IBM Field Engineering

Theory of Operation

(Manual of Instruction)

Restricted Distribution

Specifications contained herein are subject to change without notice. Revisions and/or supplements to this publication will be issued periodically.

System/360 Model 44

Principles of Operation–Processing Unit

IBM® Field Engineering

Theory of Operation

(Manual of Instruction)

Restricted Distribution

Specifications contained herein are subject to change without notice. Revisions and/or supplements to this publication will be issued periodically.

System/360 Model 44

Principles of Operation–Processing Unit

PREFACE

This volume contains the Principles of Operation for the IBM 2044 Processing Unit, the processor for the IBM System/360 Model 44. The manual describes the operation of the machine instructions (other than floating-point instructions) and program and machine interrupts, for both the basic machine and the basic machine with accelerator feature. Where the description for the basic machine differs from that for the accelerator feature, the relevant text is indented and appropriately headed 'BASIC only' or 'ACCELERATOR only'.

The manual assumes knowledge of the System/360 as described in IBM System/360 Principles of Operation, Form A22-6821.

This volume is one of the manuals that constitute the IBM Field Engineering Theory of Operation manual for the IBM System/360 Model 44. The other volumes and their form numbers are:

System/360 Model 44, Introduction and Functional Units, Form Y33-0001: Gives a general outline of digital computers and computing technique, defines the relationship of the IBM 2044 to the System/360 and describes the various parts which form the processing unit.

System/360 Model 44, Principles of Operation - Channels, Form Y33-0003: Describes the Common Channel area, the Multiplexor Channel 0 and the High Speed Multiplexor Channel.

These volumes are referenced in other volumes by the main element of their titles.

Reference is also made in these volumes to the following associated manuals:

Field Engineering Theory of Operation (FETO), IBM System/360 Model 44, Floating Point Feature, Form Y33-0005: Gives an introduction to floating-point arithmetic, describes the functional implementation of floating-point arithmetic in the 2044 and details the operation of floating-point instructions.

Field Engineering Theory of Operation (FETO), IBM System/360 Model 44, Single Disk Storage Drive, Form Y33-0006: Gives an introduction to the operation of the control unit and describes in detail the functional parts and the operations that may be performed.

Field Engineering Maintenance Manual (FEMM), IBM System/360 Model 44, Form Y33-0007: Contains information for servicing the 2044 Processing Unit.

Field Engineering Maintenance Diagrams (FEMD), IBM System/360 Model 44, Volume 2, Form Y33-0008: Contains maintenance information in the following categories: Data Flow Charts, Flow Charts, Timing Charts, MAP's.

Other related manuals that describe units used in the System/360 Model 44 are:

Field Engineering Manual of Instruction (FEMI), 1052 Adapter, Form 223-2808.

Field Engineering Maintenance Manual (FEMM) Single Disk Storage/Direct Access, Form Y26-3663.

First Edition

This manual makes obsolete Field Engineering Theory of Operation, System/360 Model 44, Forms Z33-0003-0, Z33-0004-0.

The manual is written basically to Engineering Change Level 390049 and in some cases anticipates Engineering Change Level 390063. Significant changes or additions to the information in the manual will be covered in subsequent revisions or FE supplements.

This publication was prepared by IBM European Laboratories, Product Publications. A form is provided at the back of this manual for reader's comments. If the form has been removed, comments may be addressed to: IBM Corporation, FE Manuals, Dept. P96, PO Box 390, Poughkeepsie, N. Y. 12602

ILLUSTRATIONS

The following abbreviations are used in this manual:

ac	alternating current
CLA	Carry Look-ahead
CLI	Compare Logical Immediate
dc	direct current
E-phase	Execute-phase
EXOR	Exclusive OR
GPR	General Purpose Register
HW funnel	Hardware funnel
IC	Instruction Counter
I-cycle	Instruction-cycle
I-fetch	Instruction-fetch
ILC	Instruction Length Code
I/O	Input/Output
IPL	Initial Program Loading
Op	Operation
PSW	Program Status Word
PZR	Possible Zero Remainder
SAR	Storage Address Register
SDR	Storage Data Register

INTRODUCTION TO INSTRUCTION FETCHING

- Instructions are either one or two halfwords in length.
- Instructions must be located on integral halfword boundaries.
- Instruction-fetch reads out the instruction to instruction registers and develops the effective address for all but RR type instructions.
- SS format instructions are not used in the IBM System/360 Model 44 instruction set.

The length of a machine instruction can be one or two halfwords. The length is determined by the number of storage addresses required to perform the operation. An instruction of one halfword cannot cause any reference to main storage because it is the second halfword of an instruction, where present, that contains the storage address in the form of a base address followed by a displacement. Therefore, an instruction two halfwords long is required to provide one storage address. All instructions must be located in storage on integral halfword, that is, any 24-bit address whose low-order bit is zero.

The instruction field is read out from storage by Instruction-fetch (I-fetch). The information brought out is processed in preparation for the execution of the instruction. This execution is carried out in the Execute-phase (E-phase).

The System/360 Model 44 instruction set does not contain any SS format instructions. Therefore, no Model 44 instruction exceeds two halfwords in length.

For RR format (one halfword long), the I-fetch reads out the instruction from storage and sets it into the instruction registers (Op, Ra, Rb and Rc). For RX, RS and SI formats, the I-fetch reads out the full-word instruction, sets the Op, Ra, Rb and Rc registers with bits 00 to 19 of the instruction, sets the 12-bit displacement to the B register and develops the effective address in the B register.

For all instruction formats the Op register content is decoded to set the controls for the E-phase of the instruction.

For two instructions the E-phase is overlapped with the I-fetch. These are the supervisor-call instruction and the branch-on-condition instruction where the branch is not taken. These instructions are described in the "Status Switching Instructions" and "Branching Instructions" sections respectively.

For all other instructions and for branch-on-condition where the branch is to be taken, the E-phase always follows I-fetch.

I-fetch cannot begin if, at the end of E-phase, an interrupt is pending or the interval timer requires updating. If both these conditions are present, the interrupt is taken and then the interval timer is updated prior to the next I-fetch. (Refer to the "Interrupts" section.)

Under normal operating conditions of running a program, the machine stops processing at the end of the E-phase (end-execute time) and after all pending interrupts and interval-timer updates are handled if any of the following conditions are present:

1. The wait bit, PSW 1 bit 14 is a one.
2. The console rate switch is in the INSN STEP position.
3. The stop button has been pressed.

Condition 1 causes the machine to enter the wait state and the interval timer is updated when required. The machine can proceed only if an I/O or external interrupt is requested.

Condition 2 is an operator intervention. The machine enters the manual-stop state and the I-fetch proceeds only when the start button is pressed. Any interrupt or timer update requested while the machine is in the manual-stop state remains pending until the next end-execute time.

Condition 3 causes the machine to enter the manual-stop state after all pending interrupts are taken. Any I/O activity proceeds to termination with the associated interrupt held pending.

A flow chart analyzing end-execute time is shown in FEMD, Figure 6077/78 (Basic) and 6291/92 (Accelerator).

INSTRUCTION FORMAT AND DECODING

- Four instruction registers are used: Op, Ra, Rb and Rc.
- Instruction format is defined by the op register bits 0, 1.

There are four instruction registers used in the IBM 2044 Processing Unit. These are:

Register	Bits	Format	Contents
Op	00-07	RR	Instruction op code
		RX	Instruction op code
		RS	Instruction op code
		SI	Instruction op code
Ra	08-11	RR	First operand GPR address
		RX	First operand GPR address
		RS	First operand GPR address
		SI	Bits 0 to 3 of immediate data
Rb	12-15	RR	Second operand GPR address
		RX	Index register GPR address
		RS	Index register GPR address
		SI	Bits 4 to 7 of immediate data
Rc	16-19	RX	Base GPR address
		RS	Base GPR address
		SI	Base GPR address

The alignment of the instruction with the instruction registers is shown in Figure 3A-1.

The first byte of an instruction is the operation (op) code. For convenience this is often expressed in hexadecimal form, and given a mnemonic for programming purposes, for example:

Mnemonic	Hexadecimal	Binary	Instruction
NR	14	00010100	RR binary AND
N	54	01010100	RX binary AND
CLI	95	10010101	SI compare logical

The second byte, bits 8 to 15 of the instruction halfword, is used as two four-bit registers in the RR, RS and RX formats, and is used as an immediate eight-bit operand in the SI format. These eight bits of immediate data in SI format are held in the Ra (4 bits) and Rb (4 bits) registers. When these registers are read out, there is a special data path to convey the byte to the ABC funnel.

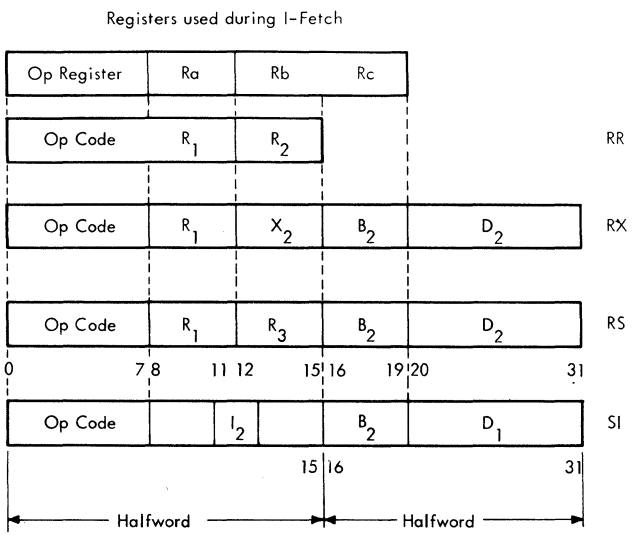


Figure 3A-1. Formats and Operation Registers

During I-fetch the eight bits of the op code are decoded to enable decisions to be made as to the format and the operation required.

The op decode is performed using the output from four decode groupings of three bits of the op register bits 0 to 7. The four combinations used are:

Op register bits 2, 4, 5
Op register bits 1, 3, 5
Op register bits 0, 6, 7
Op register bits 3, 6, 7

These lines are used to derive the controls necessary to define fully the instruction and its type.

The op decode logic is contained in ALD Pages DN 001 to 191. An example of op decoding for the RX format compare instruction follows:

Bit positions 0 1 2 3 4 5 6 7
Op code of RX compare 0 1 0 1 1 0 0 1

In this example, bits 2, 4 and 5 are 0, 1 and 0. The following shows the output of op register encoder positions for the four combinations used.

	Bits 2,4,5	Bits 1,3,5	Bits 0,6,7	Bits 3,6,7
000				
001			X	
010	X			
011				
100				
101				X
110		X		
111				

Reference to ALD Pages DN 001 to 191 shows that the following control lines are generated from these encoded lines:

ALD	Op Decode Line Name
DN 021	Compare Algebraic Op Subt + Comp + Load C-N-P Subt + Comp + Store + Test Set Add + Subt + Comp
DN 031	Load C-N-P + Arith + Log Ops Add Alg + Subt Alg + Comp Alg
DN 101	Arith + Log + Load Ops

These lines and the 'not' condition of other lines from these ALD pages provide the control for the instruction execution.

The op decode for all instructions is handled in a similar manner.

If the op code is not part of the Model 44 instruction set or is for a feature that is not installed, an 'operation exception' is signalled, causing a program interrupt (invalid operation).

If the op code represents a privileged instruction and the machine is in the problem state (PSW 1 bit 15 equals 1), a privileged-operation exception is signalled, again causing a program interrupt.

The Model 44 privileged instructions are:

Instruction	Op Code	Mnemonic
Halt I/O	9E	HIO
Load PSW	82	LPSW
Diagnose	83	DIAG
Set System Mask	80	SSM
Start I/O	9C	SIO
Test Channel	9F	TCH
Test I/O	9D	TIO

I-FETCH

- The instruction is read out of the locations defined by the Instruction Counter (IC).
- The instruction is aligned in the data flow.
- The Op, Ra, Rb and Rc registers are set from bits 0 to 19 of the data flow.
- The op decode is performed.
- Indexing, when required, is performed.
- Controls for E-phase are set up.
- In accelerator, I-fetch cycles are overlapped.

Instruction Read-Out

The conditions under which I-fetch can proceed have been described previously (see "Introduction to Instruction Fetching").

To start all I-fetch operations, an Instruction cycle (I-cycle) is taken. During this I-cycle the IC (PSW 2 bits 8 to 31) is used to address main storage, and the instruction field is read out to the Storage Data Register (SDR) as explained under "Main Storage Addressing" in Introduction and Functional Units, Form Y33-0001. The instruction is correctly aligned in the SDR if the instruction address specifies an address in the half of main storage addressed by Storage Address Register (SAR) 1. However, if the instruction address specifies an address in the half of main storage addressed by SAR 2, a criss-cross operation is performed to align the instruction correctly in the data flow.

The correctly aligned instruction is then available at the true/criss-cross output.

At the start of the I-cycle, a 'clear data flow' line is conditioned to reset all the internal registers and controls in the arithmetic and logic section of the machine. The Op, Ra, Rb and Rc registers are also reset at this time.

When the aligned instruction is available at the true/criss-cross output, bits 00 to 19 are used to set the Op, Ra, Rb and Rc registers.

Up to this stage, all instructions are treated alike and it is only after the Op, Ra, Rb and Rc registers have been set that instruction and op decoding can proceed.

The Op register is immediately decoded to determine the instruction format and type.

If the op is invalid, the operation-exception latch is turned on. If the op is a privileged operation and the machine is in the problem state (PSW 1 bit 15 equals 1), the privileged-operation-exception latch is turned on.

For both these exceptions an end-execute signal is generated at the end of the I-cycle, and the I-fetch is terminated. The machine then enters the end execute time analysis, shown in FEMD, Figures 6077/78 (Basic) and 6291/92 (Accelerator). A program interrupt would normally be taken under these exception conditions.

If the op code is valid, the instruction format together with the contents of the Rb and Rc registers are used to determine if indexing cycles are required.

The instruction format decode output is also used to determine if the displacement field should be gated to the B and C registers.

If the instruction format is not RR, the true/criss-cross output, bits 20 to 31, is gated through the ABC funnel and is set into the B and C registers. Bits 20 to 31 represent the displacement, and they are set into the B and C registers in readiness for the addition of the base address or the index field if indexing is required.

The conditions which cause either no index cycles, one index cycle or two index cycles are:

No Index		
	RR format	All instructions
	RX format	Rc = 0, Rb = 0
	RS, SI format	Rc = 0
One Index		
	RX format	Rb = 0, Rc $\neq$ 0
	RX format	Rb $\neq$ 0, Rc = 0
	RS, SI format	Rc $\neq$ 0
Two Index		
	RX format	Rb $\neq$ 0, Rc $\neq$ 0

No Index Cycles

BASIC only: For the three conditions causing no-index cycles, I-fetch can end at the completion of the I-cycle.

For the RR format, the information required for the E-phase is located in the Op, Ra and Rb registers. For the RX format, the information required for the E-phase is located in the Op, Ra, Rb, Rc and B registers. The B register contains the displacement, which, as no indexing cycles were required, is also the effective address.

For the RS and SI formats, the information required for the E-phase is located in the Op, Ra, Rb and B registers. The B register contents again represent the effective address.

For these no-index conditions the last-instruction-cycle signal is generated during the I-cycle and the op decode is used to set the appropriate controls for the E-phase of the instruction. (See "Last Instruction Cycle Conditions" section.)

ACCELERATOR only: For the RR format, indexing is never required. The op decode output signals that the instruction is in RR format, identifies the op code, and the instruction execution is allowed to proceed.

As all operands in RR format are in GPR's, no storage cycles are required and the instruction execution can be performed in parallel with the remainder of the I-cycle on which the instruction information in SDR is regenerated in storage.

For RR instructions, the common signals for I-fetch occur in the first part of the I-cycle only. In FEMD, Figure 6201/2, the timing charts for the RR I-fetch show these common signals. The timing charts for all RR-type instructions start at the beginning of I-fetch to illustrate the overlap of I-fetch and E-phase relevant to the instruction.

The common I-fetch signals for each instruction are not shown on the timing chart, but a cross reference is made to the RR I-fetch chart.

On the I-cycle, regardless of format, the compute clock is started at RC4 time. This first run of the compute clock uses the 'I-time' latch as a control, and a sequence latch is not activated unless a second run of the compute clock is required.

For the RX, SI or RS formats and the no-index conditions previously listed, I-fetch can be concluded, as the displacement in the B register represents the effective address.

As the E-phase of instructions in these three formats can start after the I-cycle or after one or two index cycles, the E-phase is not normally overlapped with I-fetch.

For the no-index conditions, the E-phase starts, using the controls set during the I-cycle by the op decode networks.

A functional op decode flowchart is provided in FEMD, Figure 6201/2 to enable controls set at the end of I-fetch to be established.

One Index Cycle

BASIC only: For the one-index condition, a B-cycle or an X-cycle is taken to read out the appropriate General-Purpose Register (GPR) contents to be added to the displacement. It is only after this cycle is complete that I-fetch can be ended.

The cycle taken for each of the one-index conditions is as follows:

Format	Conditions	Cycle
RX	Rb = 0, Rc $\neq$ 0	B-cycle
RX	Rb $\neq$ 0, Rc = 0	X-cycle
RS, SI	Rc $\neq$ 0	B-cycle

The contents of the GPR defined by the address in Rb for the X-cycle and Rc for the B-cycle are read out to the SDR on this indexing cycle.

The SDR is then gated to the A, B and C registers via the true/criss-cross and the ABC funnel. The Carry Look-Ahead (CLA) output is then gated to the B register, forming the sum of the displacement and base or index register contents in the B register.

Thus, at the end of this cycle, the effective address is contained in the B register and the I-fetch can be ended by signalling 'last instruction cycle'.

The controls set on this cycle in preparation for the E-phase of the instruction are described in the "Last Instruction Cycle Conditions" section.

ACCELERATOR only: The conditions that cause one index cycle have been listed previously.

If these conditions are present, the compute-clock cycle started at WC1 of the I-cycle is used to add the index information to the displacement.

For the RX format, if the Rb register is zero, the base address is read from the GPR defined by the Rc register, and if the Rc register is zero, the indexing address is read from the GPR defined by the Rb register.

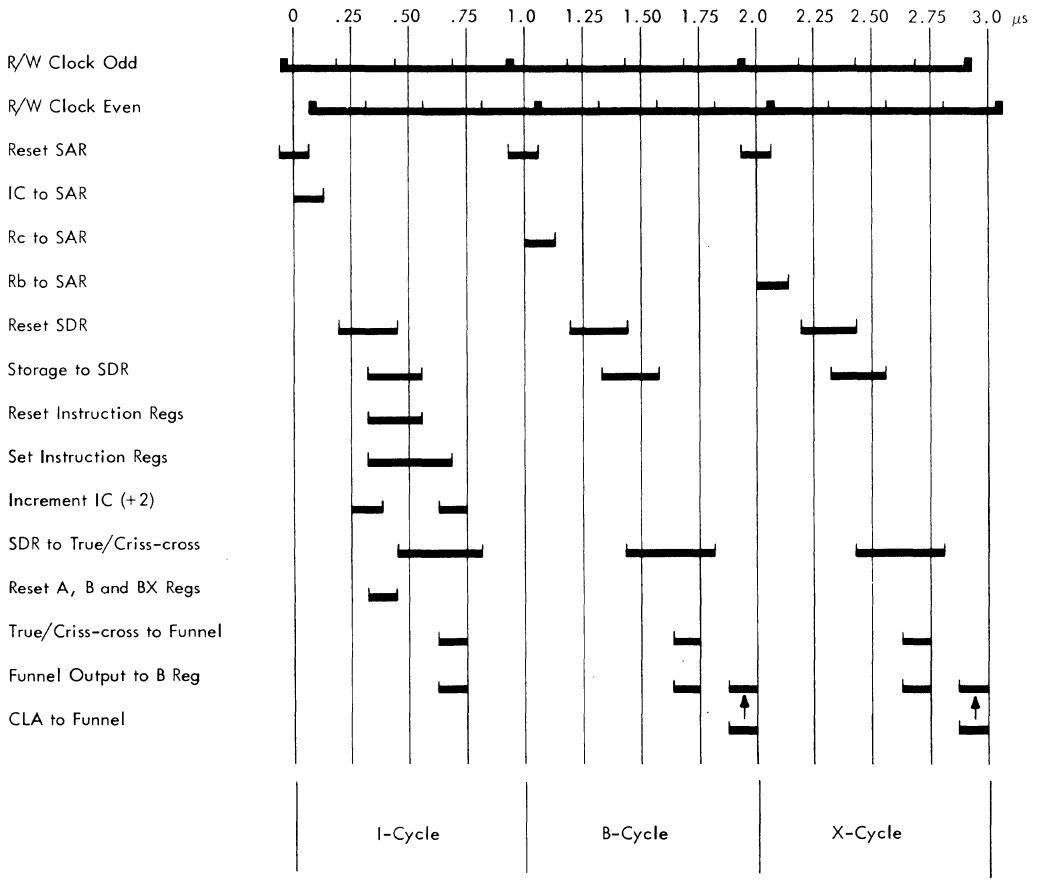


Figure 3A-2. I-Fetch Timings, Double Index (Basic Machine Only)

For RS or SI format, the base address is read out of the GPR defined by the Rc register.

At the completion of the compute-clock cycle, the first index latch is turned on and the E-phase can start, using the controls set by the op decode networks.

The flowchart and timing chart for these cycles are shown in FEMD Figure 6203/4. A flowchart for the controls set by the op decode is also provided.

