss of data.

PWFN *Power Failure*
This signal is raised during the power off or power failure sequence and is used within the system to ensure an orderly run down of operation without loss of data.

The interrupt system within the CPU enables both internal and external interrupts to indicate that certain action is required with reference to the interrupt. This indication is given by raising an interrupt signal. Efficient handling of these interrupt signals is carried out by the hardware in conjunction with the system's software, ensuring that interrupts are serviced in the correct order of priority and with complete recovery facilities to the original program once the interrupt has been serviced.

ORGANIZATION

Within any system 63 individual interrupt signals are possible to control the priority running of 64 levels of program. Interrupt priority levels are numbered and encoded from 0 to 62, level 0 being given the highest priority. Signals at interrupt levels 0 to 7 are directly connected to the interrupt system at the CPU and are not encoded in binary form on the interrupt lines of the general purpose bus. Certain of the lines 0-7 are used by internal interrupts from the system and such lines are therefore reserved, the remaining lines in the group 0-7 may be used by facilities which are fitted within the basic mounting box. Signals at interrupt levels 8 to 62 are always encoded as a 6-bit binary value corresponding to their level and connected to the interrupt system via the interrupt lines of the general purpose bus.

OPERATION OF THE PRIORITY SYSTEM

Control units have priority levels set by hardware wiring within themselves and raise the required 6-bit value directly when an interrupt is raised. Two separate types of interrupt action may take place, one handling the eight possible interrupt signals which are directly wired within the CPU and the other handling all interrupt signals received via the general purpose bus.

The 8 basic interrupt signals are connected to their own priority encoder, the 3 output lines of which are connected via multiplexer to the system comparator. The multiplexer also accepts the encoded signals from the interrupt lines of the general purpose bus but is wired to give priority to the basic interrupt signals. The system's comparator compares the value presented by the multiplexer with the value already held within the PL register. Only of the value from the multiplexer is lower, that is higher in priority, than the value in the PL register

is further interrupt action taken. Once an interrupt of higher priority than the running program is detected a check is made to see if interrupts are allowed, the instructions ENB and INH being used to control such enabling.

Note: The connection of the magnetic tape control unit to the system requires an additional bus and translator board. The interrupt signal of this control unit is wired in a slightly different way. Refer to the Interface Manual for more details.

INTERRUPT ACTION

Interrupt action is carried out in two distinct parts:

1. The initial hardware action.
2. The programmed software action.

Together these actions must ensure that the correct level of program is entered, and that sufficient information with respect to the interrupted level is kept safely so as to enable this level to be restarted correctly once the interrupt has been dealt with.

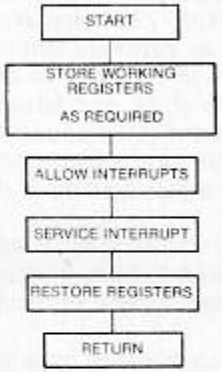
Associated with the hardware action is a fixed and reserved word in memory for each of the levels, locations /0 - /7C being used for levels 0 - 62 respectively. These locations, referred to as hardware interrupt locations, are addressed from a decode of the priority levels given to the interrupt lines and should always contain the start address of the associated level's coding. Start addresses for all the levels to be used in any program must be decided upon by the programmer and then set into the correct hardware interrupt locations by either the loading process, or by the initial running of the program.

The following flowcharts and explanations cover the hardware and software actions of the interrupt system and assume that interrupts are allowed and that the stack is empty at the commencement of the actions.

1. Hardware interrupt action commences by inhibiting further interrupts thus allowing the present to be serviced without interference if this is necessary.
2. The contents of the P and PSW registers are stored in a memory stack addressed from register A15. Hardware updating of register A15 is carried out each time a word is stacked.
3. On completion of the stacking operation the priority level of the interrupt is set into the PSW register, overwriting the original contents.

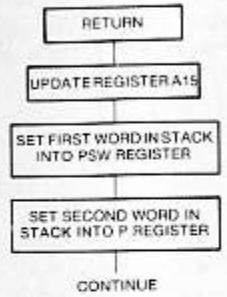
4. A branch is now made to the start of the new level's coding. This is carried out by using the priority level of the interrupt to address the required hardware interrupt location, the value in the addressed location is then set into the P register and is the address at which the program restarts after hardware interrupt action.

At this point the P and PSW registers contain information which is relative to the new level of program, the address at which the interrupted level is to restart and the PSW for this level. Further interrupts are inhibited. Instructions are now carried out from the new program level and it is these instructions which service the interrupt and define the software interrupt action to be taken, (usually an INR, OTR, SST, CIO Halt or an RIT instruction).



1. It may be required that the content of working registers being used by the interrupted level need to be stored. If this is required then the system stack may be used for this purpose and the contents of the registers will be stored consecutively with the contents of the P and PSW registers of the interrupted level. Alternatively a separate safe area may be designated and used for the preservation of the working register's contents.
2. Interrupts from higher levels may of course occur at any time and it may be required that these are serviced. If this is the case then interrupts may be allowed before any action to store the working registers, and in this case the higher interrupting level should store the contents of the working registers it intends to use. The order in which storing of registers and the enabling of interrupts is carried out is a matter for the programmer to decide with reference to the specific requirements of the overall program.
3. Whichever action or device caused the interrupt may now be serviced and any necessary program flags or switches must be set before completing the routine.
4. Before a return is made to the originally interrupted level any working registers which were saved must be restored correctly either from the system stack or from the specific safe area used.

At this point the required action of the interrupt routine is complete, all necessary flags and switches are set, and the working registers contain the values required by the originally interrupted level. The PSW and P register values required by the original level are addressed by register A15. Return action may now be requested using the Return instruction and specifying register A15 as the stack pointer within this instruction.