Two Index Cycles

BASIC only: For the two-index condition, both the base address and the index field must be added to the displacement before the I-fetch can be ended. This requires a B-cycle and an X-cycle, with the two cycles taken in that order.

The operation for the B-cycle is similar to the one-index cycle in that the base register defined by the Rc register is read and its contents added to the displacement in the B register.

The X-cycle uses the Rb register to define the GPR to be read out to the SDR.

The arithmetic and logic section of the machine is prepared for the second addition process by resetting the A register and copying the B register contents into the C register.

The indexing register content is then gated from the SDR, via the true/criss-cross and the ABC funnel, to the A, B and C registers. The CLA output is then gated to the B register to form the effective address in the B register. The timing of these controls is shown in Figure 3A-2.

The last instruction cycle can then be generated and the controls generated for the E-phase of the instruction. These controls are described in the following section.

ACCELERATOR only: As listed previously, two-index conditions occur if Rb and Rc are non-zero on an RX-format operation.

For these conditions, the compute-clock cycle started at WC1 of the I-cycle is used to add the base register contents to the displacement, and a second compute-clock cycle is called to add the index register contents to the displacement.

To define the cycles, the first index latch is turned on near the completion of the first compute-clock cycle, and the second index latch is turned on near the completion of the second compute-clock cycle.

At the completion of this second cycle, the E-phase proceeds, using the controls set up by the op decode network.

A flowchart and timing chart of the I-fetch two-index conditions and a flowchart of the controls set by the op decode are provided in FEMD, Figures 6205/6 and 6207/8.

The last instruction cycle can then be generated and the controls generated for the E-phase of the instruction. These controls are described in the following section.

Last Instruction Cycle Conditions

The last instruction cycle signal and the op decode combine to set the controls required for the E-phase of the instruction

The method of op decoding has been described previously and the signals generated from this area set the appropriate cycle and/or sequence controls for the following cycle.

Flow charts and timing charts relating to I-fetch are contained in FEMD, Figures 6001/2, 6003/4, 6005/6 and 6007/8 for basic machines and 6201/2, 6203/4, 6205/6 and 6207/8 for accelerator machines. A functional op decode flowchart shows the controls that are set on this last instruction cycle; it does not show the actual method the machine uses to determine the controls to be set, but enables them to be correctly established. The exact method can then be determined from the ALD's.

HALFWORD EXPANSION

- Halfword is expanded to appear as a full word to enter the ABC funnel.
- Halfword operation allows ones out of bits 0 to 15 of true/criss-cross.
- Admission of forced ones to ABC funnel is dependent on bit 16 (sign bit).

Figure 3A-3 is a schematic representation of the halfword expansion.

When an operand has been read out from main storage to SDR, and analysis of the address and operation code shows it to be a halfword operation, the true/criss-cross is used to align the data in the low-order 16 bits at the ABC funnel entry.

The true/criss-cross is constructed in negative logic so that if the high-order 16 bits are not controlled for use, as during halfword operations, ones are propagated from these unused positions. These ones may or may not be admitted to the ABC funnel, depending on the state of bit position 16 (the sign bit) of the halfword operand. If bit 16 is a zero (positive sign), the forced ones are not used. If bit 16 is a one (negative sign), the forced ones are allowed to enter positions 0 to 15 of the funnel with the halfword from main storage occupying the low-order bit positions 16 to 31.

To summarize, a halfword from main storage with a positive sign appears in the funnel in the form:

0000 0000 0000 0000 0101 1011 1101 0111

A halfword with a negative sign appears in the form:

1111 1111 1111 1111 1101 1011 1101 0111

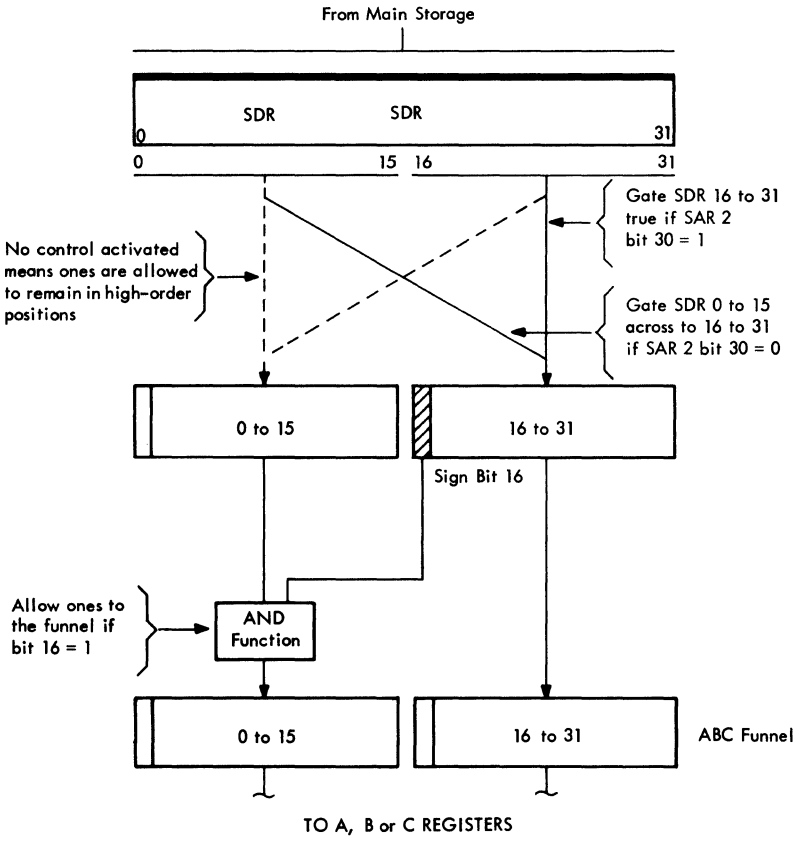


Figure 3A-3. Halfword Expansion

LOAD INSTRUCTIONS

Load, Load Halfword, Load and Test

- The load instruction is either RR or RX format.
- The respective op codes and mnemonics are 18 (LR) and 58 (L).
- The load-halfword instruction is RX format.
- The op code and mnemonic is 48 (LH).
- The load and test instruction is RR format.
- The op code and mnemonic is 12 (LTR).
- The second operand is placed in the first operand location.
- The second operand remains unchanged.
- On load halfword, the halfword is expanded to a full word by propagating the sign bit in the 16 high-order positions.
- The condition code is set only for 'load and test' instructions as follows:
  00 : Result is zero
  01 : Result is less than zero
  10 : Result is greater than zero
  11 : Not set
- In accelerator, GPR's go to B register via HW funnel.

BASIC only: The load, load halfword and load and test operations cause the second operand to be read out of storage and set in the SDR. From there it is gated through the true/criss-cross to the B and C registers. The true/criss-cross controls are set to gate SDR true for the load, and load and test instructions. For the load halfword instruction the true/criss-cross controls are used to align the halfword to the right-hand half of the B and C registers, and to propagate the sign bit to the left-hand half of these registers.

The first operand is then read out of storage but is not gated to the SDR during the read section of the storage cycle. Instead, the second operand is gated from the B register to the SDR, and is stored from there into the first operand location during the write section of the storage cycle.

The load and test instruction causes the B register to be analyzed and the condition code to be set.

The sequence and timing of these instructions are shown in FEMD, Figure 6009/10.

RX Format

ACCELERATOR only: The instructions of this group in the RX format are load (op code 58) and load halfword (op code 45).

For these instructions an EA-cycle is taken and the second operand is read out to the SDR. It is then gated to the B register (aligning and expanding the operand for the halfword operand condition).

From the B register, the second operand is gated to the GPR defined by the Ra register and 'end execute' is signalled.

RR Format

ACCELERATOR only: The instructions of this group in the RR format are load (op code 18) and load and test (op code 12).

For these instructions, the second operand is gated from the GPR defined by Rb and set into the B register using the compute clock started during I-time.

The compute clock is stopped at CC4 and 'end execute' is signalled.

Sequence 2 is then called and the compute clock restarted. The B register is gated to the GPR defined by Ra and the condition code is set for the load and test instruction early in this compute cycle.

Note that 'end execute' is signalled prior to starting the second compute cycle. This gives an overlap of the next I-fetch or interrupt cycle with the E-phase of this instruction

Load Complement, Load Positive, Load Negative

- The load complement, load positive, and load negative instructions are in RR format.
- The respective op codes and mnemonics are 13 (LCR), 10 (LPR), 11 (LNR).
- The second operand is loaded into the first operand GPR in the form specified by the instruction.
- The condition code is set from the result stored.

The load complement, load positive, or load negative instructions cause the second operand to be read out of storage and set into the SDR. The B and C registers are set to all ones, and the second operand is gated to the A, B and C registers.

Thus, the A register contains the second operand in true form and the B and C registers contain the second operand Exclusive OR'ed with ones (that is, the one's complement of the second operand).

The instruction is then analyzed to determine the operation to be performed.

Load Complement

- The two's complement of the second operand is stored.

For this operation the A register is reset and the subtract trigger is set. A subtract operation is then performed by adding the A register contents to the B register contents with the carry-in forced by the subtract trigger. This forms the two's complement of the second operand in the B register.

Load Positive

- The absolute value of the second operand is stored.

  BASIC only: The SDR sign is tested. If it is negative, the A register is reset and a subtract operation is performed similar to that described for load complement. Thus, if the second operand is negative, it is complemented, the result being formed in the B register.

  If the SDR is positive, the B and C registers are reset and the A register is gated via the ABC funnel to the B and C registers.

  Thus, the B register contains, in both cases, the absolute value of the second operand.

  ACCELERATOR only: The A register is tested. If it is negative, the A register is reset and a subtract operation is performed similar to that described for load complement. Thus, if the second operand is negative, it is complemented, the result being formed in the B register.

  If the A-register sign is positive, the B and C registers are reset and the A register is gated via the ABC funnel to the B and C registers.

  Thus the B register contains, in both cases, the absolute value of the second operand.

Load Negative

- The two's complement of the absolute value is stored.

  BASIC only: The conditions that apply for load positive are reversed for load negative. If the sign of the second operand in the SDR is negative, the B and C registers are reset and the A register is gated via the ABC funnel to the B and C registers.

  If the second operand in the SDR is positive, the A register is reset and a subtract operation is performed, forming the two's complement of the second operand in the B register.

  Thus, the B register will in both cases contain the two's complement of the absolute value of the second operand.

  ACCELERATOR only: The conditions that apply for load positive are reversed. If the sign of the second operand in the A register is negative, the B and C registers are reset and the A register is gated via the ABC funnel to the B and C registers.

  If the second operand in the A register is positive, the A register is reset and a subtract operation is performed, forming the two's complement of the second operand in the B register.

  Thus the B register contains, in both cases, the two's complement of the absolute value of the second operand.

Result Storage Cycle

BASIC only: For all these instructions the first operand is then read out, but not set to the SDR. The content of the B register is set to the SDR. The SDR is then stored in the first operand location during the write section of the storage cycle.

ACCELERATOR only: When 'end execute' is signalled, the B register is gated to the GPR defined by Ra.

The B register is also analyzed to determine the setting of the condition code as follows:

Load Complement
- 00 : Result is zero
- 01 : Result is less than zero
- 10 : Result is greater than zero
- 11 : Fixed-point overflow

Load Positive
- 00 : Result is zero
- 01 : Not set
- 10 : Result is greater than zero
- 11 : Fixed-point overflow (C0 $\neq$ C1)

Load Negative
- 00 : Result is zero
- 01 : Result is less than zero
- 10 : Not set
- 11 : Not set

For the fixed-point overflow condition, the fixed-point overflow exception latch is turned on if the mask bit (PSW 2 bit 4) is a one. This latch generates a program interrupt request (refer to "Interrupts" section).

The sequence and timing of these instructions are shown in FEMD, Figures 6011/12 (basic) and 6213/14 (accelerator).

STORE AND STORE-HALFWORD INSTRUCTIONS

- The store and store-halfword instructions are both in RX format.
- The respective op codes and mnemonics are 50 (ST) and 40 (STH).
- The first operand is stored in the second operand location.
- The store-halfword instruction stores the 16 low-order bits of the first operand GPR (bits 16 to 31).
- The condition code is not altered.
- In accelerator, GPR's enter the B register via HW funnel.

BASIC only: The store and store-halfword instructions are both in RX format. This means that the developed effective address is in the B register at the start of the E-phase of these instructions.

The objective of these instructions is first to read out the first-operand GPR, and then to store the full 32 bits (store) or bits 16 to 31 (store halfword) into the storage location defined by this effective address. This means that the first cycle must be an R1 cycle, and that the effective address must be stored out of the main data flow during this cycle. The second, or store cycle, is an EA cycle and for this cycle the effective address is temporarily returned to the data flow so that this information can be set into the SAR's for addressing purposes.

The sequence and timing of the store and store-halfword instructions are shown in FEMD, Figure 6013/14.

For the store instruction, the machine takes an R1 cycle and reads out the GPR defined by the Ra register from extension storage to the SDR. The B and BX registers are interchanged, placing the effective address in the BX register and zeros in the B register.

The SDR is gated to the B register and the B and BX registers are re-interchanged to place the effective address back in the B register in preparation for the EA cycle.

The machine takes an EA cycle using the B register to provide the information for the SAR's. The full word defined by the effective address is read out of main storage, and the SDR reset, but the storage to SDR gate is inhibited. The B register is gated to the SDR and stored on the write section of the EA cycle.

For the store halfword instruction, the machine takes an R1 cycle, reads out the GPR defined by the Ra register to the SDR, and interchanges the B and BX registers in the same way as described for the store instruction.

The halfword to be stored is contained in bits 16 to 31 of the SDR. This is aligned to the appropriate half of the B register by gating SDR bits 16 to 31 true if BX bit 30 is a one, and SDR bits 16 to 31 criss-cross if BX bit 31 is a zero. The B and BX registers are interchanged, placing the effective address back into the B register in preparation for the EA cycle.

The machine takes an EA cycle, as with the store instruction, but only 16 bits of the SDR are inhibited and the corresponding 16 bits of the B register are gated to those positions of the SDR.

If the SAR 2 bit 30 is a zero, the 16 bits are bits 00 to 15; if it is a one, the 16 bits are bits 16 to 31.

The SDR is then stored on the write section of the EA cycle.

ACCELERATOR only: The store instructions are started by the machine taking an EA cycle.

The SDR is not set with the effective-address information. That is, for the store instruction it is not set at all, while for the store-halfword instruction, the halfword to be regenerated to storage is set into the SDR.

The second operand information is gated to the B register. For the store-halfword instruction the halfword is aligned to the appropriate half of the B register by entering the Hardware (HW) funnel, bits 00 to 15, into the ABC funnel, bits 00 to 15 or 16 to 31 if the effective address bit 30 is a zero or a one respectively.

At this time the EA cycle is split, the compute clock is started and the required bits are gated from the B register to the SDR.

The write clock is started at CC3 time and the information in the SDR is written into the storage locations defined by the effective address.

As the E-phase is controlled by a storage cycle, 'end execute' is signalled at WC4 time.

The flowchart and timing chart for these instructions are provided in FEMD, Figure 6215/16.

ADD, SUBTRACT AND COMPARE INSTRUCTIONS

The Model 44 follows all the principles of fixed-point binary operations, summarized as follows:

1. Halfword operands are expanded to full word by propagating the sign bit (see also "Halfword Expansion").
2. Positive binary numbers are represented in their true form with a zero in their high-order sign position.
3. Negative binary numbers are represented in their two's complement form with a one in their high-order sign position.
4. On add instructions, no complementing is done because negative operands are already in complement form. The two operands are true added.
5. On subtract instructions the second operand is always complemented (or recomplemented if negative) and the two operands are added.
6. A fixed-point overflow occurs if the carry into the sign position is not equal to the carry out of the sign position.

NOTE: In accelerator machines, GPR's enter the B register via the HW funnel.

Flow charts and timing charts of the add, subtract and compare instructions are shown in FEMD, Figures 6015/16 and 6017/18 (basic) and 6217/18 to 6223/24 inclusive (accelerator).

ACCELERATOR only: For RR-format instructions, the E-phase is started during the I-cycle. The compute clock is started at RC4 of the I-cycle and controls the read-out of the second operand from the GPR defined by Rb, to the A, B and C register.

A sequence-2 compute cycle starts and the first operand is gated from the GPR defined by Ra, via the HW and ABC funnels to the A, B and C registers.

When 'end execute' is signalled, the B register content goes to the GPR defined by Ra. For RX-format instructions an EA-cycle is taken to read out the second operand to the SDR.

Add, and Add Halfword Instructions

- The add instruction is in either RR or RX format.
- The respective op codes and mnemonics are 1A (AR) and 5A (A).
- The add halfword instruction is in RX format.
- The op code and mnemonic are 4A (AH)
- The second operand is added to the first operand, and the result placed in the first-operand location.
- The result determines the setting of the condition code as follows:
  - 00: Result is zero
  - 01: Result is less than zero
  - 10: Result is greater than zero
  - 11: Fixed-point overflow
- A fixed-point overflow occurs if the carry from the high-order position (bit 1) fails to match a carry from the sign position (bit 0).

The add operation is fully described under "Arithmetic and Logic Section, and Registers" in Chapter 2 of Introduction and Functional Units, Form Y33-0001. Briefly, the operation is: clear A, B and C registers, set the second operand in B and C registers and then set the first operand into the A, B and C registers. The A, B and C registers consist of binary input triggers, so that the B and C registers now contain the EXOR sum of the operand without the carries. Carries are derived from the CLA unit by EXOR'ing the A and C registers. The CLA unit output is then EXOR'ed to the partial sum in the B register to produce the final sum in the B register. The state of the registers at the end of the operation is therefore:

A register, operand 1
B register, complete total
C register, addition total less the carries (partial sum).

BASIC only: Note that, as shown in FEMD Figure 6015/16, the R1 cycle is a split cycle to allow for the result to be formed before the write section of this cycle.

When a fixed-point overflow occurs, it is tested by AND'ing with PSW 2 bit 4. If PSW 2 bit 4 is a one, the 'fixed-point overflow exception' latch is set to give a 'program interrupt request'.

Subtract, and Subtract Halfword Instructions

- The subtract instruction is in either RR or RX format.
- The respective op codes and mnemonics are 1B (SR) and 3B (S).
- The subtract halfword instruction is in RX format.
- The op code and mnemonic are 4B (SH).
- The two's complement of the second operand is added to the first operand and the result is placed in the first-operand location.
- Halfword operands are expanded to full words.
- The result determines the setting of the condition code as follows:
  - 00: Result is zero
  - 01: Result is less than zero
  - 10: Result is greater than zero
  - 11: Fixed-point overflow
- A fixed-point overflow occurs if the carry out from the high-order position (bit 1) fails to match a carry out from the sign position (bit 0).

A full description of the subtract operation is given under "Arithmetic and Logic Section and Registers" in Chapter 2 of Introduction and Functional Units, Form Y33-0001. Subtraction is achieved by adding the two's complement of the second operand to operand 1. The A register is set to zero and the B and C registers are set to all ones. The second operand is EXOR'ed into the B and C registers. The first operand is gated into the A, B and C registers. The state of the registers at the end of the operation is, therefore:

A register, operand 1
B register, sum total
C register, inverse of partial sum.

The output of the B register is gated to the addressed internal unit.

Compare, and Compare Halfword Instructions

- The compare instruction is in either RR or RX format.
- The respective op codes and mnemonics are 19 (CR) and 59 (C).
- The compare halfword instruction is in RX format.
- The op code and mnemonic is 49 (CH).
- The second operand is compared with the first operand and the result determines the setting of the condition code.
- Operands remain unaltered in storage.
- Halfword operands are expanded to full words.
- The result determines the setting of the condition code as follows:
  - 00: Operands are equal
  - 01: First operand is low
  - 10: First operand is high
  - 11: Not set
- The comparison is done by subtracting operand 2 from operand 1 and analyzing the result.

The compare operation starts in the same way as a subtract operation, but differs in that the result is not stored and the condition code has an altered meaning.

BASIC only: In the compare operation, R1 is not split because the result does not have to be formed (the object of compare operation is satisfied when the condition code is set). Both operands are returned to storage unchanged.

ACCELERATOR only: The operands are gated back to their original locations unchanged.

SHIFT ARITHMETIC INSTRUCTIONS

- Instructions are in RS format.
- The R3 field is not used for these instructions.
- The low-order six bits of the developed effective address define the number of positions to be shifted (bits 26 to 31).
- The operands, which are signed integers, may be either single or double words, and are contained in the GPR or GPR pair defined by the R1 field of the instruction.
- The op codes and their mnemonics for the shift arithmetic instructions are:
  Shift left single : 8B (SLA)
  Shift left double : 8F (SLDA)
  Shift right single: 8A (SRA)
  Shift right double: 8E (SRDA)
- Shift single is a single-word operation and shifting is done in the B register.
- Shift double is a double-word operation and shifting is done in the B and BX registers.
- Multiple shift is a series of single-shift operations.
- The shift counter is reduced by one for each shift and shifting ends when the shift counter equals zero.
- For shift-right operations, the original sign bit is supplied to the high-order bit positions.
- For shift-left operations, zeros are supplied to the low-order bit positions.
- The result is stored in the original operand registers.
- Shift overflow is checked on shift-left operations.
- The condition code is set as follows:
  00: Result is zero
  01: Result is less than zero
  10: Result is greater than zero
  11: Shift Left : Overflow
  Shift Right: Not set
- In accelerator machines GPR's enter B register via HW funnel.

The four shift operations are shown in Figure 3A-4. Shifting is the operation of moving all the bits in a word or double word to the right-hand or left-hand adjacent bit position respectively without disturbing their order.

Shift single is a full-word operation, and the operand is fed into the B register. There is no limitation for the addressing of the register.

Shift double is a double-word operation and the operand is the pair of even and odd addressed registers. Addressing of the operand should always be made on the even-numbered register, otherwise the specification exception occurs and causes a program interrupt. The machine automatically addresses the next odd-numbered register. The even register address contains zeros in bit positions 29, 30 and 31. The odd address number is developed by forcing a 1 bit in SAR bit position 29 on the first-operand fetch cycle.

The odd-numbered register content, which is the low-order word, is fed into the BX register via the B register. The even-numbered register content, which is the high-order word, is then fed to the B register.

The low-order six bits (bits 26 to 31) of the instruction format indicate the number of shift places, and are fed into the shift counter before the operand is fetched.

Multiple shift is a series of single-shift operations in which the shift counter value is reduced by one at each shift. Shifting ends when the shift counter is zero.

On shift double, when the B register is shifted right, bit 31 of the B register is shifted into the bit 00 position of the BX register. Bit 31 of the BX register is defined as the low-order bit.

In shift-right operations the low-order bit is shifted out without inspection and is lost. Bit 00 of the B register (sign bit) goes to bit position 01 of the B register in the same way as any other bit. A bit equal to the original sign bit is supplied to the vacated bit position 00 of the B register as follows. The A register contains the original operand in a shift-single operation or the high-order word in a shift-double operation. Therefore, the original sign bit is stored in bit position 00 of the A register and this bit is fed into bit position 00 of the B register by each shift.

In a shift-left operation, bit 00 of the BX register is shifted into bit position 31 of the B register. Bit 01 of the B register goes to bit position 00 of the B register in the same way as any other bit. Therefor, if these two bit positions have different bit status, the operand is changed from positive to negative or the reverse. Note that the sign bit is corrected in the operand store cycle as explained later.

Bit positions 00 and 01 of the B register are examined for the overflow condition on every shift. If the status of these bits is different on a shift-left, an

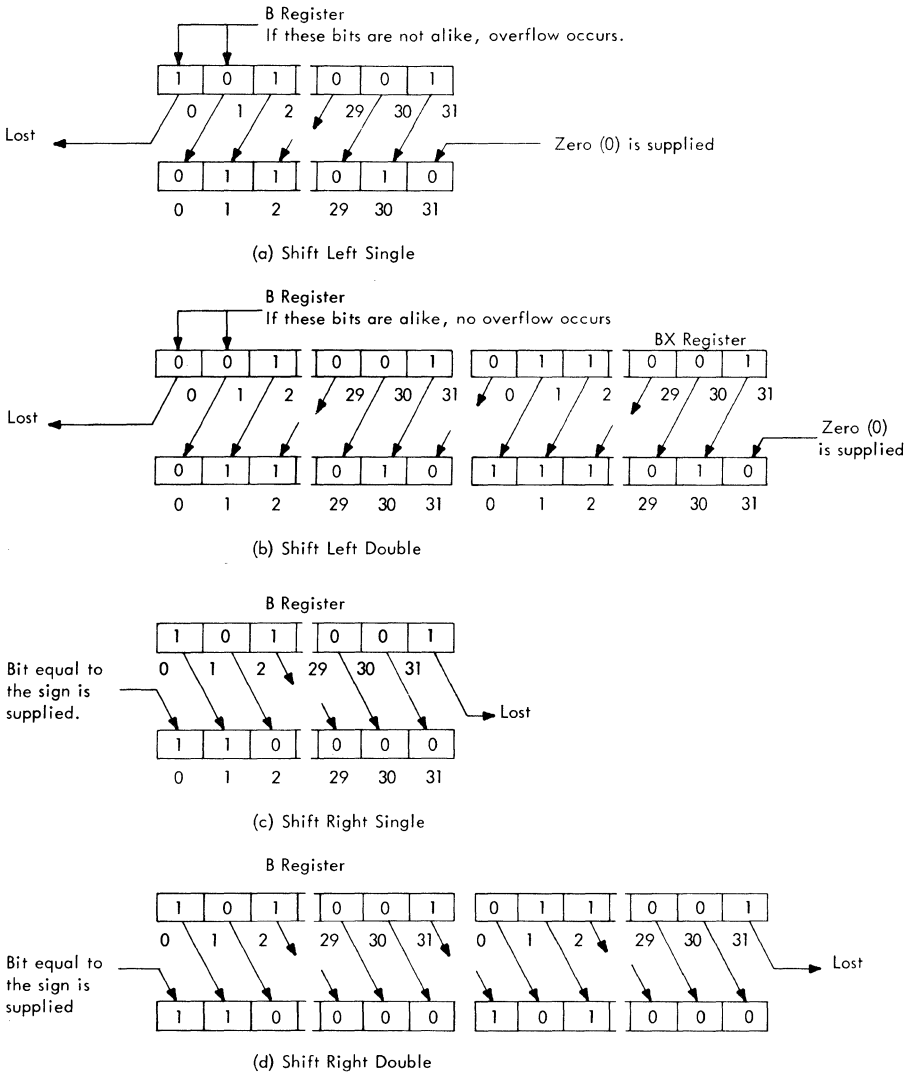


Figure 3A-4. Shift Operations -- Arithmetic

overflow occurs and sets the fixed-point overflow exception latch if the corresponding mask bit is a one. When the fixed-point overflow line becomes active it is tested against PSW 2 bit 4. If PSW 2 bit 4 is a one the overflow exception latch is set to allow a program interrupt request. (Refer to the "Checking" section in Chapter 2 of Introduction and Functional Units Form Y33-0001 and the "Interrupts" section in this manual.)

BASIC only: When the shift counter is greater than two, the compute clock wraps at CC2. In both shift-left and shift-right operations, the B and BX registers are shifted one position. In either case the shift counter is reduced by one for each wrapped cycle.

When the shift counter reaches two, the compute clock runs to CC6 instead of wrapping, but still performs the shift twice, once at CC3, CP1 time and once at CC5, CP1 time.

If the shift counter is set to two at the beginning of this operation, the compute clock runs to CC6, performing the shift operation twice, once at CC1, CP1 time and once at CC3, CP1 time. If the shift counter is set to one at the beginning of the operation, the compute clock runs to CC6, performing the shift operation once at CC1, CP1 time. If the shift counter is set to zero, the compute clock runs to CC6, but no shift takes place.

Zeros are supplied to the low-order position (bit 31) for the shift-left operation. This is the B register bit 31 for a shift-left-single and the BX register for a shift-left-double operation. The original sign bit is supplied into the bit 00 position of the B register in a shift-right opera-

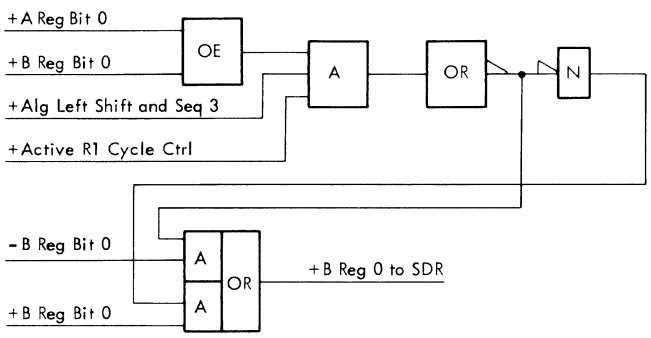


Figure 3A-5. Sign Correction

tion. The simplified diagram is shown in Figure 3A-5.

The result is stored during the write section of a second R1 cycle for single shifts, and during the write section of a second set of double R1 cycles for double shifts. The addressing method is the same as in the operand-fetch cycle previously described.

The condition code is set as follows:

00: Result is zero
01: Result is less than zero
10: Result is greater than zero
11: Fixed-point overflow (shift-left only; not used on shift-right.)

During shift-left-double operation the condition code is set at sequence latch 4 time. At this time only the B register has been analyzed. However, the BX register may contain a small positive number so that if the condition code setting is 00 (result is zero), a chance is given to test again during sequence latch 5 time. The result may remain zero or it may be amended to 10 (result is greater than zero).

The flow charts and timing charts for each of the shift operations are contained in FEMD Figures 6019/20, 6021/22, 6023/24 and 6025/26.

ACCELERATOR only: When the shift counter is greater than zero, the compute clock wraps at CC2. In both shift-left and shift-right operations, the B and BX registers are shifted one position. In either case the shift counter is reduced by one for each wrapped cycle. When the shift counter reaches zero, the compute clock runs to CC6.

If the shift counter is set to one at the beginning of this operation, the compute clock runs to CC6, performing the shift operation at CC1, CP1 time. If the shift counter is set to zero, the compute clock runs to CC6, but no shift takes place.

Zeros are supplied to the low-order position (bit 31) for the shift-left operation. This is the B register bit 31 for a shift-left-single and the BX register bit 31 for a shift-left-double operation. The original sign bit is supplied into the 00 position of the B register in a shift-right operation. A simplified diagram of the sign-correction circuit is shown in Figure 3A-5.

The result is stored during the CC3 section of the compute cycle for single shifts, and during the CC3 section of both sequence-3 and sequence-4 cycles for double shifts.

The condition code is set as follows:

00: Result is zero
01: Result is less than zero
10: Result is greater than zero
11: Fixed-point overflow (shift-left only; not used on shift-right)

For fixed-point overflow conditions on the shift-left instructions when the mask bit PSW 2 bit 4 is one, the 'fixed-point overflow exception' latch is set and a program interrupt is requested. (Refer to the "Interrupts" section of this manual.)

The flow charts and timing charts for each of the shift instructions are contained in FEMD, Figures 6225/26 to 6231/32 inclusive.

MULTIPLY AND MULTIPLY-HALFWORD INSTRUCTIONS

- The multiply instruction is either RR or RX format.
- The respective op codes and mnemonics are 1C (MR) and 5C (M).
- The multiply-halfword instruction is in RX format.
- The op code and mnemonic is 4C (MH).

Principles of Multiply Operation

The multiply operation is performed by successively shifting the multiplicand relative to the product, and executing an arithmetic operation with the multiplicand on the product field as determined by the multiplier digit. The multiplier is scanned from right to left, or low-order bit to high-order bit.

A simple binary multiply would cause an arithmetic add of the multiplicand to the product field each time the multiplier digit is a one, and, if the multiplier digit is a zero, it would cause a shift to the next position (no arithmetic operation).

This is illustrated in the following example:

```
 21 x          10101 x
  5              101
---           -----
105           10101
             10101
            -------
            1101001
```

Note that, for this simple binary multiplication, the number of combined shift and arithmetic cycles equals the number of ones in the multiplier, and the number of shift-only cycles equals the number of zeros in the multiplier.

The number of cycles using this simple type of multiply for some sample multipliers is shown as follows:

Multiplier		Cycles	
Decimal	Binary	Shift and Arithmetic	Shift Only
1	0000 0001	1	7
4	0000 0100	1	7
7	0000 0111	3	5
8	0000 1000	1	7
15	0000 1111	4	4
27	0001 1011	4	4
32	0010 0000	1	7

In the Model 44 the shift-only cycles can be performed in 250 nanoseconds whereas the combined shift and arithmetic cycle takes 750 nanoseconds. This means that if the number of arithmetic cycles is reduced, a significant saving in time is achieved.

The number of arithmetic cycles is reduced by the use of the carry-look-ahead group-of-ones principle.

Basically this principle means that if there is a multiplier of, for example, +15 (001111), a multiplier of 16 (010000) is used and added to the product field and a multiplier of 1 (000001) is used and subtracted from the product field.

This means that if a group of ones occurs in the multiplier field, an arithmetic subtract operation occurs when the multiplier digit is the first one of the group (the subsequent ones of the group cause shift-only cycles), and an arithmetic add operation is performed for the first zero after the group of ones.

Reviewing this for the preceding example of the multiplier of 001111 (+15), the cycles are:

Multiplier Digit	Cycle Type	Effective Multiplier
1	Shift and Subtract	- 1
1	Shift only	0
1	Shift only	0
1	Shift only	0
0	Shift and add	+ 16
0	Shift only	0

This gives an effective multiplier of +15. Note that only two arithmetic cycles are used as opposed to the four that are required for the simple binary multiply.

An extension of the above rule occurs when a single zero appears within a group of ones. If an example of +27 is taken (0011011), then using the principles described above, the cycles are:

Multiplier Digit	Cycle Type	Effective Multiplier
1	Subtract	- 1
1	Shift only	0
0	Shift and add	+ 4
1	Shift and subtract	- 8
1	Shift only	0
0	Shift and add	+ 32
0	Shift only	0

The effective multiplier is +27 and four arithmetic cycles were taken.

If, however, the single zero within a group of ones is regarded as only a temporary interruption to the group, and each time such a zero becomes the multiplier digit an arithmetic subtract cycle is taken, then the same result is obtained with less arithmetic cycles. (Note that the group of ones is now considered to span over single zero after the first two consecutive ones.)

Using the same multiplier of +27 the cycles are:

Multiplier Digit	Cycle Type	Effective Multiplier
1	Subtract	- 1
1	Shift only	0
0	Shift and subtract	- 4
1	Shift only	0
1	Shift only	0
0	Shift and add	+ 32
0	Shift only	0

Once again the effective multiplier is +27, but the arithmetic cycles have been reduced to three.

For single zeros and ones outside the group of ones, the simple rules of binary multiplication apply.

Thus, for a more complex multiplier of +1869 (0111 0100 1101) the cycles are:

Multiplier Digit	Cycle Type	Effective Multiplier
1	Add	+ 1
0	Shift only	0
1	Shift and subtract	- 4
1	Shift only	0
0	Shift and add	+ 16
0	Shift only	0
1	Shift and add	+ 64
0	Shift only	0
1	Shift and subtract	- 256
1	Shift only	0
1	Shift only	0
0	Shift and add	+ 2048

The effective multiplier is:
(2048 + 64 + 16 + 1) - (256 + 4) = 2129 - 260 = + 1869
Note that with this example the saving in arithmetic cycles is only one, but for multipliers with bigger groups of ones a larger saving is achieved.

The principle holds true for negative multipliers, as the sign-bit can always be regarded as the first

one of a group of ones, because, if additional high-order positions are added to the multiplier field they would be filled with bits equal to the sign-bit.

For example, a multiplier of 1101101 (-19) would cause the following cycles:

Multiplier Digit	Cycle Type	Effective Multiplier
1	Add	+1
0	Shift only	0
1	Shift and subtract	-4
1	Shift only	0
0	Shift and subtract	-16
1	Shift only	0
1	Shift only	0

The effective multiplier is -19.

Another example, for a multiplier of 10011010 (-102), follows. This illustrates the use of the sign bit as the first one of a group of ones. (The multiplier can be treated as being 1111 1011 1010, which is also equal to -102.)

Multiplier Digit	Cycle Type	Effective Multiplier
0		0
1	Shift and add	+2
0	Shift only	0
1	Shift and subtract	-8
1	Shift only	0
0	Shift and add	+32
0	Shift only	0
1	Shift and subtract	-128

Thus the effective multiplier is -102.

The rules of multiplication in the Model 44 are summarized in the following paragraphs.

A group of ones is defined as two or more ones together, scanning from right to left, which may be interspersed with single zeros. A group commences with two ones together and terminates with two zeros together. For example, such a group is 00101011. The multiplication rules are:

1. If the multiplier digit is a one and is
   (a) First of a group of ones: Subtract
   (b) Not the first of a group of ones: Shift only
   (c) An individual one: Add
2. If the multiplier digit is a zero, and is
   (a) A zero within a group of zeros: Subtract
   (b) A zero terminating a group of ones: Add
   (c) A zero part of a group of zeros: Shift only

Note that, if the multiplier is shifted right in a register on every cycle, the multiplier digits and the next high-order digit of the multiplier will be contained in the two low-order positions of this register. When both these positions become ones for the first time, this signals the start of a group of ones. When both these positions subsequently become zero, the group of ones has been completed.

Thus, the number of arithmetic cycles required on the multiply will be reduced to 16 or less. This represents a considerable time saving, as shift-only cycles take 250 nanoseconds whereas arithmetic cycles take 750 nanoseconds.

Instruction Objectives — Multiply Instruction

The multiply operation causes the product of the multiplier (second operand) and the multiplicand (first operand) to be generated and to be placed in the multiplicand field. Both the multiplier and multiplicand are 32-bit signed integers. The product is a 64-bit signed integer and is placed in the even/odd register pair specified by the R1 field of the instruction. A specification exception occurs when R1 specifies an odd-numbered register.

The multiplicand is located in the odd register of the even/odd register pair specified by R1. The original content of the even-numbered register is ignored, except in the case of an RR format when the register is specified as the multiplier register.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand signs, with the exception that a zero product is always a positive result. An overflow cannot occur since the product can always be expressed as a 64-bit signed integer. The condition code will not be altered by the multiply operation.

Execution Sequence — Multiply Instruction

The multiply instruction is executed in the following steps:

1. Operand fetch and initialize cycles
2. Multiply cycles, shift counter 32 to 1 inclusive.
3. Multiply cycle, shift counter = 0
4. Product store cycles.

The multiply instruction timing charts and associated flow charts showing the sequence of the multiply instruction are shown in FEMD, Figures 6027/28 to 6031/32 inclusive for basic machines and 6233/34 to 6237/38 inclusive for accelerator machines.

Operand Fetch and Initialize Cycles

- Fetches the multiplier and places it in the BX register.
- Fetches the multiplicand and places it in the A register.
- Initializes the multiply controls and sets the shift counter to 32.

The multiply instruction can be in either the RR or RX instruction format. For the RX format the multiplier is fetched from the storage location defined by the developed effective address. For the RR format

the multiplier is fetched from the GPR defined by the Rb field of the instruction. In both cases it is gated to the B register and from there set in the BX register by a B to BX register interchange.

For both formats the multiplicand is fetched from the GPR defined by R1 field of the instruction and gated to the A register.

During these operand-fetch cycles the multiply controls are initialized and the shift counter is set to a value of 32.

Multiply Cycles — Shift Counter 31 to 1 Inclusive

- Forms the partial product in the B register.
- Shifts right the partial product at the beginning of all cycles except the first.
- Value of the shift counter reduced on each cycle.
- On each cycle the multiplicand is either ignored, added to, or subtracted from the partial product field under control of the multiplier digit.
- When the multiplicand is ignored, the cycle is a right-shift only and the clock is wrapped at CC2.
- On arithmetic cycles (CC1 to CC6) the state of the subtract trigger and the A register determines whether an add or subtract is performed.
- Overflow conditions and negative results on the arithmetic operation cause a high-order bit to be inserted on the subsequent shift-right operation.
- The cycles are defined by the value of the shift counter at the end of the cycle.

The operand-fetch cycles set the multiplicand to the A register and the multiplier to the BX register. The multiplier digit is in the BX register bit 31, and the BX register bits 30 and 31 are used to signal the beginning and end of a group of ones. The multiplier digit and this group-of-ones condition are analyzed to determine whether the multiply cycle is to be a shift-only cycle, an add cycle or a subtract cycle.

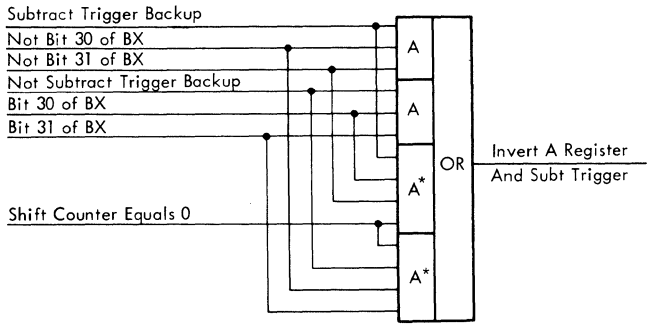
For arithmetic cycles the multiplicand in the A register is either added to, or subtracted from, the partial product in the B register.

On each cycle except the first, the B and BX registers are shifted right and the partial product that is progressively shifted into the BX register causes a new multiplier digit to be placed into the BX register bit 31 and the old multiplier digit to be lost. The 64-bit product will eventually be formed in the B and BX registers.

The BX register bits 30 and 31, in conjunction with the subtract trigger and the subtract-trigger-backup latch, provide the major controls for the multiply operation. The subtract trigger is the trigger which is used to provide the CLA carry-in and to signal that complement arithmetic is being performed. The subtract-trigger-backup latch is set to match the status of the subtract trigger early in each cycle and is used to hold, for a complete multiply cycle, the condition of the subtract trigger from the preceding cycle. This is necessary as the state of the subtract trigger from the preceding cycle is required for analysis and may change before the analysis can be performed.

The subtract trigger and the contents of the A register are inverted together and when the subtract trigger is on, and an arithmetic cycle is performed, the cycle will be a subtract cycle. If the subtract trigger is off the cycle will be an add cycle. The logic for the inversion of the subtract trigger and the A register is shown in Figure 3A-6. Two of the four AND blocks are degated with 'shift counter equals zero' and the logic of the remaining two AND blocks causes the subtract trigger to be turned on when there are two ones in bits 30 and 31 of the BX register. This signals the beginning of a group of ones. Once the subtract trigger is on, it will turn off only when bits 30 and 31 both become zero. This signals the end of a group of ones.