1. The contents of register A15 are updated to address the first word of the stack as it stands. The PSW and P registers still contain the original level.
2. The first word of the stack is set into the PSW register, restoring the original level's PSW.
3. The second word of the stack is set into the P register, thus specifying a branch to continue the original level when program action is resumed.

Before program action is resumed any outstanding interrupts are checked for priority against the value in the PL part of the PSW register. If a higher level interrupt exists then this is serviced in the manner just explained. If no higher level interrupt exists then the original level continues.

Figure 10.1 shows diagramatically a possible interrupt sequence.

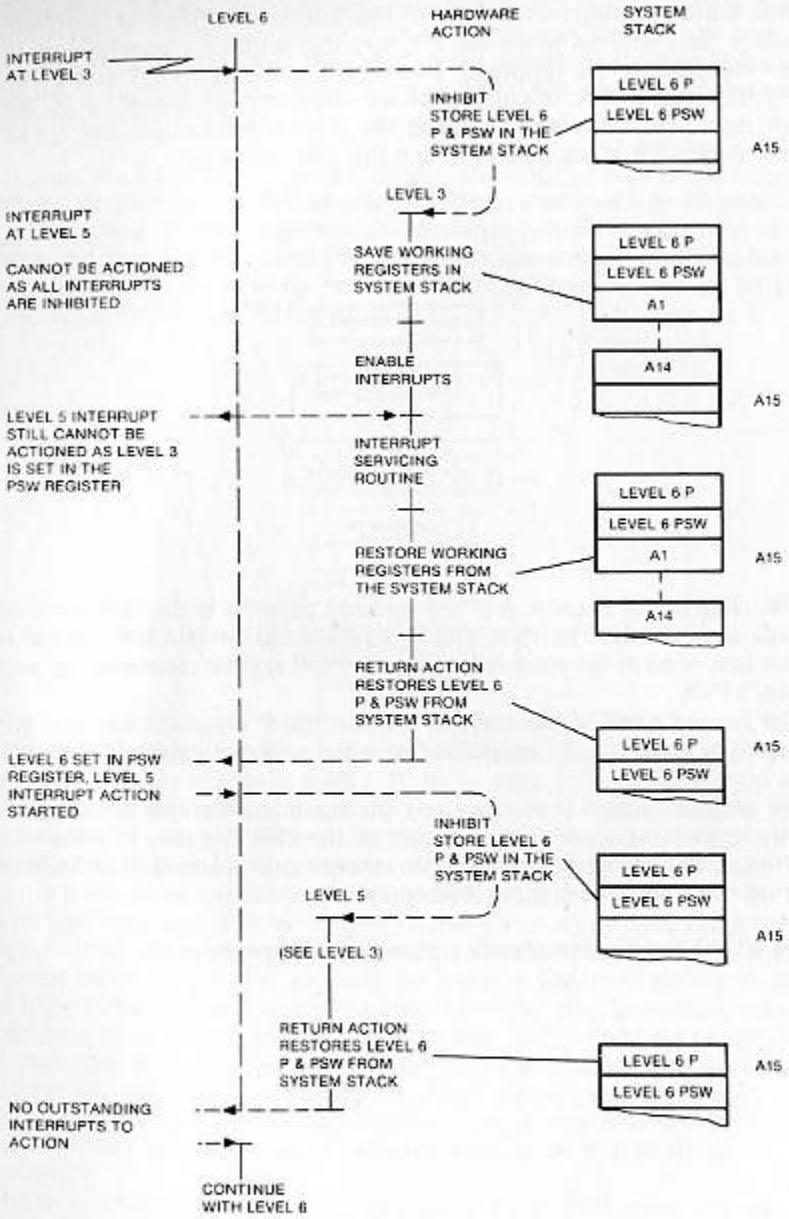


Figure 10.1 Diagram of interrupt sequence

STACKING

The use of register A15 as a stack pointer gives the user the facility of automatic updating of this register. Updating is carried out each time A15 is used for addressing purposes by decrementing its contents by 2, thus the stack is filled from the higher addressed locations to the lower addressed locations. The programmer is responsible for determining the size of the stack required and for setting the start address of the stack into register A15. A further facility available to stack handling is the generation of an interrupt when the stack pointer address is <128, this enables action to be taken to ensure that the stack does not overwrite any of the reserved area at the beginning of memory.

11 Input/Output

Data transfers within the system may take place between any unit which is destinated a master and another master or slave unit, or between two slave units under the control of a master. All data transfers take place via the general purpose bus and within the system take the form of parallel 16-bit word, or 8-bit character transfers. Figure 11.1 shows a diagramatic layout of the units concerned with data transfers and the channels with which each is associated. The exchange paths available within the system are:

- CPU/Control Unit
- CPU/External Register
- Memory Slave
- Memory Master

The basic transfer channel is the programmed channel which uses only the CPU/Control Unit path to transfer data, one character or word at a time. between a CPU register and a control unit. Optionally available within the standard system are the input/output processor channels. These channels may be used by devices connected directly to the general purpose bus and may use the same control units which are used for connection to the programmed channel. Up to 64 input/output processor subchannels may be connected and used simultaneously via a priority system of servicing exchange requests. Priority allocations being made initially to an input/output processor as a master, and secondly to one of the subchannels available within each processor. Transfers carried out via the input/output processor channels use three of the exchange paths:

- CPU/External Register to set up the transfer parameters
- CPU/Control Unit to start or stop the transfer and check the status of a device as necessary.
- Memory/Slave to allow the direct transfer of data between the memory and the control unit as initiated by the input/output processor.