The condition of the subtract-trigger-backup latch is delayed one cycle from the subtract trigger. It will thus be on when the multiplier digit is the second one of a group of ones, and remains on for the remainder of the group of ones and for the first of the two zeros signalling the end of a group of ones. It will be off for the second of these zeros, the first one of the group of ones, and for other digits that are not part of a group of ones.



* Shift Counter Equals 0 only

Figure 3A-6. Invert A Register and Subtract Trigger Logic

The conditions where no arithmetic cycle is required (refer to "Principles of Multiply Operation") are:

Multiply digit of 1 in a group of ones, but not the first.

Multiply digit of 0 which is not part of a group of ones.

The first condition is equivalent to BX register bit 31 being a one and the subtract-trigger-backup latch being on; the second is equivalent to the BX register bit 31 being a zero and the subtract-trigger-backup latch being off. This logic (Figure 3A-7) causes the clock to wrap at 250 (CC2), thereby causing the arithmetic operation to be skipped for that cycle.

Arithmetic cycles are performed when the clock is not wrapped at CC2. They occur when:

BX register bit 31 = 1 and subtract-trigger-backup off.

BX register bit 31 = 0 and subtract-trigger-backup on.

The first of these two cases represents a multiplier digit of one, which is either the first one of a group of ones or a one not part of a group of ones. The corresponding states of the subtract trigger are on and off respectively. Thus, the multiplier digit that is the first one of a group of ones causes a subtract operation, and the multiplier digit of one that is not part of a group of ones causes an add operation.

The second case represents a multiplier digit of zero, which is either a single zero in a group of ones or the first zero at the end of a group of ones. The corresponding states of the subtract trigger are on and off respectively. Thus, the multiplier digit which is a single zero within a group of ones causes a subtract operation, and the first zero at the end of a group of ones causes an add operation.

These two cases correspond to the conditions described under "Principles of the Multiply Operation." The rules governing these conditions are summarized in Figure 3A-8.

On each multiply cycle, regardless of the type of cycle, the shift counter is reduced by one very early in the cycle, and on each cycle except the first the

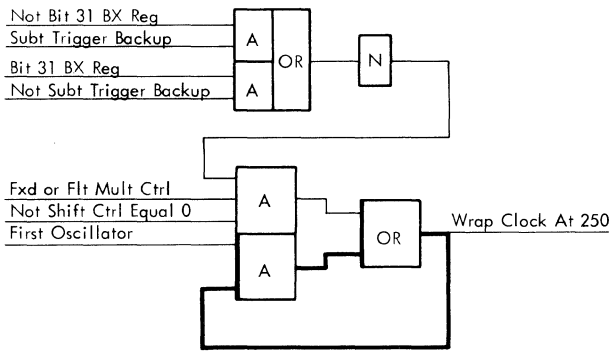


Figure 3A-7. Compute Clock Control for Multiply Operation

Subtract Trigger Backup	BX Register Bit 30	Bit 31	Effective Arithmetic Cycle	Subtract Trigger
Off	0	0	-	Off
Off	1	0	-	Off
Off	0	1	Add	Off
Off	1	1	Subtract	On
On	1	1	-	On
On	0	1	-	On
On	1	0	Subtract	On
On	0	0	Add	Off

Figure 3A-8. Multiply Conditions

B and BX registers are shifted right one position. The shifting is an arithmetic shift with the added facility that any arithmetic overflow from the previous arithmetic cycle can be corrected. The logic for the 'shift-right-one insert' which controls the digit shifted into the bit 00 position of the B register, is shown in Figure 3A-9. A one is inserted on the shift-right operation immediately following an arithmetic cycle (CC1 to CC6) if the result was either negative with no overflow conditions, or positive with overflow conditions. On the shift-right operation immediately following a cycle where there was no arithmetic operation (CC1 to CC2), a one is inserted if the B register is negative (i.e., bit 00 equals 1).

An instructional chart of the multiply cycles is shown in Figure 3A-10. Each cycle is defined by the value of the shift counter at the end of the cycle. As the shift counter is reduced by one extremely early in the cycle (special early CC1, CP1) this also represents the value of the shift counter during the effective part of that cycle.

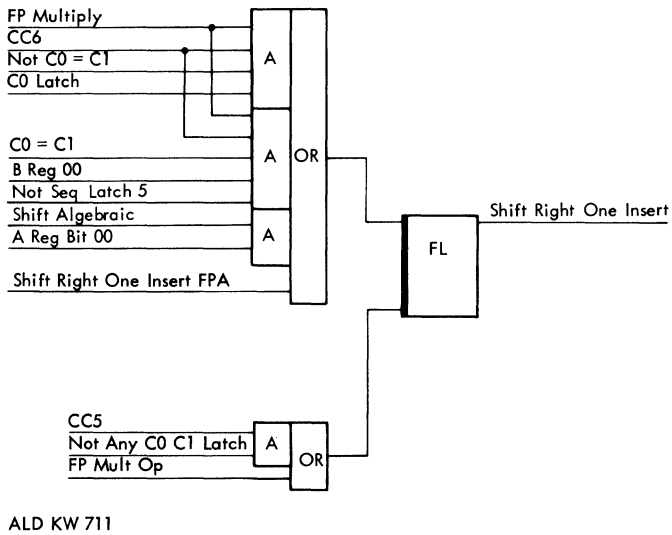


Figure 3A-9. Shift Right One Insert for Multiply Operations

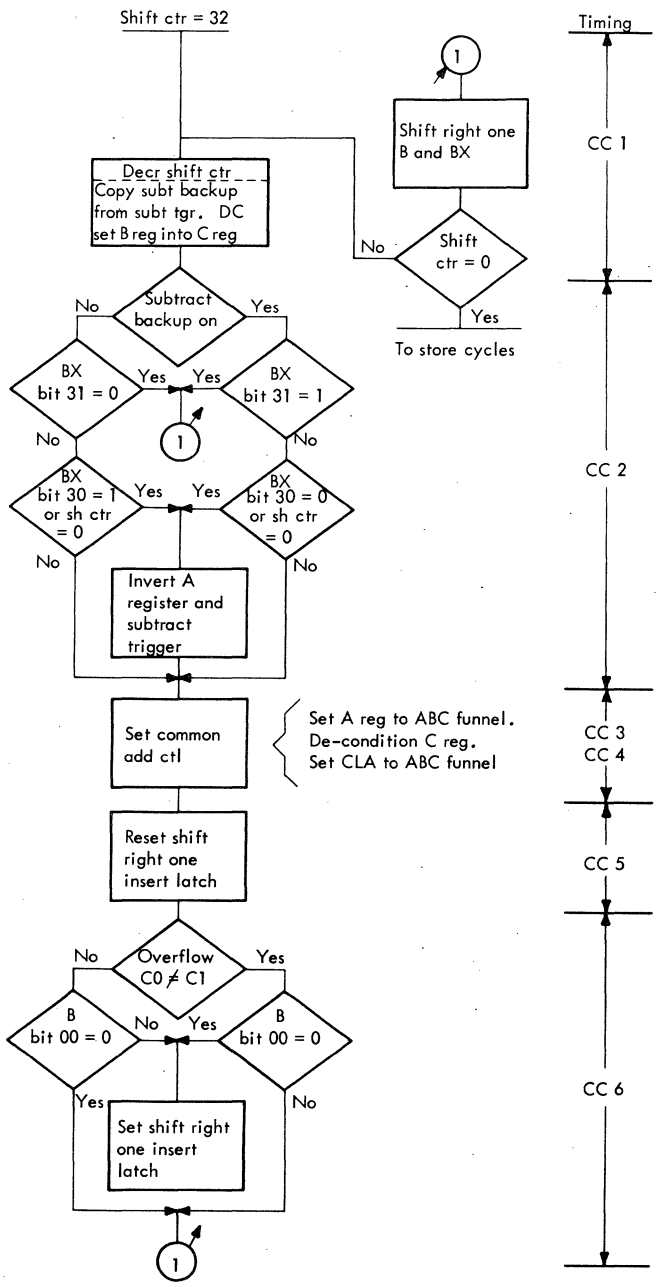


Figure 3A-10. Instructional Multiply Flow Chart

Multiply Cycle, Shift Counter = 0

- The multiplier sign bit becomes the multiplier digit.
- The clock is always allowed to run to CC6.

The cycle on which the shift counter becomes zero is the last cycle, and the sign bit becomes the multiplier digit. The analysis for the 'invert A register and subtract trigger' logic, and that for the signaling of an arithmetic cycle, is performed only on bit 31 of the BX register, as at this time bit 30 is the low-order bit of the product. The effect of BX register bit 30 on the logic for the 'invert A register and subtract trigger' is removed by the addition of the bottom two AND blocks which are conditioned with 'shift count equals zero'. (See Figure 3A-6.) This means that, for this cycle, regardless of bit 30 of the BX register, the A register and subtract trigger are inverted if either the subtract trigger is off and bit 31 equals one, or the subtract trigger is on and bit 31 equals zero. In both cases an arithmetic cycle is performed by bringing up the common-add control.

For the inverse of these conditions it is usual to wrap the clock at CC2. For this last multiply cycle, the clock is allowed to run to CC6 and, if no arithmetic operation is required, the common-add control is not conditioned.

Since the carry latches (C0 and C1) are usually reset by the 'de-condition C register' signal during an arithmetic cycle, special provision is made to reset them on this last cycle if no arithmetic operation is performed. This prevents incorrect conditions being used to set the shift-right-one insert logic on this last cycle. (See Figure 3A-9 for the shift-right-one insert logic.)

Product Store Cycles

- The last shift right of the product is performed.
- The product is stored, using two compute cycles to the even/odd GPR pair defined by the Ra register.

The last shift-right-one is performed to move out the multiplier sign and align the product in the B and BX registers.

The high-order bits of the product are stored in the GPR defined by Ra. The low-order bits are interchanged from the BX to the B register and stored in the GPR defined by Ra plus one.

Figure 3A-11 shows two multiply examples using an eight-bit multiplier. The multiplier digits are still labelled bits 30 and 31 for convenience in relating to the text and the ALD's.

Instruction Objectives, Multiply Halfword

This operation differs only slightly from the multiply instruction previously described. The only differences are:

Multiplies the full-word multiplicand by the half-word multiplier.

Stores the low-order 32 bits of the product in the multiplicand field.

MULTIPLIER 01001011 (+75 dec.)

Shift	Subt Trg	After Shift		CC2		CC3-CC6		Operation	
Count	& Backup	Bit 30	Bit 31	Inv A&S	CC Wrap	S Trg	A Reg	Arith	Ef Mply
7	Off	1	1	X	-	On	Inv	Subt	- 1
6	On	0	1	-	X			None	
5	On	1	0	-	-	On	Inv	Subt	- 4
4	On	0	1	-	X			None	
3	On	0	0	X	-	Off	True	Add	+ 16
2	Off	1	0	-	X			None	
1	Off	0	1	-	-	Off	True	Add	+ 64
0	Off	-	0	-	X			None	

MULTIPLIER 10010110 (-106 dec.)

Shift	Subt Trg	After Shift		CC2		CC3-CC6		Operation	
Count	& Backup	Bit 30	Bit 31	Inv A&S	CC Wrap	S Trg	A Reg	Arith	Ef Mply
7	Off	1	0	-	X			None	
6	Off	1	1	X	-	On	Inv	Subt	- 2
5	On	0	1	-	X			None	
4	On	1	0	-	-	On	Inv	Subt	- 8
3	On	0	1	-	X			None	
2	On	0	0	X	-	Off	True	Add	+ 32
1	Off	1	0	-	X			None	
0	Off	-	1	X	-	On	Inv	Subt	- 128

Figure 3A-11. Multiply Examples with Eight-Bit Multiplier

Execution Sequence, Multiply Halfword

The multiply halfword is executed in the following steps:

1. Operand fetch and initialize cycles.
2. Multiply cycles, shift counter 15 to 0 inclusive.
3. Product store cycle.

The multiply cycles, shift counter 15 to 0 inclusive do not differ in any way from the normal multiply instruction and are not described again. The sequence and timing for the cycles listed in steps 1, 2 and 3 are detailed in FEMD, Figures 6033/34 to 6037/38 inclusive.

Operand Fetch and Initialize Cycles

- The multiplier is fetched from the halfword location defined by the effective address.
- The multiplier is expanded to a full word by sign propagation to the high-order 16 bits.
- The multiplier is set to the BX register.
- Shift counter is set to 16.
- The multiplicand is fetched from the general-purpose register defined by Ra.
- The multiplicand is set to the A register.

Product Store Cycle

- The final shift right is performed to remove the multiplier sign bit from BX register bit 31.
- Four shift-left-four operations are performed to align the low-order bit of the product with B register bit 31.
- The low-order 32 bits of the product are placed in the GPR defined by Ra.

Note that the low-order bit of the product is located in bit 15 of the BX register after the calculation is completed. To align this bit 15 to bit 31 of the B register, four shift-left-four operations are performed.

DIVIDE INSTRUCTION

- The instruction is in either RR or RX format.
- The respective op codes and mnemonics are 1D (DR) and 5D (D).
- The divisor, remainder and quotient are all 32-bit signed integers.
- The dividend is a 64-bit signed integer.
- The quotient sign is determined by the rules of algebra.
- The remainder sign is the same as that of the dividend, except for a zero remainder, when the sign is always positive.
- The condition code is not altered by this instruction.

The divide instruction causes the dividend (first operand) to be divided by the divisor (second operand) and the remainder and quotient developed to be stored in the original dividend field.

The dividend is a 64-bit signed integer and occupies the even/odd pair of general-purpose registers specified by the R1 field of the instruction. A 32-bit signed remainder and a 32-bit signed quotient replace the dividend in the even-numbered and the odd-numbered register respectively. A specification exception occurs if R1 specifies an odd-numbered register.

The divisor is a 32-bit signed integer and it is contained either in the register specified by the R2 field of the RR instruction, or in the full word specified by the effective address of the RX instruction. Normal rules for address and specification exceptions apply to this operand.

The sign of the quotient is determined by the rules of algebra, and the sign of the remainder is the same as that of the dividend. An exception to this rule occurs when either quotient or remainder is zero, in which case the sign is always positive.

When the relative magnitude of the dividend and divisor is such that the quotient cannot be expressed as a 32-bit integer, a fixed-point divide exception occurs and the divide execution is stopped. The divisor and dividend remain unchanged in the original fields. A program interrupt is signalled by this condition. The condition code is not altered by the divide operation.

Principles of Division

- For the division process the dividend is accepted with its original sign.
- The dividend field is reduced towards zero by the value of the divisor.
- A successful reduction occurs when the result of the reduction has the same sign as the original dividend.
- An overdraw occurs when the result of the reduction is of opposite sign to that of the original dividend.
- A correction cycle resulting from an overdraw is combined with the subsequent reduction cycle by shifting before correcting.
- The same rules apply to this combined cycle for successful reduction and overdraw as are stated for the other cycles.
- The quotient bit is developed for each successful reduction.
- The dividend is placed in the B and BX registers.
- The divisor is placed in the A register.
- The quotient is developed in the BX register.
- The remainder is left in the B register.

The dividend is placed in the B and BX registers in its original form, regardless of sign. The divisor is placed in the A register, and the arithmetic operations are performed between the A and B registers.

After each arithmetic operation the B and BX registers are shifted left one position and any quotient bit developed is inserted in the low-order positions of the BX register.

Thus, the 64-bit dividend is reduced to a 32-bit remainder as it travels leftward in the B and BX registers and the quotient correspondingly increases until it fills the BX register.

The dividend field in the B register is reduced towards zero by the divisor field. A successful reduction occurs if the result of this arithmetic operation has the same sign as the original dividend.

This is illustrated in Figure 3A-12 by the representation (a) for both positive and negative dividends.

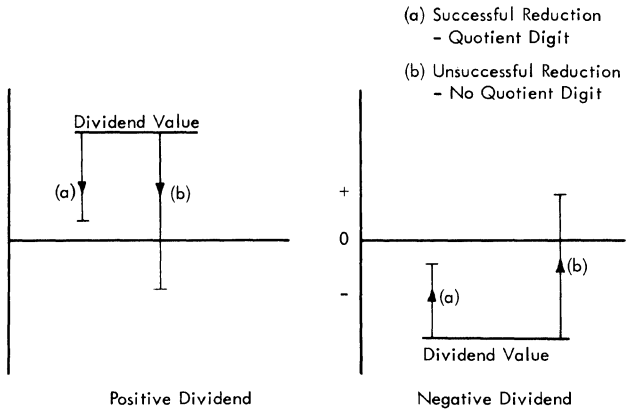


Figure 3A-12. Illustrations of Normal Reduction Cycle

An overdraw occurs if the result of the arithmetic operation has the opposite sign to the original dividend. This is shown as representation (b).

After an overdraw has occurred the dividend field is corrected by adding back the divisor and then executing a trial reduction. The reduction is made by shifting and then repeating the arithmetic process.

The shift left of the dividend field can be regarded as a shift right (or halving) or the divisor, relative to the dividend field (Figure 3A-13). This means that if normal practice were followed, the divisor would be restored to the dividend field and then it would be reduced by half its value. This is shown as representation (x).

If the shift (or halving) takes place before the correction cycle, effectively only half the divisor is added back. This is equivalent to adding back the whole divisor and then reducing by half the divisor value. This cycle is known as the combined correction and reduction cycle. This is shown as representation (y).

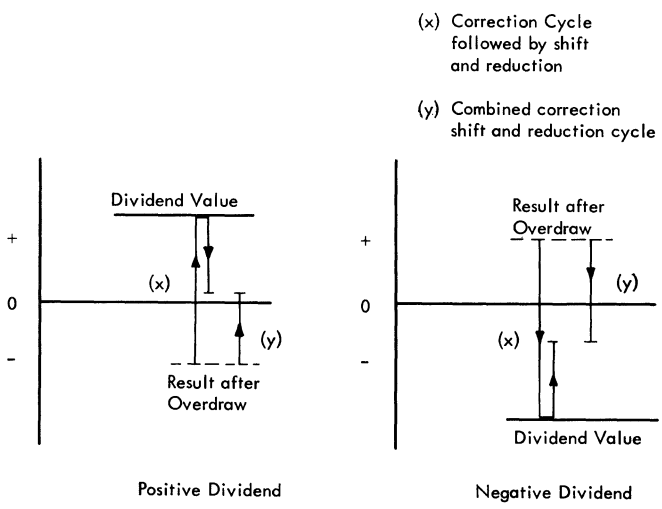


Figure 3A-13. Illustration of Combined Correction and Reduction Cycles

For this combined cycle, a successful reduction occurs if the result sign is the same sign as the original dividend, and an overdraw occurs if the result has the opposite sign to the original dividend.

Where it is required to change the direction of the arithmetic operation (after an overdraw on a reduction cycle or a successful reduction after a combined correction and reduction cycle), the divisor (in the A register) and the subtract trigger are inverted.

A quotient bit is developed for each successful reduction, and on an overdraw no quotient bit is inserted. The quotient is always developed in true form and complemented at the end of the divide operation if the divisor and dividend signs are unlike.

Zero is always positive; thus, a negative dividend when the result of a reduction is zero in the B register, is signalled as an unsuccessful reduction (overdraw), and no quotient bit is developed. However, if the whole of the dividend field (the B register and the dividend part of the BX register) is zero, the reduction is in fact a perfect reduction cycle.

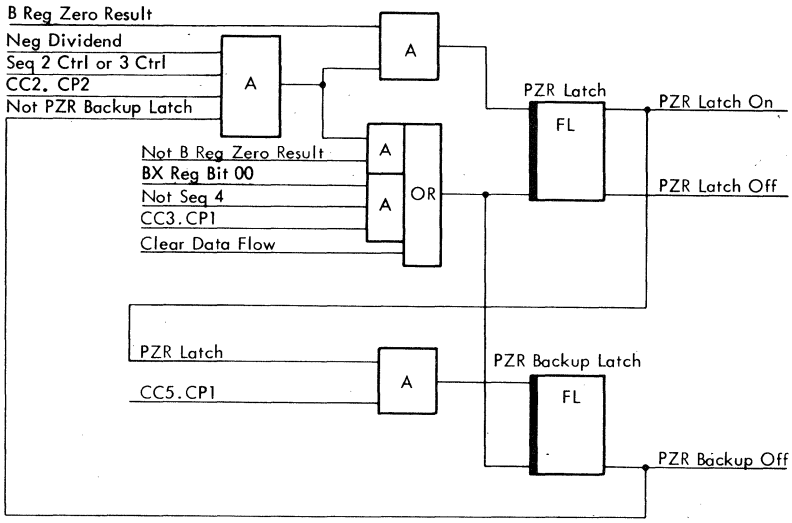
For the condition where the dividend is negative and a reduction cycle results in zero in the B register, the Possible Zero Remainder (PZR) latch is set. This latch is reset if, during the subsequent divide cycles, the BX register bit 00 is a one (Figure 3A-14).

Note that, for the condition where a perfect reduction occurs, no quotient bit is developed at that time and the next cycle will be a combined correction and reduction cycle. On this and all subsequent cycles the result in the B register will be negative, and a quotient bit will always result.

If the PZR latch is on at the end of the divide operation, the remainder is set to zero and a one is added to the quotient. This one is propagated to the position corresponding to the cycle on which the perfect reduction occurred, due to the series of ones developed in the quotient after that cycle.

Also, the divide exception, and the possibility of developing a quotient which represents the maximum negative number, provide further complications.