In addition to these channels, two non standard modes of operation are possible, each using one of the available exchange paths. Direct access to memory is possible using the Memory/Master path and transfers between internal and external registers are possible using the CPU/External Register path.

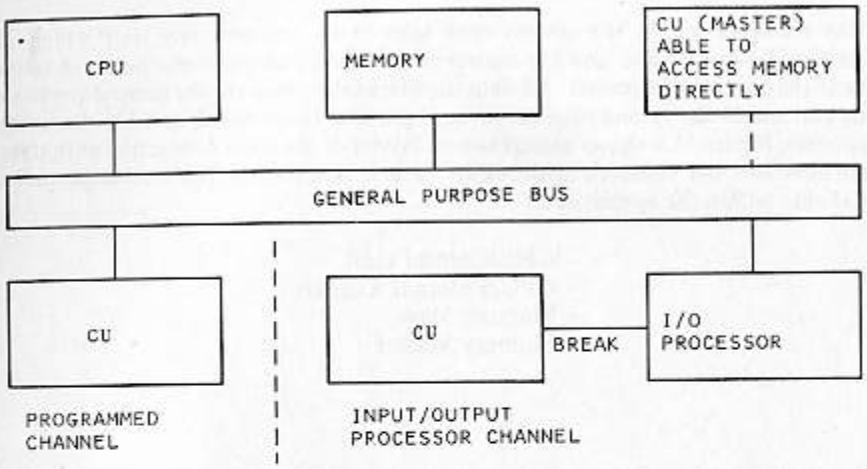


Figure 11.1 Units concerned with transfers

CONTROL UNITS

Details of the standard control units available within the system are given in chapter 16. A control unit is required to connect any external device to the system. The function of the control unit is to translate the address, control, and timing signals of the general purpose bus into the necessary signals to exercise discrete control of the device. The basic requirements of any control unit are:

1. Address Decoder
2. Function Decoder
3. Sequence Control - to enable the device to transmit and receive data and to control the initial starting and stopping of the device.

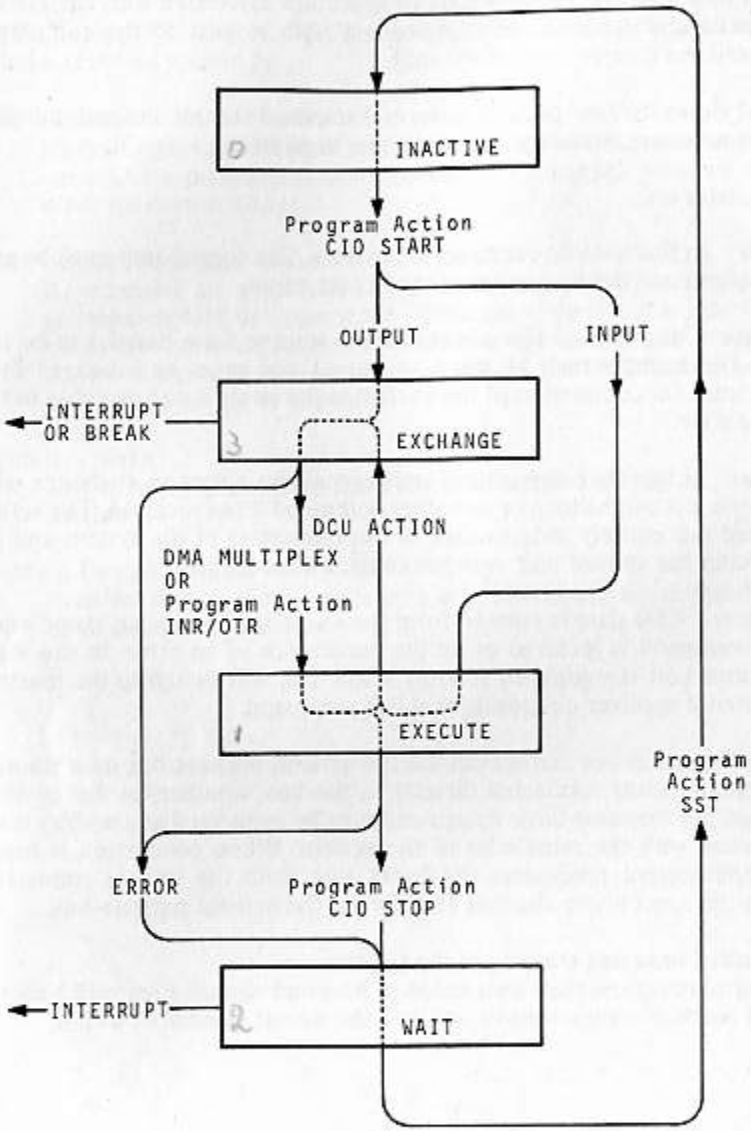


Figure 11.2 Four states of standard control unit

The standard control units used within this system are required to carry out four separate actions and have a state of operation associated with each of the actions, particular functions are carried out with respect to the commands received and the current state of the unit.

Figure 11.2 shows the four possible states of a standard control unit and indicates the actions necessary to change from one state to another.

The four states are:

1. *Inactive* In this state no exchange is possible. The control unit must be sent a start command before any transfer can take place.

2. *Exchange* In this state the control unit is waiting for a transfer to be initiated. The transfer may be input or output and must be intialized by a master unit. On completion of the exchange the control unit switches to the execute state.

3. *Execute* In this state the control unit carries out either an exchange with the device it is controlling, or any other command it has received. The action is carried out entirely independant of the remainder of the system and on completion the control unit switches to the exchange state.

4. *Wait State* This state is entered from the exchange or execute states when a stop command is received or on the occurrence of an error. In this state the control unit is waiting to send its status and will switch to the inactive state when it receives a request for status command.

Because all transfers are carried out via the general purpose bus on a priority basis all control units connected directly to the bus, whether for fast or slow devices, can use the same basic design and may be connected at a priority level in accordance with the remainder of the system. When connection is made via the input/output processors the break line from the unit is connected directly to the appropriate channel and not via the general purpose bus.

Control Units Connected Directly to the GP Bus
These control unit form their own encoded interrupt signals. Figure 10.3 shows an overall block of such a control unit and the signals associated with it.

DEFINITION OF UNITS

The definition of units as masters or slaves and the control exercised by masters within a standard system is:

CPU - Master
Normally given the lowest priority access to the bus, apart from specific cases. As a master it is able to control exchanges between itself and other masters or slaves.

Input/Output Processors - Master
As a master an input/output processor is normally given a priority in accordance with its importance within the system and is able to control exchanges between the memory and a device control unit, both of which will be designated slaves. Acting as a slave the input/output processors are initially set up by the CPU.

Memory - Slave
The memory is always a slave and is controlled during an exchange by a master.

Standard Device Control Units - Slave
Standard device control units are always slaves, and are controlled during an exchange by a master.

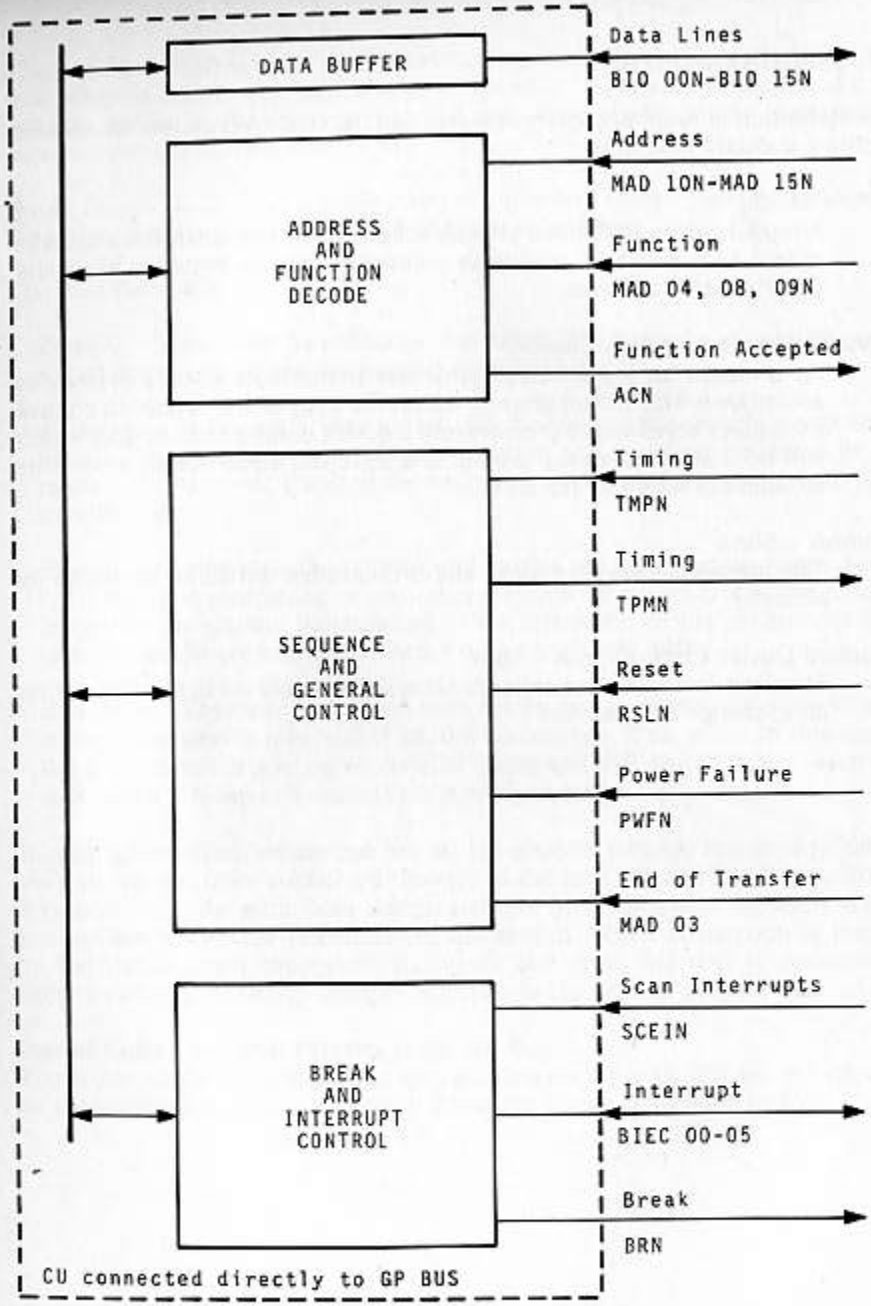


Figure 11.3 Signal Exchange

PROGRAMMED CHANNEL

The programmed channel is the basic transfer channel and is standard with all systems. Apart from providing an exchange facility between a control unit and the CPU, it also provides the initialization path between the CPU and the input/output processors. In all actions concerning the programmed channel the CPU is the controlling master.

Data are transferred by the use of specific input/output and external register instructions within a program and each word or character exchange requires a separate instruction. Apart from the instructions which carry out the exchanges, instructions are available to start and stop a device and to check the status of a device. In practice program loops are used in the control of a block transfer.