The basic principle used is that the quotient is developed in true form. If a quotient bit is developed on either of the first two of the 32 divide cycles, the quotient cannot be expressed in a 32-bit signed integer, and a divide exception occurs. The exception to this case is the condition when a quotient bit is developed on the second cycle, the rest of the quotient is zeros, and the quotient sign is negative. This condition represents the maximum negative number, and the development of a quotient bit on the second cycle is valid. For all other conditions, the development of a quotient bit on the second cycle is a divide exception, but the divide must proceed to



ALD KW 571

Figure 3A-14. Possible Zero Remainder Latch

determine whether the quotient is the maximum negative number. To allow this condition to be detected, the 'possible maximum negative number' latch is added.

The use of this latch, in detecting a quotient representing the maximum negative number, and the application of the preceding rules in the Model 44, are shown in the flow chart in FEMD, Figures 6041/42 and 6043/44 (basic) and 6241/42 and 6243/44 (accelerator).

Instruction Execution

The divide instruction is described in five major steps:

1. Division set-up cycles.
2. Division compute cycles.
3. Remainder correction cycle.
4. Quotient correction cycle.
5. Quotient and remainder store cycle.

Step 1 consists of three storage cycles on which the operands are fetched and positioned in the appropriate registers and the divide controls are initialized.

Step 2 usually consists of 32 divide cycles during which the quotient is developed.

Step 3 is a compute cycle that is always taken, but it is used effectively only if an overdraw has occurred on the last cycle and the PZR latch is off. For this condition the remainder needs a full correction to restore it to the correct magnitude and sign.

Step 4 consists of a cycle on which the last quotient bit is inserted, corrected and/or complemented where necessary.

Step 5 consists of a cycle on which the remainder is placed in the even-numbered GPR and the quotient placed in the odd-numbered GPR of the original dividend field.

The sequence and timing of cycles listed in steps 1, 3, 4 and 5 are shown in the flow charts and timing charts in FEMD, Figures 6039/40 to 6043/44 (basic) and 6239/40 to 6243/44 (accelerator).

Division Set-Up Cycles

- The divisor, or second operand, is fetched and set in the A register.
- The 'divisor sign' latch is set from the A register sign bit 00.
- The low-order 32 bits of the dividend are fetched from the odd-numbered GPR defined by Ra, and set into the BX register.
- The shift counter is set to a value of 33.
- The high-order 32 bits of the dividend are fetched from the even-numbered GPR defined by Ra, and set into the B register.
- The 'dividend sign' latch is set from the B register sign bit 00.
- A signal indicating the expected sign of the quotient is developed from the 'dividend and divisor sign' latches.

The divisor is fetched from the second operand location and set into the A register. The divisor sign is analyzed and set into the negative-divisor latch.

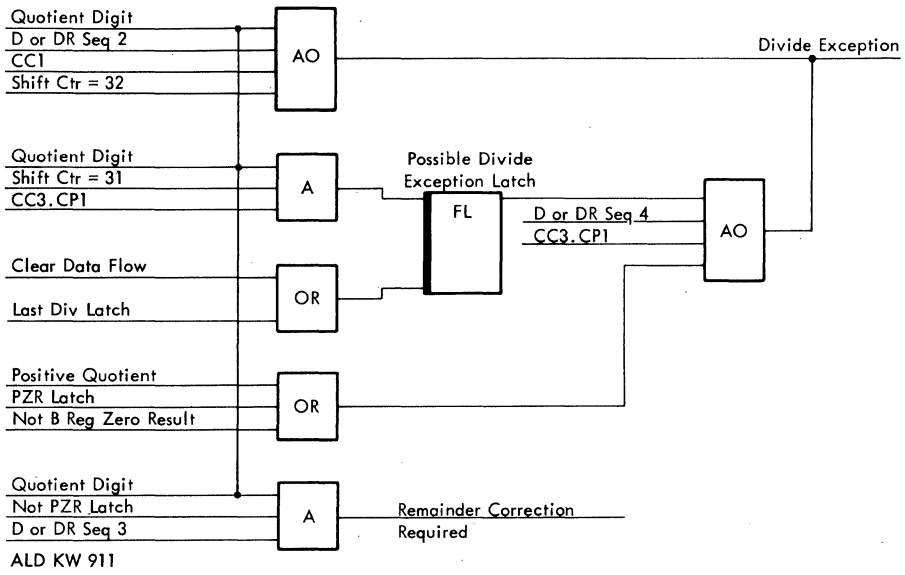


Figure 3A-15. Divide Exception and Remainder Correction

The dividend low-order fetch cycle fetches the 32 bits from the dividend register specified by Ra plus 1, and sets them in the B register. A 32-bit interchange then occurs between the B and BX registers, transferring these bits to the BX register. Sequence latch 5 is active on this cycle. During this cycle, the shift counter is reset and then set to a value of 33.

The dividend high-order fetch cycle fetches the 32 bits from the register specified by Ra, and sets them in the B register. On this cycle, sequence latch 1 is active for the basic machine, and sequence latch 2 is active for the accelerator.

During this cycle, the dividend sign is analyzed and set in the negative-dividend latch. The negative-divisor, and negative-dividend, sign latches are then EXOR'ed to form the quotient sign control signal.

Division Compute Cycles

- These are 32 compute cycles, the clock always running from CC1 to CC6.
- The B and BX registers are shift-left-one on all cycles except the first.
- The shift counter is reduced by one on all cycles, in parallel with the arithmetic operation.
- Any quotient bit developed is inserted in BX register bit 30 on the following cycle, after the shift left has taken place.
- A quotient bit is developed if the sign of the result in the B register is the same as the sign of the original dividend.

Division cycles are identified by the value of the shift counter at the beginning of the cycle.

At the start of the first division cycle the shift counter value is 33. This first cycle (33) is a trial reduction cycle to test for a fixed-point divide exception. During this first effective division cycle (33) the A register and subtract trigger are inverted if the quotient sign control indicates a positive quotient sign. The A register contents are then added to the B register and the result tested to determine if a quotient bit is developed. The shift counter is also reduced by one during the cycle and the value is 32 at the end of the cycle.

If a quotient bit is developed on this first cycle, then the quotient cannot be expressed as a 32-bit signed integer, and a fixed-point divide exception is signalled, ending the divide operation (Figure 3A-15).

Sequence latch 2 is turned on during this first cycle and is used for all division cycles until the shift counter becomes zero.

The second divide cycle (shift counter = 32) performs a shift followed by a combined correction and reduction cycle, storing the result for subsequent testing for a possible fixed-point divide exception condition.

The B and BX registers are shifted left one position at the start of the cycle. The shift counter is reduced by one during the cycle and the value is 31 at the end of the cycle.

The 'invert A register and subtract trigger' logic (Figure 3A-16) is used on the second divide cycle and an invert A should always result, because the alternative condition would have caused a divide exception on the previous cycle.

If a quotient bit is developed on the second divide cycle, it is not inserted in the quotient field but

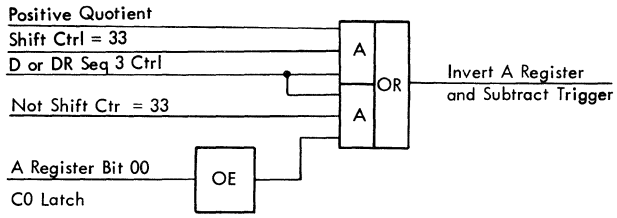


Figure 3A-16. Invert A Register and Subtract Trigger for Divide Instruction

stored in the 'possible divide exception' latch. This latch is tested at the end of the divide operation because it represents a fixed-point divide exception in all cases except one, that is, when the quotient is the maximum negative number. (See Figure 3A-14.)

A series of division cycles (from cycle 31 to cycle 1) progressively reduces the dividend field by the divisor and develops a quotient in the BX register and the remainder in the B register.

On each cycle the B and BX registers are shifted left one position early in the cycle, the shift counter is reduced by one, and the A register and subtract trigger are inverted where necessary.

The conditions under which the A register and subtract trigger are inverted are:

1. An overdraw on a full reduction cycle.
2. A successful reduction on a combined correction and reduction cycle.

For positive dividends:

Condition 1 is given by:
A register negative (bit 00 = 1)
and no carry-out (C0 = 0)

Condition 2 is given by:
A register positive (bit 00 = 0)
and carry-out (C0 = 1)

For negative dividends:

Condition 1 is given by:
A register positive (bit 00 = 0)
and carry-out (C0 = 1)

Condition 2 is given by:
A register negative (bit 00 = 1)
and no carry-out (C0 = 0)

These four cases can be formed by the EXOR of the A register sign bit 00 and the C0 latch.

On the type of arithmetic cycles being performed during the divide operation, a carry-out of the sign position (C0) represents a positive result, and a no carry-out of this position represents a negative result. The AND condition of C0 with the dividend sign produces the quotient bit (Figure 3A-17).

The CLA bit 0 carry is not available until late in the cycle and so the quotient digit is inserted during the following cycle into bit 30 of the BX register, after the shift left has been performed.

On the divide cycle during which the shift counter becomes zero, sequence 3 control is conditioned and this signals the end of the effective division cycles.

The PZR latch is set and reset on each division cycle from the PZR backup latch. (See Figure 3A-14.) The PZR latch is successively set in each cycle if all previous B register bits were zero and the sign of the original dividend was negative.

Remainder Correction Cycle

- This cycle is always taken, even if remainder correction is not required.
- If a quotient bit was developed on the division cycle with 'shift count equals zero', a quotient bit is inserted in BX register bit 31.
- If a bit was not developed on the division cycle with 'shift count equals zero', a remainder correction is required.
- No shift left takes place on this cycle.
- For a remainder correction, the normal 'A register and subtract trigger' logic is used.

The quotient, developed on the division cycle during which the shift counter becomes zero, is stored on the remainder correction cycle. As there is no shift left on this cycle the quotient bit is stored in bit 31 of the BX register.

If this quotient bit is zero, an overdraw has occurred on the previous cycle and a full correction must occur. (Refer to Figure 3A-14 for the remain-

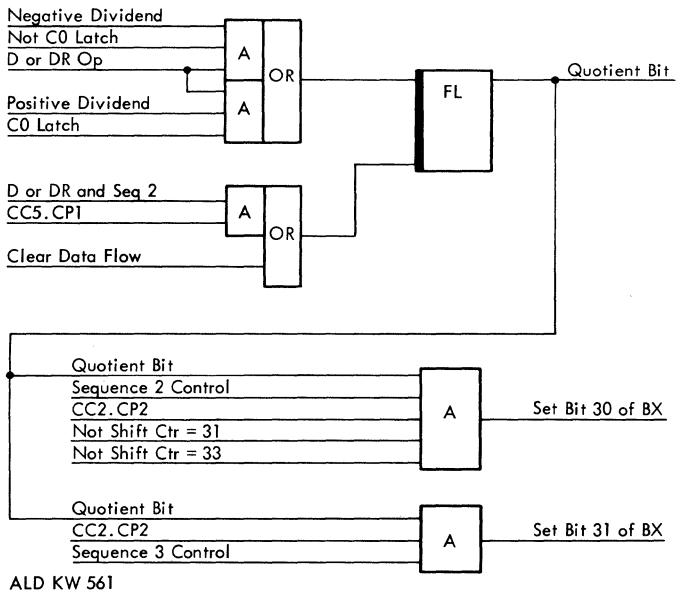


Figure 3A-17. Quotient Bit Logic

der correction logic.) The 'invert A register and subtract trigger' logic is still active on this current (remainder correction) cycle and the A register contents are added to the B register if a remainder correction is required. As there is no shift left on this cycle, a full correction occurs.

The PZR latch is tested and, if found to be set, the remainder should be zero; the B register is thus reset to zero on this cycle.

The controlling sequence for this cycle is sequence 3.

Quotient Correction Cycle

- Divide exception is signalled if a quotient bit was developed previously on the second divide cycle and the quotient does not represent the maximum negative number.
- If the quotient represents the maximum negative number, the zero quotient is gated to SDR with its sign bit inverted (100 - - - 00).
- For negative quotients the developed quotient is complemented.
- For cases where the PZR latch is on, a one is added to the quotient regardless of its sign.

This cycle is performed under the control of sequence 4 and is the cycle used for correcting the quotient where necessary.

The quotient is placed in the B register by a B to BX register interchange and is tested for a zero quotient. If the quotient is zero, the PZR latch is off, the quotient sign is negative and the possible-divide-exception latch is on, the quotient represents the maximum negative number. Under these conditions the B register is reset and the sign bit is gated in inverted form to the output bus (1000 - - - 00).

If the possible-divide-exception latch is on and either the quotient sign is positive, or the B register is not zero, or the PZR latch is on, the capacity of the 32-bit register has been exceeded and a fixed-point divide exception is signalled. (See Figure 3A-15.) This inhibits the store operation, terminates the divide instruction, and requests a program interrupt.

If the quotient is negative and the possible-divide-exception latch is off, the quotient has to be converted to two's complement form prior to storing.

If the PZR latch is on, a one has to be added to the quotient, regardless of its sign. For negative quotients this is included in the complementing process.

The two's complementing process is achieved by resetting the A register, gating one's to the B register and then gating the CLA output to the B register. Normally the subtract trigger is set on to provide a CLA carry-in.

If the PZR latch is on, the magnitude of the quotient is increased by one, by not setting the subtract trigger for this complementing operation.

If the PZR latch is on and the quotient is positive, the one is added by resetting the A register, setting the subtract trigger on, and adding the A register to the B register. The subtract trigger carry-in to the CLA causes a one to be added to the B register.

Quotient and Remainder Store Cycles

- BASIC only: The quotient is stored on an R1 cycle into the odd-numbered GPR specified by Ra plus 1.
- ACCELERATOR only: The quotient is gated into the odd-numbered GPR defined by Ra plus 1.
- The remainder is interchanged and placed in the B register.
- BASIC only: On an R1 cycle, the remainder is stored into the even-numbered GPR specified by Ra.
- ACCELERATOR only: The remainder is gated into the even-numbered GPR specified by Ra.

The remainder is placed in the B register by a B to BX register interchange, and from there it is stored in the GPR specified by Ra.

At the end of the remainder store cycle, the 'end execute' is signalled and the divide instruction is complete.

Three examples of print-outs from the simulate program for single cycle divide are shown in Figures 3A-18, 19 and 20. These figures show the register contents and the state of the controlling latches and triggers at each single-cycle point. They should be used in conjunction with the divide flow charts in FEMD, Figures 6038 to 6043 (basic) and Figures 6239 to 6243 (accelerator).

```
ORIGINAL DIVIDEND IS
 11111111 11111111 11111111 11111101  01010101 01010101 01010101 01010110

DIVIDEND SIGN LATCH ON
DIVISOR  SIGN LATCH ON

A REG TRUE     = 11111111 11111111 11111111 11111000
A REG INVERTED = 00000000 00000000 00000000 00000111

   SHIFT     POS QUOT A                    B                          BX       S
  COUNTER    ZERO BIT REG                  REG                        REG      E
DEC  BINARY  REM LTCH      ---------------BINARY--------------        HEX      Q
                                                                              5
 33  100001  OFF OFF TRUE 00000000 00000000 00000000 00000000   55 55 55 56 1
 32  100000  OFF OFF INVR 00000000 00000000 00000000 00000101   55 55 55 56 2
 31  011111  OFF OFF TRUE 00000000 00000000 00000000 00000010   AA AA AA AC 2
 30  011110  OFF ON  TRUE 11111111 11111111 11111111 11111101   55 55 55 58 2
 29  011101  OFF OFF INVR 00000000 00000000 00000000 00000010   AA AA AA B2 2
 28  011100  OFF ON  TRUE 11111111 11111111 11111111 11111101   55 55 55 64 2
 27  011011  OFF OFF INVR 00000000 00000000 00000000 00000010   AA AA AA CA 2
 26  011010  OFF ON  TRUE 11111111 11111111 11111111 11111101   55 55 55 94 2
 25  011001  OFF OFF INVR 00000000 00000000 00000000 00000010   AA AA AB 2A 2
 24  011000  OFF ON  TRUE 11111111 11111111 11111111 11111101   55 55 56 54 2
 23  010111  OFF OFF INVR 00000000 00000000 00000000 00000010   AA AA AC AA 2
 22  010110  OFF ON  TRUE 11111111 11111111 11111111 11111101   55 55 59 54 2
 21  010101  OFF OFF INVR 00000000 00000000 00000000 00000010   AA AA B2 AA 2
 20  010100  OFF ON  TRUE 11111111 11111111 11111111 11111101   55 55 65 54 2
 19  010011  OFF OFF INVR 00000000 00000000 00000000 00000010   AA AA CA AA 2
 18  010010  OFF ON  TRUE 11111111 11111111 11111111 11111101   55 55 95 54 2
 17  010001  OFF OFF INVR 00000000 00000000 00000000 00000010   AA AB 2A AA 2
 16  010000  OFF ON  TRUE 11111111 11111111 11111111 11111101   55 56 55 54 2
 15  001111  OFF OFF INVR 00000000 00000000 00000000 00000010   AA AC AA AA 2
 14  001110  OFF ON  TRUE 11111111 11111111 11111111 11111101   55 59 55 54 2
 13  001101  OFF OFF INVR 00000000 00000000 00000000 00000010   AA B2 AA AA 2
 12  001100  OFF ON  TRUE 11111111 11111111 11111111 11111101   55 65 55 54 2
 11  001011  OFF OFF INVR 00000000 00000000 00000000 00000010   AA CA AA AA 2
 10  001010  OFF ON  TRUE 11111111 11111111 11111111 11111101   55 95 55 54 2
  9  001001  OFF OFF INVR 00000000 00000000 00000000 00000010   AB 2A AA AA 2
  8  001000  OFF ON  TRUE 11111111 11111111 11111111 11111101   56 55 55 54 2
  7  000111  OFF OFF INVR 00000000 00000000 00000000 00000010   AC AA AA AA 2
  6  000110  OFF ON  TRUE 11111111 11111111 11111111 11111101   59 55 55 54 2
  5  000101  OFF OFF INVR 00000000 00000000 00000000 00000010   B2 AA AA AA 2
  4  000100  OFF ON  TRUE 11111111 11111111 11111111 11111101   65 55 55 54 2
  3  000011  OFF OFF INVR 00000000 00000000 00000000 00000010   CA AA AA AA 2
  2  000010  OFF ON  TRUE 11111111 11111111 11111111 11111101   95 55 55 54 2
  1  000001  OFF OFF INVR 00000000 00000000 00000000 00000011   2A AA AA AA 2
  0  000000  OFF ON  TRUE 11111111 11111111 11111111 11111110   55 55 55 54 2
  0  000000  OFF     INVR 11111111 11111111 11111111 11111110   55 55 55 55 3
  0  000000  OFF     RSET 01010101 01010101 01010101 01010101   FF FF FF FE 4
  0  000000  OFF     RSET 11111111 11111111 11111111 11111110   55 55 55 55 4
  0  000000  OFF     RSET 01010101 01010101 01010101 01010101   FF FF FF FE 4
RESULT IN GR0 GR1 SHOULD BE
 11111111 11111111 11111111 11111110  01010101 01010101 01010101 01010101
RESULT OF DIVIDE IS
 11111111 11111111 11111111 11111110  01010101 01010101 01010101 01010101
NOW MACHINE LOOPS IN DIVIDE OPERATION.
RESTART BY PSW RESTART.  LOOP ADDRESS=HEX D00
```

Figure 3A-18. Example of Divide with Two Negative Operands

```
ORIGINAL DIVIDEND IS
 11111111 11111111 11111111 11111100  00000000 00000000 00000000 00000000

DIVIDEND SIGN LATCH ON
DIVISOR  SIGN LATCH ON

A REG TRUE     = 11111111 11111111 11111111 11110000
A REG INVERTED = 00000000 00000000 00000000 00001111

   SHIFT     POS QUOT A                  B                          BX       S
  COUNTER    ZERO BIT REG                REG                        REG      E
DEC  BINARY  REM LTCH     ---------------BINARY--------------       HEX      Q
                                                                              5
 33  100001  OFF OFF TRUE 00000000 00000000 00000000 00000000   00 00 00 00 1
 32  100000  OFF OFF INVR 00000000 00000000 00000000 00001100   00 00 00 00 2
 31  011111  OFF OFF TRUE 00000000 00000000 00000000 00001000   00 00 00 00 2
 30  011110  OFF OFF TRUE 00000000 00000000 00000000 00000000   00 00 00 00 2
 29  011101  ON  ON  TRUE 11111111 11111111 11111111 11110000   00 00 00 00 2
 28  011100  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 00 02 2
 27  011011  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 00 06 2
 26  011010  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 00 0E 2
 25  011001  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 00 1E 2
 24  011000  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 00 3E 2
 23  010111  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 00 7E 2
 22  010110  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 00 FE 2
 21  010101  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 01 FE 2
 20  010100  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 03 FE 2
 19  010011  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 07 FE 2
 18  010010  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 0F FE 2
 17  010001  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 1F FE 2
 16  010000  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 3F FE 2
 15  001111  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 7F FE 2
 14  001110  ON  ON  INVR 11111111 11111111 11111111 11110000   00 00 FF FE 2
 13  001101  ON  ON  INVR 11111111 11111111 11111111 11110000   00 01 FF FE 2
 12  001100  ON  ON  INVR 11111111 11111111 11111111 11110000   00 03 FF FE 2
 11  001011  ON  ON  INVR 11111111 11111111 11111111 11110000   00 07 FF FE 2
 10  001010  ON  ON  INVR 11111111 11111111 11111111 11110000   00 0F FF FE 2
  9  001001  ON  ON  INVR 11111111 11111111 11111111 11110000   00 1F FF FE 2
  8  001000  ON  ON  INVR 11111111 11111111 11111111 11110000   00 3F FF FE 2
  7  000111  ON  ON  INVR 11111111 11111111 11111111 11110000   00 7F FF FE 2
  6  000110  ON  ON  INVR 11111111 11111111 11111111 11110000   00 FF FF FE 2
  5  000101  ON  ON  INVR 11111111-11111111 11111111 11110000   01 FF FF FE 2
  4  000100  ON  ON  INVR 11111111 11111111 11111111 11110000   03 FF FF FE 2
  3  000011  ON  ON  INVR 11111111 11111111 11111111 11110000   07 FF FF FE 2
  2  000010  ON  ON  INVR 11111111 11111111 11111111 11110000   0F FF FF FE 2
  1  000001  ON  ON  INVR 11111111 11111111 11111111 11110000   1F FF FF FE 2
  0  000000  ON  ON  INVR 11111111 11111111 11111111 11110000   3F FF FF FE 2
REMAINDER CORRECTION REQUIRED
DC RESET B REG
  0  000000  ON      INVR 00000000 00000000 00000000 00000000   3F FF FF FF 3
ADD ONE ONLY
  0  000000  ON      RSET 01000000 00000000 00000000 00000000   00 00 00 00 4
  0  000000  ON      RSET 00000000 00000000 00000000 00000000   40 00 00 00 4
  0  000000  ON      RSET 01000000 00000000 00000000 00000000   00 00 00 00 4
RESULT IN GR0 GR1 SHOULD BE
 00000000 00000000 00000000 00000000  01000000 00000000 00000000 00000000
RESULT OF DIVIDE IS
 00000000 00000000 00000000 00000000  01000000 00000000 00000000 00000000
```