Two possible modes of operation exist when using the programmed channel for data transfers:

Wait Mode
This is the simplest but slowest form of transfer and is in most cases never used. Each word or character is exchanged separately and the complete program is held up in a waiting loop between individual exchanges. In this mode the maximum transfer rate obtainable is dependant on the time taken to execute the necessary program loop, or the time taken by the device concerned to execute a single exchange, whichever is the slowest.

Interrupt Mode
By the use of this mode, operation of the programmed channel is carried out without the use of time consuming waiting loops. Each word or character is still exchanged separately, but the necessary instructions form a part of an interrupt routine. This means that the main part of any overall program can continue to be executed during the time taken to actually carry out an exchange.

When the device control unit is ready to exchange another character it raises an interrupt and the main program is stopped whilst the new exchange is initiated by the interrupt routine. On completion of the interrupt routine the main program is restarted and continues whilst the exchange is in progress. This sequence can be continued until either the necessary transfer is complete or until the main program requires to use the transferred data. In the second case the main program must be made to wait as necessary.

Commands and Responses
To enable operation of a control unit via the programmed channel the following instructions may be used as commands to control units.

CIO START
CIO STOP
INR
OTR
SST
TST

The responses given to these commands are set into the condition register:

CR = 0 Command Accepted
CR = 1 Command Rejected
CR = 3 Address Unknown

The use of the External Register instructions used to initialize the input/output processors will be covered when the input/output processors are explained.

Control and Data Flow
As only one method of forming the operand for input/output instructions exists, T8 Short Constant, the control sequence for all instructions is similar, the main difference being between the input, INR, SST, TST instructions and output, OTR, CIO instructions. Figures 11.4 and 11.5 show diagramatically the data flow involved during the action of the input/output instructions.