Figure 3A-19. Example of Divide with Remainder Correction Required

```
ORIGINAL DIVIDEND IS
 11111111 11111111 11111111 11111100  00000000 00000000 00000000 00000100

DIVIDEND SIGN LATCH ON
DIVISOR  SIGN LATCH ON

A REG TRUE     = 11111111 11111111 11111111 11111000
A REG INVERTED = 00000000 00000000 00000000 00000111

   SHIFT     POS QUOT A                    B                          BX       S
  COUNTER    ZERO BIT REG                  REG                        REG      E
DEC  BINARY REM LTCH     ----------------BINARY----------------       HEX      Q
                                                                               5
 33  100001 OFF OFF TRUE 00000000 00000000 00000000 00000000   00 00 00 04 1
 32  100000 OFF OFF INVR 00000000 00000000 00000000 00000100   00 00 00 04 2
 31  011111 OFF OFF TRUE 00000000 00000000 00000000 00000000   00 00 00 08 2
 30  011110 ON  ON  TRUE 11111111 11111111 11111111 11111000   00 00 00 10 2
 29  011101 ON  ON  INVR 11111111 11111111 11111111 11111000   00 00 00 22 2
 28  011100 ON  ON  INVR 11111111 11111111 11111111 11111000   00 00 00 46 2
 27  011011 ON  ON  INVR 11111111 11111111 11111111 11111000   00 00 00 8E 2
 26  011010 ON  ON  INVR 11111111 11111111 11111111 11111000   00 00 01 1E 2
 25  011001 ON  ON  INVR 11111111 11111111 11111111 11111000   00 00 02 3E 2
 24  011000 ON  ON  INVR 11111111 11111111 11111111 11111000   00 00 04 7E 2
 23  010111 ON  ON  INVR 11111111 11111111 11111111 11111000   00 00 08 FE 2
 22  010110 ON  ON  INVR 11111111 11111111 11111111 11111000   00 00 11 FE 2
 21  010101 ON  ON  INVR 11111111 11111111 11111111 11111000   00 00 23 FE 2
 20  010100 ON  ON  INVR 11111111 11111111 11111111 11111000   00 00 47 FE 2
 19  010011 ON  ON  INVR 11111111 11111111 11111111 11111000   00 00 8F FE 2
 18  010010 ON  ON  INVR 11111111 11111111 11111111 11111000   00 01 1F FE 2
 17  010001 ON  ON  INVR 11111111 11111111 11111111 11111000   00 02 3F FE 2
 16  010000 ON  ON  INVR 11111111 11111111 11111111 11111000   00 04 7F FE 2
 15  001111 ON  ON  INVR 11111111 11111111 11111111 11111000   00 08 FF FE 2
 14  001110 ON  ON  INVR 11111111 11111111 11111111 11111000   00 11 FF FE 2
 13  001101 ON  ON  INVR 11111111 11111111 11111111 11111000   00 23 FF FE 2
 12  001100 ON  ON  INVR 11111111 11111111 11111111 11111000   00 47 FF FE 2
 11  001011 ON  ON  INVR 11111111 11111111 11111111 11111000   00 8F FF FE 2
 10  001010 ON  ON  INVR 11111111 11111111 11111111 11111000   01 1F FF FE 2
  9  001001 ON  ON  INVR 11111111 11111111 11111111 11111000   02 3F FF FE 2
  8  001000 ON  ON  INVR 11111111 11111111 11111111 11111000   04 7F FF FE 2
  7  000111 ON  ON  INVR 11111111 11111111 11111111 11111000   08 FF FF FE 2
  6  000110 ON  ON  INVR 11111111 11111111 11111111 11111000   11 FF FF FE 2
  5  000101 ON  ON  INVR 11111111 11111111 11111111 11111000   23 FF FF FE 2
  4  000100 ON  ON  INVR 11111111 11111111 11111111 11111000   47 FF FF FE 2
  3  000011 OFF ON  INVR 11111111 11111111 11111111 11111000   8F FF FF FE 2
  2  000010 OFF ON  INVR 11111111 11111111 11111111 11111001   1F FF FF FE 2
  1  000001 OFF ON  INVR 11111111 11111111 11111111 11111010   3F FF FF FE 2
  0  000000 OFF ON  INVR 11111111 11111111 11111111 11111100   7F FF FF FE 2
  0  000000 OFF     INVR 11111111 11111111 11111111 11111100   7F FF FF FF 3
  0  000000 OFF     RSET 01111111 11111111 11111111 11111111   FF FF FF FC 4
  0  000000 OFF     RSET 11111111 11111111 11111111 11111100   7F FF FF FF 4
  0  000000 OFF     RSET 01111111 11111111 11111111 11111111   FF FF FF FC 4
RESULT IN GR0 GR1 SHOULD BE
 11111111 11111111 11111111 11111100  01111111 11111111 11111111 11111111
RESULT OF DIVIDE IS
 11111111 11111111 11111111 11111100  01111111 11111111 11111111 11111111
```

Figure 3A-20. Example of Divide with Remainder Correction and PZR On

OR, EXCLUSIVE OR AND AND INSTRUCTIONS

The following instructions differ for the basic and accelerator in that GPR's are gated direct via the HW funnel to the A, B and C registers.

OR Instructions

- The instruction is in either RR, RX or SI format.
- The respective op codes and mnemonics are 16 (OR), 56 (O) and 96 (OI).
- The first and second operands are combined, bit by bit, in a logical OR function and the result stored in the first operand location.
- The A register performs the logical OR function.
- The condition code is set as follows:
  00 : Result is zero
  01 : Result is non-zero
  10 : Not set
  11 : Not set

In the logical OR instruction, the corresponding bits of each of the first and second operands are combined in a logical OR function. This OR function is summarized as follows:

OR		First Operand Bit	
		0	1
Second Operand Bit	0	0	1
	1	1	1

RR and RX Format

BASIC only: The machine takes an R2 or EA cycle, depending on the format, and reads out the second operand to the SDR. From here it is gated to the A register. The first operand is then read out to the SDR and is also gated to the A register. The A register performs the OR function shown in the above table. That is, if either of the bits of input data is a one, the latch turns on.

The result in the A register is gated to the B register via the ABC funnel. The result is then gated to the SDR from where it is stored into the first-operand location on the write section of the R1 cycle.

The result in the B register is analyzed and if the result is zero the condition code is set to 00. If the result is non-zero the condition code is set to 01.

The sequence and timing of this instruction are shown in FEMD, Figure 6046.

ACCELERATOR only: For RX format, the machine takes an EA cycle and reads out the second operand to the SDR. From here it is gated to the A register. The first operand is then set into the A register. Each position of the A register is implemented in latches, and the register performs the OR function as shown in the example in the previous section "OR Instruction." That is, if either of the bits of input data is a one, the latch turns on.

The result in the A register is gated to the B register via the ABC funnel. From here it is gated to the GPR defined by Ra.

The result in the B register is analyzed and, if the result is zero, the condition code is set to 00. If the result is non-zero, the condition code is set to 01.

For RR format, the second operand is gated out of the GPR defined by Rb, during the I-cycle, and set to the A, B and C registers. The compute clock is then restarted under sequence-2 control. The B register is reset and the first operand is gated from the GPR defined by Ra to the A and C registers.

As the A register is gated to the B register, 'end execute' is signalled.

The sequence and timing of this instruction are shown in FEMD, Figure 6249.

SI Format

The immediate byte, operand 2, in registers Ra and Rb is gated to the HW funnel. The effective address of the first operand in the B register transfers to the SAR, and an EA storage cycle is initiated causing the first operand to enter the SDR. For the OR operation the A register is conditioned and the B register deconditioned so that the resultant OR function is developed in the A register.

Decoding of the last two bits of the storage address, determines the correct byte address for gating the one-byte operand 2 into the correct position of the ABC funnel. Thus, the second operand enters the A register. Similar byte positioning or byte control occurs to gate the required byte of operand 1 from the SDR to the ABC funnel. The first-operand byte enters the A register where it is OR'ed with the second-operand byte.

The B register is conditioned and the A register contents are transferred via the ABC funnel to reg-

ister B. Byte control is again employed to reset the appropriate byte of the SDR which still contains the first operand. The resultant OR byte from the B register is gated to the SDR under byte control, and the complete word is stored in the first operand location.

Analysis of the result in the B register causes the appropriate condition code setting, after which, the 'end execute' is performed. The sequence and timing of the OR instruction are shown in FEMD, Figure 6089.

Exclusive OR Instruction

- The instruction is in either RR, RX or SI format.
- The respective op codes and mnemonics are 17 (XR), 57 (X) and 97 (XI).
- The first and second operands are combined, bit by bit, in an Exclusive OR function and the result stored in the first-operand location.
- The B register performs the Exclusive OR function.
- The condition code is set as follows:
  00 : Result is zero.
  01 : Result is non-zero.
  10 : Not set.
  11 : Not set.

In the logical Exclusive OR instruction, the corresponding bits of each of the first and second operands are combined in a logical EXOR function. This EXOR function is summarized as follows:

EXOR		First Operand Bit	
		0	1
Second Operand Bit	0	0	1
	1	1	0

RR and RX Format

BASIC only: The machine takes an R2 or EA cycle, depending on the format, and reads out the second operand to the SDR. From here it is gated to the A and B registers. The first operand is then read out on an R1 cycle and also gated to the A and B registers. As each position of the B register is implemented in binary-coupled triggers, the state of each position of the B register changes every time a bit is gated to that position. That is, the B register performs the logical EXOR function on the two operands, as shown in the previous table.

The result in the B register is gated to the SDR from where it is stored in the first-operand location on the write section of the R1 cycle.

The result in the B register is tested for a zero result and, if it is zero, the condition code is set to 00. If the result is non-zero, the condition code is set to 01.

The sequence and timing of this instruction are shown in FEMD, Figure 6045/6.

ACCELERATOR only: For RX format, the machine takes an EA cycle and reads out the second operand to the SDR. From here it is gated to the A and B registers. The first operand is then set into the A and B registers. Because each position of the B register is implemented in binary-coupled triggers, the state of each position of the B register changes every time a bit is gated to that position. That is, the B register performs the logical EXOR function on the two operands as shown in the example under the preceding section.

The result in the B register is stored in the first-operand location.

The result in the B register is tested for a zero result and, if it is zero, the condition code is set to 00. If the result is non-zero, the condition code is set to 01.

For RR format, the second operand is gated from the GPR defined by Rb during the I-cycle, and set to the A, B and C registers.

The compute clock is restarted under sequence-2 control and the first operand is gated from the GPR defined by Ra, and set to the A, B and C registers. At this point, the OR of the two operands is in the A register and the EXOR in the B register.

'End execute' is signalled and the B register content is gated to the GPR defined by Ra.

The sequence and timing of this instruction are shown in FEMD, Figures 6245/6 (RR) and 6247/8 (RX).

SI Format

For this operation, the A register is deconditioned and is initially reset. The B register is initially conditioned and the immediate operand (I2) from the Ra and Rb registers enters the B register via the HW funnel and the ABC funnel under byte control. An EA storage cycle reads out the first operand into the SDR, and the byte address controls the byte transferred from the SDR to the ABC funnel (byte control). Operand 1 enters the designated byte of the B register and is EXOR'ed with operand 2 by the

action of the binary-coupled input to the register. The B register is then deconditioned to prevent further inputs from changing the resultant EXOR.

Under byte control the appropriate byte of the SDR is reset and then set with the resultant EXOR byte from the B register. The SDR content is written into storage during the split-cycle write phase, and the condition code is set by analysis of the B register contents. Finally, 'end execute' is signalled. The sequence and timing for this instruction is shown in FEMD, Figure 6089/90 (basic) and 6297/98 (accelerator).

AND Instruction

- The instruction is in either RR, RX or SI format.
- The respective op codes and mnemonics are 14 (NR), 54 (N) and 94 (NI).
- The first and second operands are combined, bit by bit, in an AND function and the result stored in the first-operand location.
- Both the A and B registers are used to perform the AND function.
- The condition code is set as follows:
  00 : Result is zero.
  01 : Result is non-zero.
  10 : Not set.
  11 : Not set.

In the logical AND instruction, the corresponding bits of each of the first and second operands are combined in a logical AND function. This AND function is summarized as follows:

AND		First Operand Bit 0	1
Second Operand Bit	0	0	0
	1	0	1

The AND function can be performed by the combination of the OR and EXOR functions previously described. If the result of the OR function is itself EXOR'ed with the result of the EXOR function, then the result is the AND function. This is demonstrated, as follows, using two operands labelled X and Y.

X	Y	X + Y OR	X ⊻ Y EXOR	(X + Y) ⊻ (X ⊻ Y) OR V EXOR	X. Y AND
0	0	0	0	0	0
0	1	1	1	0	0
1	0	1	1	0	0
1	1	1	0	1	1

The AND function can be performed by forming the OR function of the two operands in the A register, the EXOR of the two operands in the B register, and combining, or EXOR'ing, these two partial results by gating the A register to the B register via the ABC funnel. As demonstrated in the preceding table, the result in the B register will be the AND function of the two operands.

RR and RX Formats

BASIC only: The machine executes the AND instruction by reading out the second operand to the SDR on the R2 or EA cycle, and gating it to the A and B registers. The first operand is then read out to the SDR on an R1 cycle, and gated from the SDR to the A and B registers.

The A register is then gated to the B register, forming the result in the B register, as described previously. The B register is then set to the SDR from where it is stored in the first-operand location on the write section of the R1 cycle.

The result in the B register is tested and, if it is zero, the condition code is set to 00. If the result is non-zero, the condition code is set to 01.

The sequence and timing of this instruction are shown in FEMD, Figure 6045/6.

ACCELERATOR only: The machine executes the AND instruction by gating the second operand to the A and B registers. The first operand is then gated to the A and B registers, so that the A register contains the OR of operands 1 and 2 and the B register contains the EXOR.

The A register is then gated to the B register, forming the AND result in the B register as previously described. The B register is then set via SDR to the first-operand location.

The result in the B register is tested and, if it is zero, the condition code is set to 00. If the result is non-zero, the condition code is set to 01.

The sequence and timing of this instruction are shown in FEMD, Figures 6253/4 (RR) and 6255/6 (RX).

SI Format

For this operation both the A and B registers are conditioned. Thus, the immediate byte, operand 2, from the Ra and Rb registers enters both the A and B registers under byte control. The specified byte of operand 1 (from SDR) also enters the A and B registers under byte control. The two single-byte operands are OR'ed in the A register and EXOR'ed in the B register.

The OR content of the A register is now set into the binary-coupled B register producing an EXOR of the OR and EXOR functions of operands 1 and 2. This result is equivalent to an AND function of operand 1 and operand 2. The SDR is reset in the designated byte before the resultant AND byte is gated to the SDR under byte control. The complete word in the SDR is stored in the first-operand location, the condition code is set, and 'end execute' is signalled. The sequence and timing for this instruction are shown in FEMD, Figures 6089/90 (basic) and 6297/98 (accelerator).

ADD LOGICAL, SUBTRACT LOGICAL AND COMPARE LOGICAL INSTRUCTIONS

Logical instructions are a separate set distinct from arithmetic instructions. They differ in that the operands are used as 32-bit unsigned binary integers. In an arithmetic operation the operands keep the high-order bit (00) as the sign indicator.

Add Logical Instructions

- The instruction is in either RR or RX format.
- The respective op codes and their mnemonics are 1A (ALR) and 5A (AL).
- The first operand is added to the second operand and the result stored in the first-operand location.
- Operands are treated as 32-bit unsigned binary integers.
- The result and carry-out of bit 0 are recorded in the condition code:
  - 00 : No carry: result is zero.
  - 01 : No carry: result is not zero.
  - 10 : Carry: result is zero.
  - 11 : Carry: result is not zero.

The principles of operation of the binary add operation are described in Chapter 1 of Introduction and Functional Units, Form Y33-0001.

The sequence of operation is similar to the arithmetic add instruction. The carry-out of bit 0 of the result is used to set PSW 2 bit 2, and the B register not zero is used to set PSW 2 bit 3. These positions of PSW 2 are the condition code of the PSW.

The add logical instruction may be used for the addition of operands more than one word long; it is used to add the low-order bits. The carry from this operation is used to develop a carry-in for the addition of the high-order bits.

The sequence and timing of these instructions are shown in FEMD, Figures 6015/6 for basic machines and 6217/8 (RR), 6219/20 (RX) for the accelerator.

Subtract Logical Instructions

- The instruction format is in either RR or RX format.
- The respective op codes and their mnemonics are 1F (SLR) and 5F (SL).
- The second operand is always complemented prior to addition by dc setting the B and C registers to ones.
- The second operand is subtracted from the first operand and the result is stored in the first-operand location.
- Operands are treated as 32-bit unsigned binary integers.
- The result and carry-out of bit 0 is recorded in the condition code.

The principles of operation of the binary subtract operations are described in Chapter 1 of Introduction and Functional Units, Form Y33-0001.

The sequence of operation is similar to the arithmetic subtract instruction. The carry-out of the bit 0 position and the B register not zero are used to set the condition code in the PSW in a similar manner to the add logical instruction.

The sequence and timing of these instructions are shown in FEMD, Figures 6015/6 for basic machines and 6217/8 (RR), 6219/20 (RX) for the accelerator.

Compare Logical Instructions

- Compare logical instructions can be in RR, RX or SI format.
- The respective op codes and mnemonics are 15 (CLR), 55 (CL) and 95 (CLI).
- The two operands are compared and the result is indicated in the condition code.
- The two operands remain unaltered in storage.
- The operands are treated as two unsigned binary fields.
- The condition code settings are as follows:
  - 00 : Operands equal.
  - 01 : First operand is low.
  - 10 : First operand is high.
  - 11 : Not set.

RR and RX Formats

The compare logical instructions of both RR and RX formats cause the 32-bit first operand to be compared with the 32-bit second operand. Each operand is treated as an unsigned binary word, and the operation is executed by subtracting the second operand from the first operand and analyzing the result.

The only differences between the compare logical operation and the subtract logical operation are in the conditions used to set the condition code and the handling of the first operand.

In the compare logical operation the result of the subtraction is not stored, but the first operand GPR is regenerated in storage.

The sequence of operations for fetching the operands for the subtraction process has been described previously. The result of the subtraction is obtained in the B register, and it is this register that is analyzed to determine the setting of the condition code as follows:

B Register	Condition Code	Condition
Zero and C0 = 1	00	Operands equal
Not C0 = 1	01	First operand is low
Non zero and C0 = 1	10	First operand is high
	11	Never set

The sequence and timing of these instructions are shown in the FEMD, Figures 6017/8 for the basic machines and 6221/2 (RR), 6223/4 (RX) for the accelerator.

SI Format

The Compare Logical Immediate (CLI) instruction causes the field of immediate data in the instruction to be compared with the byte of data from storage defined by the developed effective address. The result of this comparison is used to set the condition code.

The operation consists of subtracting the second operand (immediate data) from the first operand (storage byte) and analyzing the result.

In preparation for the subtraction, the B and C registers are set to ones and the A register is reset. Operand 1 is read out of storage and enters the SDR. Analysis of the last two bits of the storage address gives the byte address, which is used to gate the correct byte of SDR to the ABC funnel and on to the A, B and C registers. Note that SDR true is always gated to the ABC funnel. Byte address analysis, called byte control, is used to ensure that the correct bytes of operands 1 and 2 are gated to the ABC funnel.

The byte of immediate data (operand 2) is set from the Ra and Rb registers into the HW funnel in positions 16 to 23 and 24 to 31. Under byte control, the operand-2 byte is gated into its correct byte position of the ABC funnel and so into the B and C registers. This completes the partial sum in B and C registers. The subtract operation is completed, as described under subtract logical instruction, by gating the CLA to the ABC funnel. The result obtained in the B register is analyzed in the same way, and the condition codes have the same meaning, as for the RR and RX format described under "Compare Logical Instructions."

The sequence and timing of the SI format compare instruction are shown in FEMD, Figures 6089/90 for basic machines and 6297/98 for the accelerator.

CHARACTER HANDLING INSTRUCTIONS

Insert Character Instruction

- The instruction is in RX format.
- The op code and mnemonic is 43 (IC).
- The eight-bit character at the second operand address is inserted into bit positions 24 to 31 of the GPR specified by the R1 field.
- The remaining bits of the GPR (bits 0 to 23) remain unchanged.
- The byte inserted is neither inspected nor altered.
- The condition code is not altered.

The word containing the selected character is read out of storage to the SDR during an EA cycle. The byte defined by the effective address is then aligned into bits 24 to 31 of the B register by gating and shifting as follows:

EA Bits 30, 31	Gate SDR	Shift Right Eight
00	Criss-cross	Yes
01	Criss-cross	No
10	True	Yes
11	True	No

BASIC only: An R1 cycle is taken and the GPR read out of storage. Bits 0 to 23 of the GPR content are set into the SDR bits 0 to 23, and bits 24 to 31 of the B register are gated to the SDR bits 24 to 31.

ACCELERATOR only: Bits 24 to 31 of the B register are gated to bits 24 to 31 of the GPR defined by the Ra register.

The SDR is then stored on the write section of the R1 cycle.

Thus, the character (or byte) defined by the developed effective address is stored in bits 24 to 31 of the GPR, leaving bits 0 to 23 of the GPR unaltered.

The timing and sequence of this instruction are shown in FEMD, Figures 6051/52 (basic) and 6259/60 (accelerator).

Store Character Instruction

- The instruction is in RX format.
- The op code and mnemonic is 42 (STC).
- Bit positions 24 to 31 of the GPR defined by the R1 field are placed in the byte defined by the second operand address.
- The byte stored is neither inspected nor altered.
- The condition code is not altered.

BASIC only: Instructions of the store type require that the first operand be read out before the second operand. This means that the developed effective address has to be stored temporarily in the BX register, out of the main data flow, while the first operand is fetched. This is done with a B to BX register interchange.

The first operand is read out of extension storage into the SDR. Bits 24 to 31 are then aligned to the byte defined by the effective address, by the following operations:

EA Bits 30, 31	Gate SDR	Shift Left Eight
00	Criss-cross	Yes
01	Criss-cross	No
10	True	Yes
11	True	No

The effective address is brought back into the B register by a re-interchange of the B and BX registers, and an EA cycle is initiated. The effective address is used to set the SAR on this cycle and once again a B to BX register interchange is performed.

The word defined by the effective address is read out of storage, but only three bytes are set to the SDR. The fourth byte, which is not set to the SDR, is the byte defined by the effective address. This byte position in the SDR is set from the corresponding B register byte, in which the first operand bits 24 to 31 have been aligned.

The SDR is then stored on the write section of the EA cycle.

The timing and sequence of this instruction are shown in FEMD, Figures 6053/4.

ACCELERATOR only: Store instructions require that the first operand be read out before the second operand.

Operand 2 in the GPR is fed to the B register, either by gating straight through, or gating bits 16 to 31 to bits 00 to 15 as required. The B register is shifted-left-four twice if the effective address is an even number designated by bit 31. The object is to align this byte to fill the gap in the word brought into the SDR.

After alignment, the EA byte character of the B register is set to the SDR, and the now complete word, the entire contents of the SDR, is stored.

The timing and sequence of this instruction is shown in FEMD, Figure 6261/2.

Test Under Mask

- The instruction is in SI format.
- The op code and mnemonic is 91 (TM).
- The state of the first operand bits selected by the mask is used to set the condition code.
- The immediate data field (I2) of the instruction is used as the mask.
- The first operand is the character, or byte, defined by the effective address.
- A mask bit of one indicates that the corresponding storage bit is to be selected.
- A mask bit of zero indicates that the corresponding storage bit is to be ignored.
- The character in storage is not changed.
- Testing is performed between the Ra and Rb registers and bits 00 to 07 of the B register.
- The condition code is set as follows:
  - 00 : Selected bits all zero; mask is all zero.
  - 01 : Selected bits mixed zero and one.
  - 10 : Not set.
  - 11 : Selected bits all one.

BASIC only: The first operand character is fetched from storage and aligned to bits 0 to 7 of the B register. The controls are:

EA Bits 30,31	Gate SDR	Shift Left Eight
00	True	No
01	True	Yes
10	Criss-cross	No
11	Criss-cross	Yes

Bits 0 to 7 of the B register, and the I2 field of the instruction from the Ra and Rb registers, are analyzed in logic circuits.

The sequence and timing of this instruction are shown in FEMD, Figure 6055/56.

ACCELERATOR only: The machine takes an EA cycle, and operand 1 is read out to the SDR. It is then gated to the B register, via the true/criss-cross, and aligns the byte (defined by the effective address) into either bits 00 to 07, or bits 08 to 15 of the B register.

To perform the test, it is necessary to align the first-operand byte to the B register bits 00 to 07. Thus, for the case where the effective address is an odd address, the compute clock is run under the control of sequence 5, and two shift-left-four operations are performed.

When the first-operand byte has been correctly aligned, it is tested in logic circuits under control of the mask, or I2 field of the instruction, in the Ra and Rb registers.

If a bit in the mask field is a one, the corresponding bit in the first-operand byte is selected for testing. If a bit in the mask field is a zero, the corresponding bit in the first-operand byte is ignored. The test is performed by the logic circuit.

The sequence and timing of this instruction is shown in FEMD, Figure 6263/64.

Test and Set

- The instruction is in SI format.
- The op code and mnemonic is 93 (TS).
- The high-order position of the byte defined by the developed effective address is tested.
- The result of the test sets the condition code as follows:
  - 00 : High-order bit is zero.
  - 01 : High-order bit is one.
  - 10 : Not set.
  - 11 : Not set.
- The original information in the tested byte is replaced in storage by all ones.
- The immediate data (I2) field of the instruction is not used.

The machine reads out the word defined by the developed effective address into the SDR, on a split EA cycle.

The selected byte is aligned into either bits 0 to 7 or 8 to 15 of the data flow and gated to the A register as follows:

EA Bit 30		Gate SDR
0	:	True
1	:	Criss-cross

The B and C registers are set to all ones early in the execution of this instruction.

The byte, defined by the low-order bits 30 and 31 of the effective address in SAR 2, is reset in the SDR. Ones are then gated from the B register into this SDR byte position. The SDR is then stored on the write section of the EA storage cycle.

The A register is tested to determine the setting of the condition code. Bit 00 is tested if the effective address bit 31 is a zero, and bit 08 is tested if the effective address bit 31 is a one.

If the bit tested is a one, the condition code is set to 01, and if it is a zero, the condition code is set to 00.

The timing and sequence of this instruction are shown in FEMD, Figures 6057/8 (basic) and 6265/66 (accelerator).

MISCELLANEOUS LOGICAL OPERATIONS

Shift Logical Instructions

- Instructions are in RS format.
- The R3 field is not used for these instructions.
- The low-order six bits of the developed effective address define the number of positions to be shifted (bits 26 to 31).
- The operands, which are unsigned integers, may be either single or double words and are contained in the GPR or GPR pair defined by the R1 field of the instruction.
- The op codes and their mnemonics for the shift logical instructions are as follows:
  Shift left single : 89 (SLL)
  Shift left double : 8D (SLDL)
  Shift right single : 88 (SRL)
  Shift right double : 8C (SRDL)
- Shift single is a single-word operation and shifting is done in the B register.
- Shift double is a double-word operation and shifting is done in the B and BX registers.
- Multiple shift is a series of single-shift operations.
- The shift counter is reduced by one for each shift, and shifting ends when the shift counter equals zero.
- For all logical shifts, zeros are supplied to fill vacated high or low-order bit positions.
- The result is stored in the original operand register.
- The condition code is not altered.

The four logical shift operations are shown in Figure 3A-21.

Shifting is the operation of moving all the bits in a word or double word to the right-hand or left-hand adjacent bit position without disturbing their order.

In these logical operations, the B register bit 00 is of no significance as a sign bit and is treated as an ordinary binary bit.

Shift single is a full-word operation and the operand is read out of the register specified by Ra and set into the B register.

Shift double is a double-word operation and the operand is located in a pair of even and odd addressed registers. Addressing of the operand should always be made on the even-numbered register, otherwise, a specification exception occurs and causes a program interrupt. The machine automatically addresses the odd-numbered register. The even-numbered register address contains zeros in bit positions 29, 30 and 31. The odd-numbered register address is developed by forcing a one in bit position 29. (Set SAR to odd.)

The odd-numbered register content, which is a low-order word, is fed into the BX register via the B register. The even-numbered register content, which is a high-order word, is then fed to the B register.

The low-order six bits (bits 26 to 31) of the developed effective address indicate the number of shift places and are fed into the shift counter before the operand is fetched.

Multiple shift is a series of single-shift operations in which the shift counter value is reduced by one at each shift. Shifting ends when the shift counter is zero.

On shift double, when the B register is shifted right, bit 31 of the B register is shifted into the bit 00 position of the BX register. Bit 00 of the B register is defined as the high-order bit, and bit 31 of the BX register as the low-order bit. In shift-right operations, the low-order bit is shifted out without inspection and is lost. A zero is supplied in the high-order bit position.

In shift-left operations, the action is reversed. Bit 00 of the BX register is shifted into the bit 31 position of the B register. The high-order bit is shifted out without inspection and is lost.

When the shift counter is greater than two, the shift is performed at CC1, CP1 time and the compute clock is wrapped at CC2. In a shift-single operation the B register is shifted and in a shift-double operation the B and BX registers are shifted one position. In either case the shift counter is reduced by one for each cycle.

BASIC only: When the shift counter reaches two, the compute clock runs to CC6, instead of

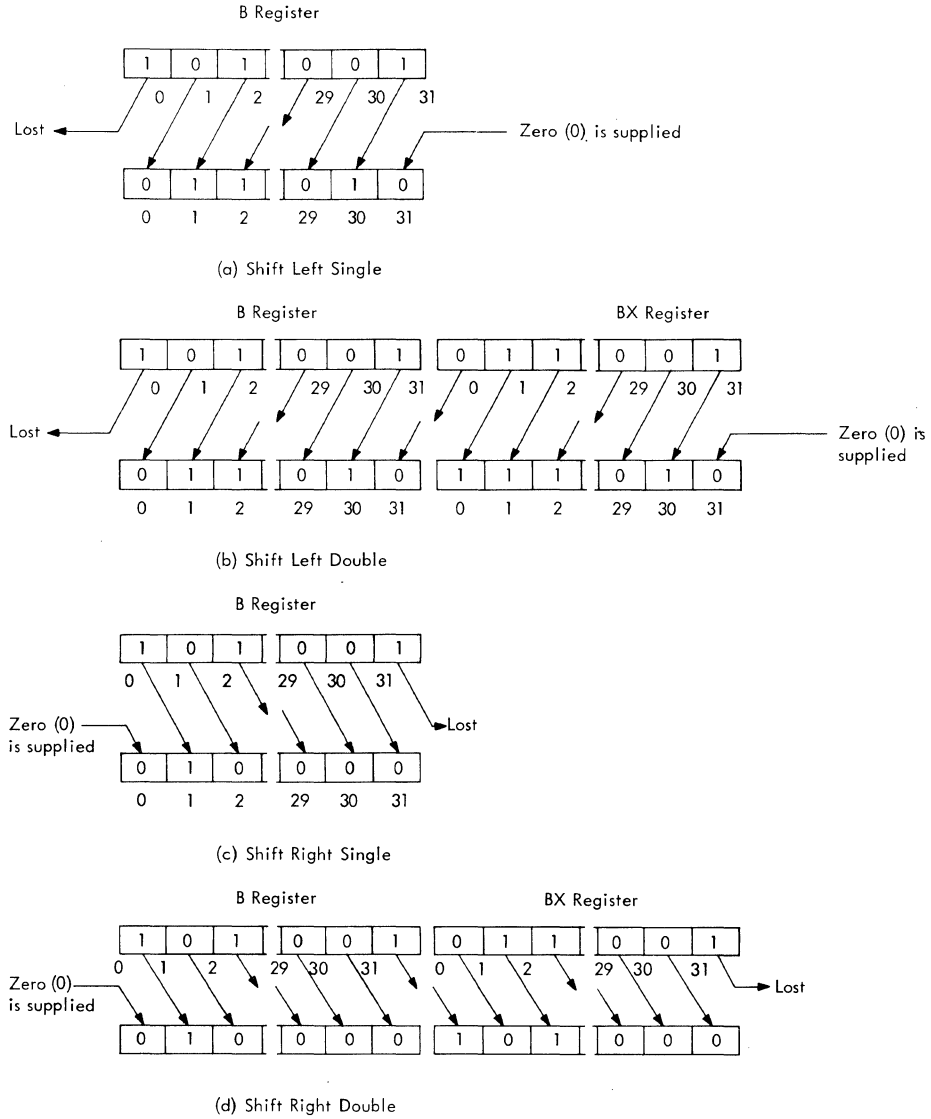


Figure 3A-21. Shift Operations -- Logical

wrapping, and the shift is performed twice, once at CC3, CP1 time and again at CC5, CP1 time.

If the shift counter is set to two at the beginning of the operation, the compute clock runs to CC6, performing the shift operation twice, once at CC1, CP1 time and once at CC3, CP1 time. If the shift counter is set to one at the beginning of the operation, the compute clock runs to CC6, performing the shift operation once at CC1, CP1 time.

If the shift counter is set to zero, the compute clock runs to CC6, but no shift takes place. The condition code remains unchanged.

The result in the B and BX registers is stored in the even and odd-numbered registers respectively. The addressing method is the same as in the operand-fetch cycle.

The sequence and timing of these shift instructions are shown in FEMD, Figures 6019 to 6026.

ACCELERATOR only: When the shift counter is greater than zero, the shift is performed at CC1, CP1 time and the compute clock is wrapped at CC2. In a shift-single operation the B register is shifted and in a shift-double operation, the B and BX registers are shifted one position. In either case the shift counter is reduced by one for each cycle. When the shift counter reaches zero, the compute clock runs to CC6.

If the shift counter is set to two at the beginning of this operation, the compute clock runs to CC6, performing the shift operation at CC1, CP1 time. If the shift counter is set to

zero, the compute clock runs to CC6, but no shift takes place. The condition code remains unchanged.

For shift double, the result in the B and BX registers is gated to the even and odd-numbered first-operand GPR's respectively. For shift single the B register content is gated to the first-operand GPR.

The sequence and timing of these shift instructions is shown in FEMD, Figures 6225 to 6232.

Load Address

- The instruction is in RX format.
- The op code and mnemonic is 41 (LA).
- Loads bits 08 to 31 of the developed effective address into bits 08 to 31 of the GPR specified by the R1 register. Bits 00 to 07 of the word are made zero.
- The condition code is not altered.

BASIC only: The machine takes an R1 cycle, the 'inhibit store to SDR' line is made active and the SDR is reset.

Bits 00 to 07 of the B register, which contains the effective address developed during I-fetch, are reset and the B register is gated to the SDR. From the SDR it is stored in the GPR defined by the R1 field on the write section of the storage cycle.

The sequence and timing of this instruction are shown in FEMD, Figure 6059/60.

ACCELERATOR only: At the end of I-time in this RX instruction, bits 00 to 07 of the B register are reset and 'end execute' is signalled. This leaves the developed effective address bits 08 to 31 in the B register.

The compute clock is started under control of sequence 2, and the B register is gated to the GPR defined by the Ra register.

The sequence and timing of this instruction is shown in FEMD, Figure 6267/68.

Move Instruction

- The instruction is in SI format.
- The op code and mnemonic is 92 (MVI).
- One byte of immediate information (operand 2) is placed in operand-1 location.
- The condition code is not altered.

The move instruction contains the one-byte I2 field (operand 2). This instruction causes the one byte of immediate data (operand 2) to be stored at the location specified by operand 1 (B1 + D1 in the instruction). The condition code is not set at the completion of this instruction.

The sequence and timing of the MVI instruction are shown in FEMD, Figures 6089/90 (basic) and 6297/98 (accelerator). The immediate operand is gated from the Ra and Rb registers to the ABC funnel and enters the B register under byte control. The A register is deconditioned (it was previously reset so that the register contains zeros throughout the operation). After receiving the operand-2 byte, the B register is deconditioned to prevent the entry of the operand-1 byte which follows. The degating of the B register at this time is necessary owing to the use of common logic for the instructions MVI, NI, OI and XI.

The designated byte of the SDR is reset and the second operand byte in the B register is gated to the SDR under byte control. The word in the SDR is written into the operand-1 storage location and the 'end execute' is signalled.

Branching instructions are introduced to allow a departure from the normal sequential execution of instructions. The branch conditions cause the Instruction Counter (IC) section of PSW 2 to be replaced by the branch address. This address is then used on the next instruction-fetch cycle. The no-branch conditions cause the machine to proceed to the next sequential instruction as defined by the IC section of the current PSW 2.

Branching may be conditional or unconditional; additional operations may be performed in conjunction with either of these conditions.

For further details, refer to IBM System/360 Principles of Operation, Form A22-6821 and Introduction and Functional Units, Form Y33-0001.

Instruction Formats

- Branching instructions use either RR or RX formats.
- For RR format, the R2 field (bits 12 to 15) define the GPR containing the branch address; when this field is zero, no branching takes place.
- Bits 8 to 11 of both instruction formats indicate the M1, or mask, field (for branch on condition) or the R1, or GPR address, field (for branch and link, branch on count).

Branch On Condition

- The instruction is in either RR or RX format.
- The respective op codes and mnemonics are 07 (BCR) and 47 (BC).
- Bits 8 to 11 of the instruction are called the M1, or mask, field.
- The M1 field is used as a four-bit mask to correspond with the condition code as follows:

Condition Code		Instruction Bit
00	:	8
01	:	9
10	:	10
11	:	11

- The branch is successful whenever the condition code (PSW 2 bits 2 and 3) has a corresponding mask bit of one.
- The condition code is not altered.

During instruction fetch, the M1 field is set into the Ra register. The current condition code (PSW 2 bits 2 and 3) and the Ra register are checked for matching in an analysis circuit, and if they match, a branch occurs. If they do not match, an 'end execute' signal is developed at the end of I-fetch and the machine proceeds to the next sequential instruction.

An M1 field of all ones causes an unconditional branch because all condition codes are satisfied; conversely, an M1 field of all zeros allows an 'end execute' signal to be developed at the end of I-fetch in the same way as an R2 field of zero (a no-operation). Between these two extremes, one, two or three conditions may be masked.

BASIC only: For branch conditions, if the instruction is in RR format, an R2 cycle is taken to read out the branch address from the GPR and set it into the B register.

ACCELERATOR only: For branching conditions, a compute cycle is taken and, if the instruction is in RR format, the branch address from the GPR is set into the B register.

For RX format the branch address (effective address) is already in the B register.

From this point, both formats follow the same sequence. The B and BX registers are interchanged and the IC of the current PSW is replaced by bits 8 to 31 of the BX register. An 'end execute' signal is developed and the I-fetch proceeds, using the branch address now located in the IC.

The sequence and timing of this operation is shown in FEMD, Figures 6061/62 for basic machines and 6269/70 (RR) and 6271/72 (RX) for the accelerator.

Branch and Link

- The instruction is in either RR or RX format.
- The respective op codes and mnemonics are 05 (BALR) and 45 (BAL).
- The R1 field specifies the GPR into which the current PSW 2 is stored as link information (PSW 2 contains the IC).
- Branching is unconditional, except when the R2 field of an RR format instruction is zero.
- The condition code is not altered.

For RR format, the branch address is first fetched and set to the B register. From this point the operation is similar for both formats, as the B register in both cases now contains the branch address.

The B and BX registers are interchanged, causing the branch address to be contained in the BX register.

BASIC only: The PSW 2 register is set to the B register, and from there is gated to the SDR and stored on the write section of the R1 cycle.

ACCELERATOR only: PSW 2 is set to the B register, and from there is gated to the GPR defined by the Ra register.

The BX register bits 8 to 31 are then set to the IC (provided that Rb $\neq$ 0) and an 'end execute' signal is generated. The operation proceeds using the branch address now located in the IC.

The sequence and timing of this instruction is shown in FEMD, Figures 6065/66 for basic machines and 6277/78 (RR) and 6279/80 (RX) for the accelerator.

Branch On Count

- The instruction is in either RR or RX format.
- The respective op codes and mnemonics are 06 (BCTR) and 46 (BCT).
- The R1 field specifies the count GPR.
- The count GPR is arithmetically reduced by one on each instruction execution.
- Branching occurs when the count is not equal to zero.
- No branching occurs when the count is equal to zero, and normal sequential instructions are executed.
- Arithmetic overflow of the count is ignored.
- The condition code is not altered.
- Counting is performed without branching if the R2 field of the RR format is zero.
- Reduction of the count is achieved by adding the number to all ones (-1). In preparation for this operation, the BX register is set to all ones during this instruction fetch.

BASIC only: In the RR format, the contents of the R2 register are sent to the B register.

ACCELERATOR only: In the RR format, the content of the GPR defined by Rb register is gated to the B register.

After this, with the B register containing the branch address, the operation is the same for both formats.

The branch address goes to the BX register by a B to BX register interchange, in which the B register is set to all ones from BX. The A register is reset and the C register is set to the B register with all ones.