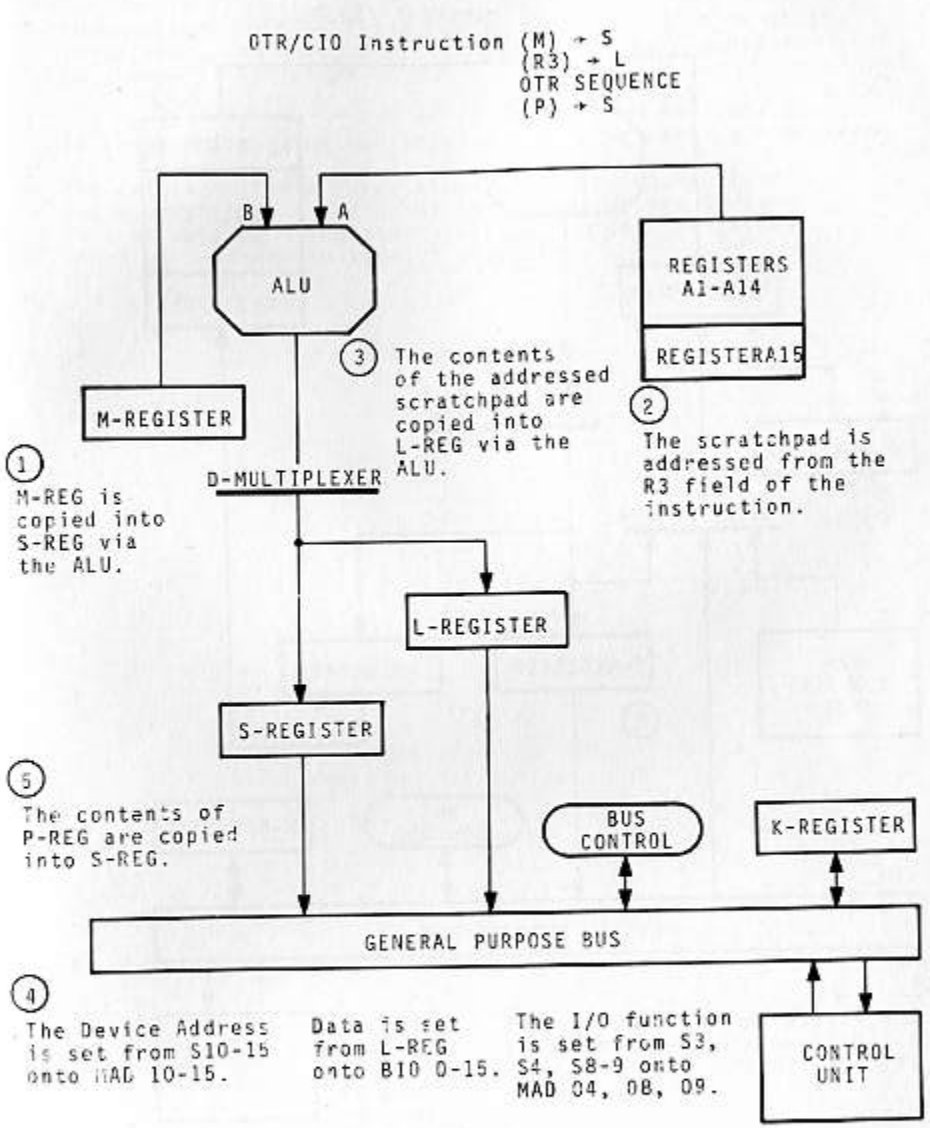


Figure 11.4 OTR/CIO Instructions Flow

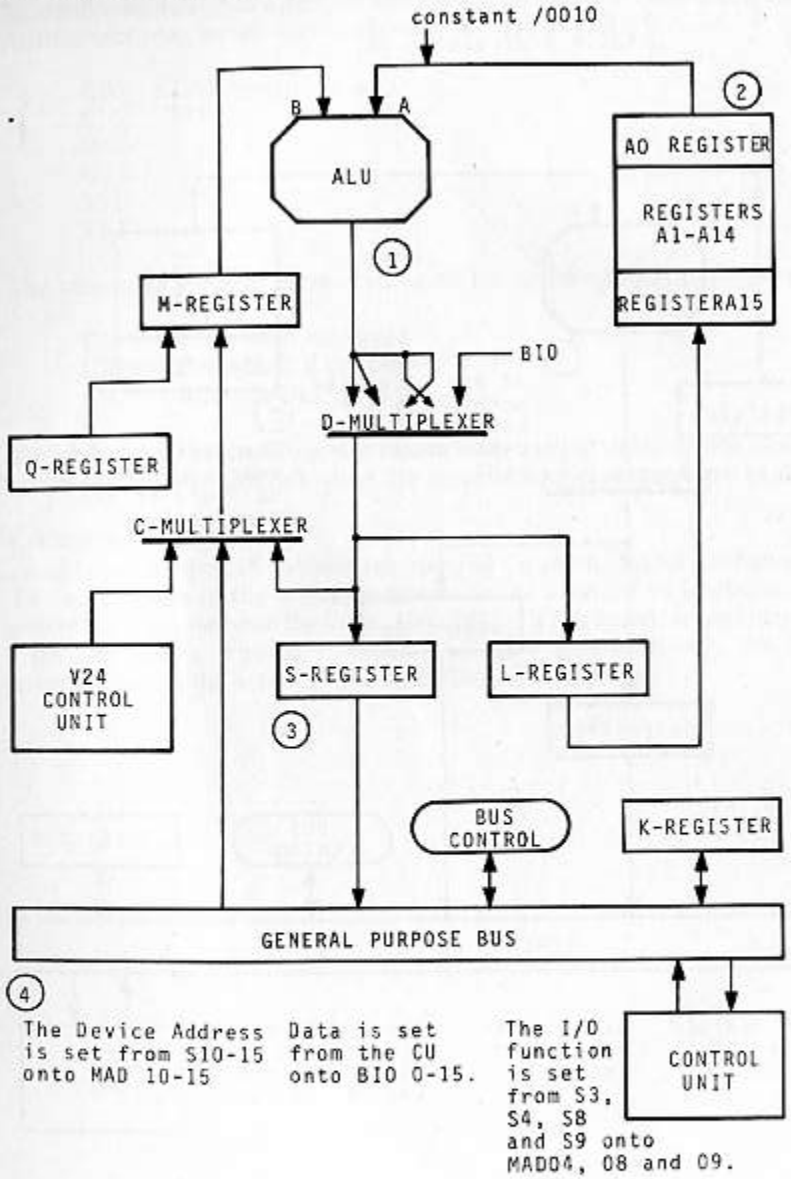


Figure 11.5 INR/SST/TST Instructions Flow

1 The constant /1000 is loaded in M (/10 on A input of ALU, D in exchange character mode, and C Multiplexer).

2 The constant /800 is loaded in AO through ALU, D in shift right mode, and L register. In the same time Q contents (0, K08-K15) is loaded in M.

3 The logical "or" of M and AO is loaded in S register. At the exchange time contents of S will be sent on MAD lines.

4 The exchange itself takes place - BIO are copied in R3 register through D and L and the output of the Serial Control Unit in M register - Then a logical "or" of R3 and M is returned to scratch pad R3.

INPUT/OUTPUT PROCESSOR CHANNELS

The input/output processor channels provide a fast method of block transfer between the system's memory and up to 64 different device control units. Transfer rates of up to 830k words/sec or 1.2Mw for the fast memory being possible. A maximum of 8 input/output processors is possible and each is capable of controlling up to 8 subchannels. These channels replace the normal instruction sequence of a programmed channel transfer with a hardware sequence to carry out each exchange, and a data path is provided between the appropriate device control unit and memory via the general purpose bus.

Both types of control unit available may be connected via the input/output processor channels and in all cases the control units may be addressed from either the appropriate input/output processor or from the CPU. The normal states and sequencing of the device control units apply but the interrupt raised in the exchange state for the programmed channel is replaced by a break signal wired directly to the input/output processor's priority system and used to initiate each hardware exchange sequence.

Organization

Any number of the devices connected to the input/output channels may be set up and started so that the overall transfers of each are carried out together. All break requests would be serviced according to a priority given to each device and derived in two separate ways:

1. Where more than one input/output processor is connected to the system each separate processor would have a priority according to its position in the chain of masters.
2. To enable each processor to differentiate between the 8 break signals from the possible 8 devices connected to it, the break lines are connected to the channels via a priority system.

Each time a break request is received by a channel, the channel requests a bus cycle. When the cycle is granted the channel carries out a single exchange between the device control unit having the highest priority break request outstanding, and the memory.

Associated with each of the possible 8 control units connected to any channel are two 16-bit registers which hold the parameters of any transfer to be carried out with a unit. Prior to any transfer the two registers must be correctly set up to hold the transfer parameters. During any transfer the registers are hardware addressed from the priority decoding of the break signals and are used and updated by the I/O processor.

Figure 11.6 shows the layout of the control registers, the contents of which are known as First Control Word and Second Control Word respectively.

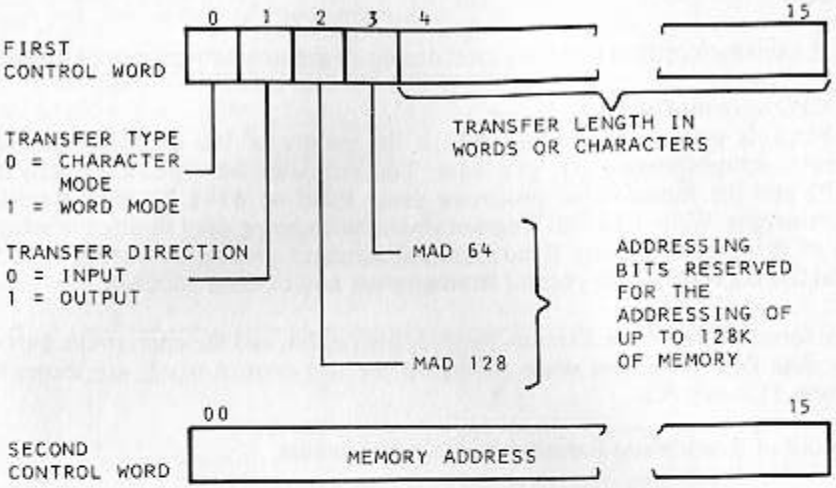


Figure 11.6 I/O Processor Control Words

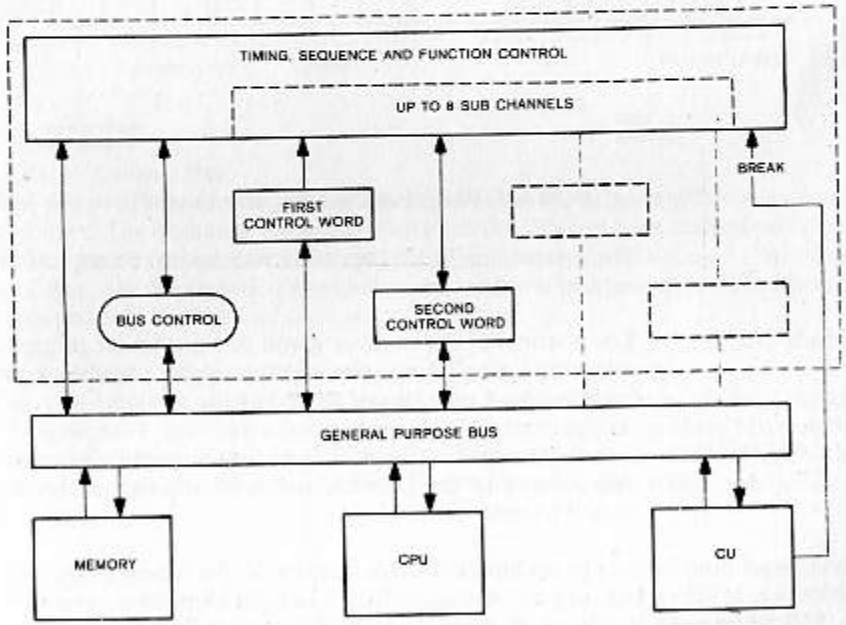


Figure 11.7 I/O Processor within the system

Control and Data Flow
Figure 11.7 shows a diagramatic layout of an input/output processor within the system.

Three separate control paths are used during an input/output processor transfer:

1. CPU to Input/Output Processor
When this path is in use the CPU is the master of the exchange and the input/output processor acts as a slave. The exchange takes place between the CPU and the input/output processor using Read or Write External Register Instructions. Write External Register instructions being used during the setting up of the processors and Read External Register instructions being used to read the contents of the control words during end or error routines.

The format of the Write External Register Instruction and the appropriate part of the data flow concerned when setting up the two control words are shows by figures 11.8 and 11.9.

Layout of Read/Write External Register Instruction.

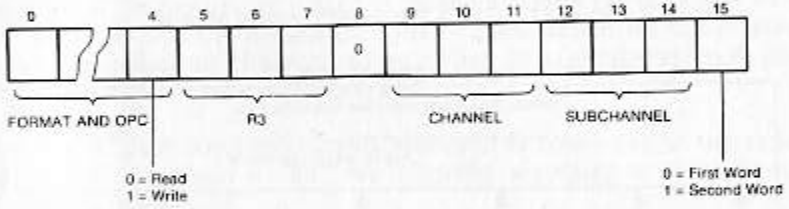


Figure 11.8 Read/Write External Register Layout.

R3 – The contents of R3 is the value which is to be set into the control word.

CHANNEL – The channel is the number given to a particular processor with respect to its priority relative to the possible 8 processors which may be connected to the system.

SUB CHANNEL – The subchannel is the priority given to a particular control unit relative to the possible 8 devices which may be connected to each channel.

The action of the instruction within the CPU is to copy bits 8-15 from K REG into S REG via M REG and copy the contents of R3 into L REG, before carrying out an OTR SEQUENCE. Note, bit 4 from the OPC is used to define the instructions as Read or Write.

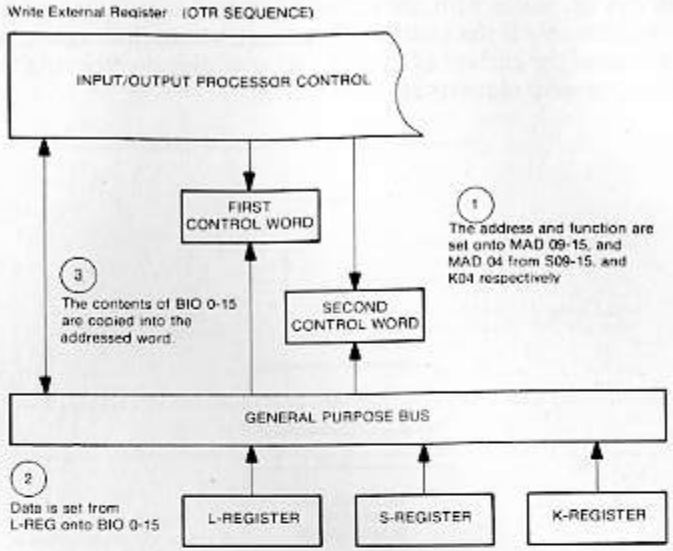


Figure 11.9 WER Instruction Flow

2. CPU to Control Unit
When this path is in use the CPU is the master of the exchange and the control unit is the slave. The exchange takes place between the CPU and the control unit, data flow being as for programmed channel transfers. (See figures 11.4 and 11.5). By the use of this path the devices connected to the input/output processor channels may be stopped and started, and their status requested.

3. Input/Output Processor to Memory and Control Unit
When this path is in use the input/output processor is the master of the exchange and the memory and control unit are slaves. The exchange takes place between the memory and the control unit. Figures 11.10 to 11.12 show an outline of the exchange action and the data flow during the two cycles which comprise an exchange.

Such an exchange would be initiated after the receipt of a break signal and after the input/output processor had requested and been granted a bus cycle. The break signal would be removed once the necessary actions to initiate an exchange had been carried out.

Note: A further exchange with respect to the overall transfer would be carried out immediately if the control unit raises its break line again, during the execution of the current exchange, and providing no other higher priority breaks or master requests are outstanding.

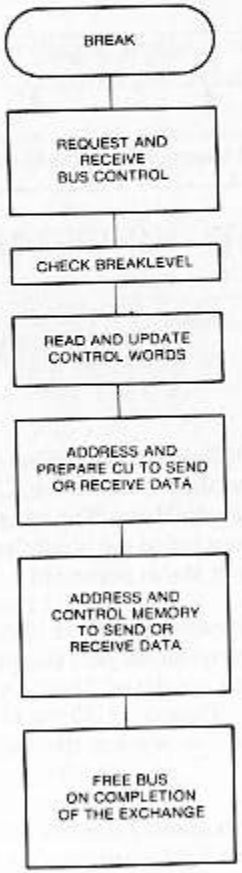


Figure 11.10 Exchange action

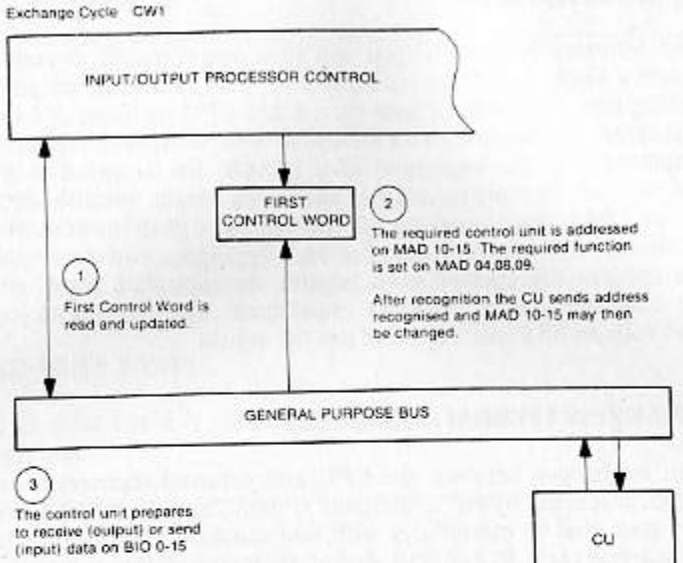


Figure 11.11 Exchange Cycle CW1

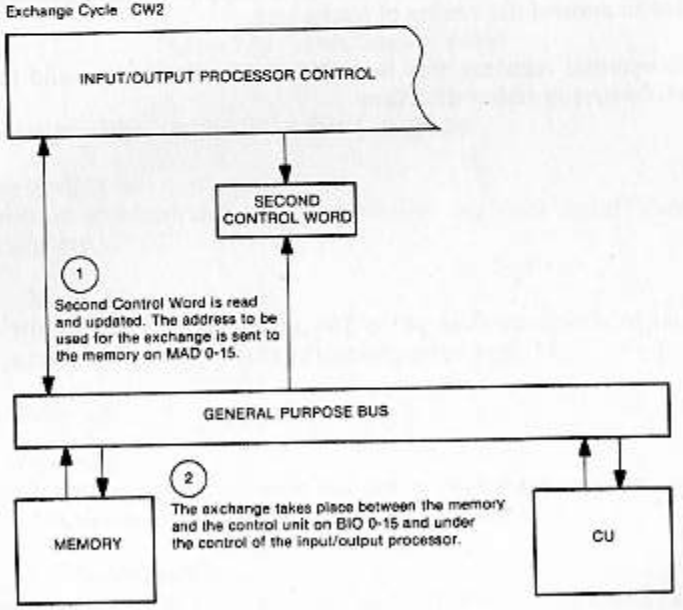


Figure 11.12 Exchange Cycle CW2

DIRECT MEMORY ACCESS

The Direct Memory Access channel manages data transfers directly between memory and a single high-speed control unit. This unit must be plugged into the mounting box. Transfers of data do not use CPU registers and there is no need for program control except for starting the exchange and testing the status after completion. At the beginning of a transfer the program uses a WER instruction to load the starting address and block length into the control word register. The DMA logic provides all Bus timing signals to control the data transfers directly between the memory and the high-speed control unit. The logic also updates the control word register for each data word and detects when the complete block has been transferred. The data block transfer is terminated with an SST instruction to get the status.

TRANSFER CPU/EXTERNAL REGISTERS

The use of exchanges between the CPU and external registers to set up an input/output processor within a standard system has already been explained. Exchanges may also be carried out with non-standard control units to enable transfers between the CPU and an external register within a control unit for use by specialized input/output systems. Such systems would operate in a similar manner to the programmed channel and may or may not use the interrupt system to control the timing of exchanges.

Up to 256 external registers may be addressed by the system and the facility enables exchanges in either direction.