BASIC only: The first operand is fetched and set to the SDR and gated on to the A, B and C registers.

ACCELERATOR only: The first operand is set from the GPR defined by the Ra register to the A, B and C registers.

The B register result is completed by the CLA which is now the count reduced by one.

BASIC only: This result is set into the SDR and is stored back into the GPR on the write section of the R1 cycle.

ACCELERATOR only: This result is set into the GPR.

If the result from either the B register or the Rb register is zero, 'end execute' is generated and the machine proceeds to the next sequential instruction. However, if the result (count) is not zero, 'set IC from BX' is generated. The branch address was stored in BX at the beginning of the operation. After it is transferred to IC, the machine can use it as the address to fetch the next instruction.

The sequence and timing of this operation is shown in the FEMD, Figures 6063/64 for basic machines and 6273/74 (RR) and 6275/76 (RX) for the accelerator.

Set System Mask

- The instruction is in SI format and is a privileged instruction.
- The op code and mnemonic is 80 (SSM).
- The immediate data (I2) field of the instruction is not used.
- The byte at the location defined by the effective address replaces the system mask bits of the current PSW (PSW 1 bits 00 to 07).
- The condition code is not altered.

The operand is read out of storage and the required byte aligned to bits 00 to 07 of the B register by gating as follows:

EA Bits 30, 31	Gate SDR	Shift Left Eight
00	True	No
01	True	Yes
10	Criss-cross	No
11	Criss-cross	Yes

Bits 00 to 07 of the B register are then gated to PSW 1 bits 00 to 07, and the 'end execute' condition is signalled.

If this instruction is encountered when the machine is in the problem state, a privileged-operation exception is recognized and the operation is suppressed.

The sequence and timing of this instruction is shown in FEMD, Figures 6067/68 (basic) and 6281/82 (accelerator).

Set Program Mask

- The instruction is in RR format.
- The op code and mnemonic is 04 (SPM).
- The R2 field is not used in this instruction.
- Bits 2 to 7 of the GPR specified by the R1 field are used to replace the program mask bits of the current PSW (PSW 2 bits 2 to 7).
- Bits 0 and 1 and bits 8 to 31 of the GPR are ignored and the original content of the GPR remains unchanged in extension storage.
- The condition code is set according to bits 2 and 3 of the GPR specified by R1.

BASIC only: The GPR containing the data is read out of storage and gated from the SDR to the B register.

ACCELERATOR only: The GPR containing the data is gated to the B register. 'End execute' is generated.

Bits 02 to 07 of the B register are then gated to the PSW 2 bits 2 to 7, where they replace the condition code (bits 2 and 3) and the program mask (bits 4 to 7). The operation is then terminated by the generation of an 'end execute' signal.

The sequence and timing of this instruction is shown in FEMD, Figures 6069/70 (basic) and 6283/84 (accelerator).

Supervisor Call

- The instruction is in RR format.
- The op code and mnemonic are 0A (SVC).
- The instruction causes a supervisor-call interrupt.
- The R1 and R2 fields provide the interrupt code for the subsequent interrupt.
- The condition code remains unchanged in the old PSW.

This instruction forces a request for a supervisor-call interrupt.

During this instruction fetch the supervisor-call interrupt request is set and 'end execute' is signalled. This leaves the machine free to take the pending supervisor-call interrupt.

The R1 and R2 fields in the Ra and Rb registers provide the bits 24 to 31 of the interrupt code when the old PSW is being stored. Except for when the old PSW is being stored, the interrupt caused by the supervisor-call instruction is similar to all other interrupts. (See "Interrupts" section.)

The sequence and timing of signals generated during this instruction fetch is shown in FEMD, Figures 6071/72 (basic) and 6285/86 (accelerator).

Load PSW

- The instruction is in SI format and is a privileged operation.
- The op code and mnemonic is 82 (LPSW).

- The immediate data (I2) field of this instruction is not used.
- The double word at the location defined by the developed effective address replaces the current PSW.
- The condition code is set according to bits 2 and 3 of the new PSW 2 loaded.
- A double EA cycle is required to fetch the PSW.
- PSW 1 bits 0 to 15 are set from the B register.
- PSW 1 bits 16 to 31 are not in registers; they are the interrupt code inserted during the interrupt.
- PSW 2 bits 0 and 1 cannot be loaded; they reflect only the state of op register bits 0 and 1.
- PSW 2 bits 2 to 7 are set from the B register.
- PSW 2 bits 8 to 31 are set from the BX register.
- The machine enters the problem state when the load-PSW loads a double word with a one in bit position 15.
- The machine enters the wait state if the load-PSW loads a double word with a one in bit position 14.

The machine takes a double EA cycle for the load-PSW instruction. On the first EA cycle, bits 0 to 31 of the new PSW are read out to the SDR and gated to the B register. The PSW 1 is then set from the B register. Only bits 0 to 15 are set into PSW 1, as bits 16 to 31 are generated by an interrupt code when the old PSW 1 is being stored (refer to "Interrupts" section).

Bits 8 to 11 are also checked for zero when the storage protection feature is not installed. If they are not zero a specification exception occurs.

The double EA cycle latch causes another EA cycle to occur consecutively without the possibility of an intervening I/O storage cycle. It inhibits the reset of the SAR and forces on bit 29 of each SAR (increasing each SAR by four) for the second cycle. Thus, the consecutive word is read out of storage to the SDR on this cycle.

Bits 32 to 63 of the operand are gated from the SDR to the B register; they are also set into the BX register by a B to BX register interchange, in which the BX to B path is inhibited.

The B register bits 2 to 7 are then gated to PSW 2 bits 2 to 7, and the BX register bits 8 to 31 are gated to PSW 2 bits 8 to 31. The 'end execute' signal is then generated and the machine proceeds, using the new IC address fetched on the load-PSW instruction.

If this instruction is encountered when the machine is in the problem state, a privileged-operation exception is signalled and the operation is suppressed.

The sequence and timing of this instruction is shown in FEMD, Figures 6073/74 (basic) and 6287/88 (accelerator).

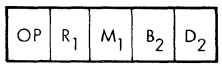
Diagnose Instruction

- The instruction is in RS format and is a privileged instruction.
- The op code and mnemonic is 83 (DIAG).
- Bits 12 to 15 of the instruction are used as a mask field (M1).
- Storage accesses on the execute phase are to extension storage.
- Acts as either a load or store operation depending on mask bit 3.
- Turns on or off 'test parity equal to one' depending on mask bit 2.

Instruction Format

The instruction is in RS format as follows:

DIAG



DIAG = Mnemonic. (For IBM internal use only.)
OP = Op code of 83.
R1 = Address of source or destination GPR (according to M1 field).
M1 = Mask field.
B2 = Base register address.
D2 = Displacement.

The mask field is a four-bit field defined as follows:

Bit 0: Not used.
Bit 1: Not used.
Bit 2: Mask bit of zero turns off test parity latch. Mask bit of one turns on test parity latch.
Bit 3: Mask bit of zero causes a load operation. Mask bit of one causes a store operation.

The B2 and D2 fields are used to generate an effective address, but this is converted to an access to extension storage by turning on the extension storage latch for the EA cycle of this instruction.

This instruction is treated as a privileged operation, and if it is encountered when the machine is in the problem state a privileged-operation exception is signalled and the operation is suppressed.

Instruction Applications

The use of this instruction is:

1. For extension storage worst-case and pattern tests, using main storage programs to address extension storage. (There is not enough room in the 32K machine to write a relocatable section to operate in the storage extension alone.) This will allow the optimum operating point voltage to be adjusted with one program for storage extension and the main storage.

2. To provide the ability to check the contents of unit control words located in the storage extension. This ability is used in certain multiplexor channel diagnostics to check stacked-status operations.

3. To provide the ability to set and reset the 'test parity equals one' condition by program means, in addition to the console switch facility.

Operation of Test Parity Latch

The state of Rb register bit 2 is tested at WC4, CP2 during a diagnose operation. If this bit is a one, the test parity latch is turned on and remains on until reset by a system reset or by another diagnose instruction where the mask bit 2 is a zero. The output is blocked during an interrupt cycle.

The effect of the latch is to allow a one to be placed in the SDR parity bit position of a byte where a B register to SDR transfer is being carried out.

The same effect may be obtained manually by operating the 'test parity equals one' console switch. In this case, the output of the test-parity latch is activated by the switch as if the latch was set on. This output remains active, blocked only by an interrupt cycle, until the console switch is released. The action of the switch completely overrides the mask bit of the diagnose instruction. Operation of the 'test parity equals one' switch illuminates the test lamp. Whenever the output of the test-parity latch is active, irrespective of the source, the interval timer is suspended.

Instruction Execution

The execution of the diagnose instruction is dependent on bit 3 of the mask field (M1) of the instruction.

If the mask bit is zero, a load operation is performed similar to that described for the load instruction. If the mask bit is one, a store operation is performed similar to that described for the store instruction.

In both cases the extension storage latch is on for the EA cycle, and the storage access is converted from a main storage to an extension storage address.

For details of the sequence and timing of the execute phase of this diagnose instruction, refer to FEMD, Figure 6075/76 (basic) and 6289/90 (accelerator).

UPDATE INTERVAL TIMER

- Timer value is contained in main storage fixed location 50 hex.
- A real-timer constant is used to decrement the timer value.
- A timer cycle request is generated once per cycle of line input frequency.
- Timer value update can be disabled from the console by 'disable interval timer' switch and 'test parity equals one' switch.
- Timer value update is suspended when the machine is in either of the stopped states, during system reset and during instruction step.
- When the timer value changes from positive to negative an external interrupt is requested.

A fixed storage location is set aside at location 50 hex where a full word of timer value may be inserted. This value may be changed at any time by storing a new value provided that the location is not protected. The objective is to put in a value at location 50 hex and produce a constant, at line frequency, to reduce the value to a point where it changes from positive to negative. At this point an external-interrupt request is made. The maximum value of the full word at location 50 hex is 15.5 hours real time.

The timer update is initiated by a singleshot driven by a 7v ac transformer. The singleshot gives a pulse once per cycle. Thus, the opportunity to update occurs either 50 or 60 times per second, dependent on supply frequency. The output of the singleshot sets the timer cycle request latch which requests priority to inhibit I-cycle control and reduce the timer value in storage. The timer cycle request latch cannot be set if the disable interval timer or force parity switches are active.

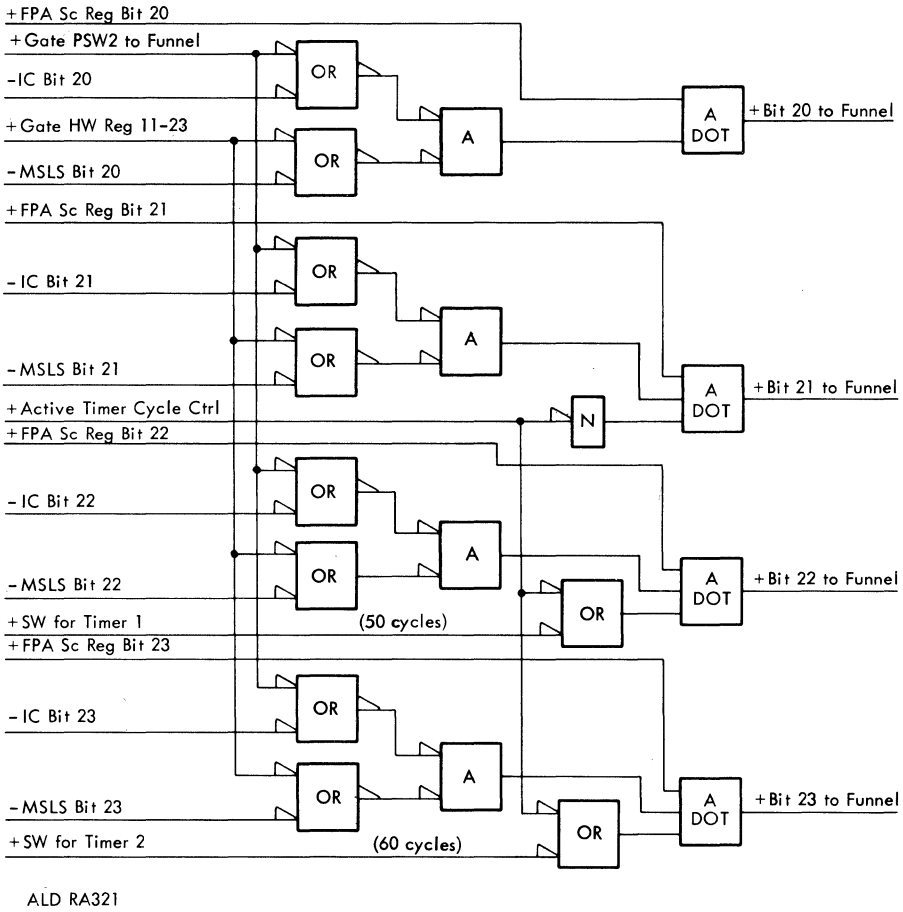


Figure 3A-22. PSW 2 and HW Register to Funnel Logic

When the 'end execute' occurs, it sets the timer cycle control latch if I/O or console operations are not in progress. These two operations inhibit the output of the active timer cycle control latch so that this output is available to set the T-cycle latch when I/O and console functions are complete. Address 50 hex is forced into SAR and the old timer value is set into the SDR while the A, B and C registers are reset. A subtract operation is started by the subtract trigger to reduce the old timer value by the constant. The value of this constant is -600 hex for 50 cycles per second, and -500 hex for 60 cycles per second machines. (See Figure 3A-22 for the HW funnel logic.)

The constant is allowed to be fed from the negative logic construction at the HW funnel. The output of the HW funnel is logical ones, if not otherwise gated, except that the line 'T-cycle not console function' inhibits the one from bit 21 and, combined with the choice of supply frequency, inhibits a one from either bit 22 or 23. The net result is gated to the B and C registers in complement form as follows:

1111 1111 1111 1111 1111 1001 1111 1111 (-600 hex) 50 cps
1111 1111 1111 1111 1111 1010 1111 1111 (-500 hex) 60 cps

The choice of supply frequency is determined by a switch card inserted on installation of the machine (Figure 3A-23). In Figure 3A-22, note that both the gate PSW 2 and gate hardware register lines are negative, and the floating-point scratch register output is positive when it is not gated onto its output bus. As the constant is now in the B and C registers, the old timer value is brought from the SDR into the A, B and C registers.

With the subtract trigger set, an arithmetic cycle takes place. The A and C registers set the CLA,

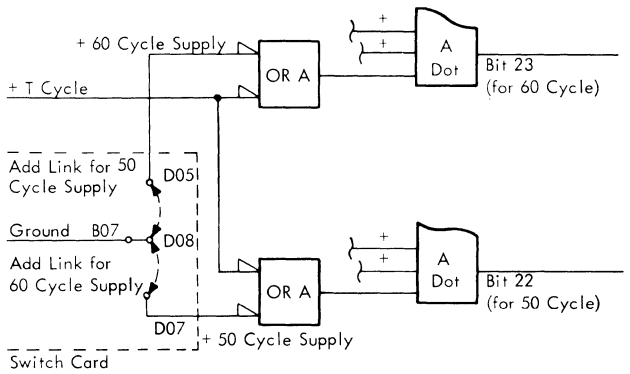


Figure 3A-23. Main Frequency Determination

and the result appears in the B register. This is the new timer value, which goes from the B register to the original location in storage to replace the old timer value on the last write section of the T-cycle.

When the A register contains the old timer value and the B register contains the new timer value, bit 00 of both registers is sampled at the final RC4 time.

If A register bit 00 is a positive sign and B register bit 00 is a negative sign, this implies that the latest subtraction has brought about a change from a small positive number to a negative number. An external interrupt is then requested. This is the only condition to cause an interrupt request. A change from A register negative sign to B register positive sign (an overflow condition) is ignored.

The sequence and timing of the operation is shown in FEMD, Figures 6087/8 (basic) and 6295/96 (accelerator).

INTERRUPTS

The interrupt system permits the CPU to change its status as a result of conditions in the CPU itself, conditions external to the system, or conditions in I/O units. The five classes of interrupts and their priorities are:

Class of Interrupt	Priority
Internal machine check	1
Supervisor call or program	2*
External interrupt	3
External machine check	4
I/O	5

*The program and supervisor-call interrupts are mutually exclusive and cannot occur at the same time.

At the end of each instruction execution, an 'end execute' signal is generated and used to initiate another instruction fetch if no inhibiting conditions are present. One of the inhibiting conditions is a pending interrupt request. To take the interrupt, the interrupt request line has to be active before the 'end execute' signal. If the interrupt request line is active, the 'end execute' signal is used to set the interrupt cycle instead of the usual next I-cycle. Thus, an interrupt, if pending, is always taken after one instruction execution is finished and before a new instruction is started.

Some interrupts may be masked off by the programmer. The I/O and external interrupts may be masked by the system mask. Three of the ten program interrupts may be masked by the program mask, and the machine-check interrupt may be masked by the machine-check mask (Figure 3A-24).

Interrupt Type	Priority	Source Identification	Interrupt Code PSW Bits 16 to 31	Mask Bits 1=PSW 1 2=PSW 2	ILC SMT	Execution
Internal Machine Check	1	CPU Control Check	0000 0000 0001 0000	1/13	x	Terminate
		SDR Check	0000 0000 0000 1000	1/13	x	Terminate
Program	2	Operation	00000000 00000001		1,2,3	Suppress
		Privileged Operation	00000000 00000010		1,2	Suppress Terminate
		Addressing	00000000 00000101		0,1,2,3	Suppress Terminate
		Specification	00000000 00000110		1,2,3	Suppress
		Fixed-Point Overflow	00000000 00001000	2/5	1,2	Complete
		Fixed-Point Divide	00000000 00001001		1,2	Suppress Complete
		Exponent Overflow	00000000 00001100		1,2	Terminate
		Exponent Underflow	00000000 00001101	2/7	1,2	Complete
		Significance	00000000 00001110	2/8	1,2	Complete
		Floating-Point Divide	00000000 00001111		1,2	Suppress
Supervisor Call	2	Instruction Bits	00000000 rrrrrrrr		1	Complete
External	3	External Signal 1	00000000 xxxxxxx1	1/7	x	Complete
		External Signal 2	00000000 xxxxxx1x	1/7	x	Complete
		External Signal 3	00000000 xxxxx1xx	1/7	x	Complete
		External Signal 4	00000000 xxxx1xxx	1/7	x	Complete
		External Signal 5	00000000 xxx1xxxx	1/7	x	Complete
		External Signal 6	00000000 xx1xxxxx	1/7	x	Complete
		Interrupt Key	00000000 x1xxxxxx	1/7	x	Complete
		Timer	00000000 1xxxxxxx	1/7	x	Complete
External Machine Check (Channel Ctl)	4	Channel 2 Error	1000 0001 000X X000	1/13	x	Terminate
		Channel 1 Error	1000 0010 000X X000		x	Terminate
		Channel 0 Error	1000 0100 000X X000		x	Terminate
External Machine Check (Interface Ctl)	4	Channel 2 Error	1000 0001 0000 0000	1/13	x	Terminate
		Channel 1 Error	1000 0010 0000 0000		x	Terminate
		Channel 0 Error	1000 0100 0000 0000		x	Terminate
Input/Output	5	HSMPX Channel 2	00000010 aaaaaaaa	1/2	x	Complete
		HSMPX Channel 1	00000001 aaaaaaaa	1/1	x	Complete
		Multiplexor Channel	00000000 aaaaaaaa	1/0	x	Complete

Notes

a Device Address Bits
r Bits of R1 and R2 Field of Supervisor Call
x Unpredictable

Figure 3A-24. Table of Interrupts

During execution of an instruction, more than one interrupt-causing event may occur. The causes of these interrupts are held in various latches until the interrupt cycle is taken. If more than one class of interrupt is pending at the end of instruction execution, the machine must decide which interrupt is to be taken first.

The order of priority is shown in Figure 3A-24. The program and supervisor-call interrupts are mutually exclusive and cannot occur at the same time.

The way in which the preceding instruction is finished may be influenced by the cause of the interrupt. The instruction is said to have been completed, terminated or suppressed.

In the case of instruction completion, results are stored and the condition code is set as for normal instruction operation, although the result may be influenced by the exception that has occurred.

In the case of instruction termination, all, part, or none of the result may be stored. Therefore, the result data is unpredictable. The setting of the condition code, if called for, may also be unpredictable. In general, the results should not be used for further computation.

In the case of instruction suppression, the execution proceeds as if no operation were specified. The results are not stored and the condition code is not changed.

The machine can handle only one class of interrupt at a time, and when more than one interrupt is requested, they must be taken in order of priority. The interrupt class latches are set in the order, or priority, listed previously. An interrupt class latch cannot be set unless all interrupt latches of higher priority are off.

An interrupt cycle consists of storing the current PSW as an old PSW and fetching the new PSW as a current PSW. To exchange the PSW's, the CPU takes four storage cycles.

An 'end execute' signal is active at the end of the interrupt cycles in the same way as for normal

Type of Interrupt	Priority Class	Priority Irpt	Cause	Exception or Check Latch	Request Latch	Accept Latch	Irpt Latch	Comment
Internal Machine Check	1	- - - -	Sequence Check Cycle Ctl Chk Any SDR Parity Chk CPU Console Forced Chk Error	X C X C X C X	X		X	'Machine check mask auxiliary' latch is set by 'machine check mask on' giving 'allow machine interrupt'.
Program	2	- - - - - - - - - -	Operation Privileged Op Addressing Specification Fixed-point Overflow Fixed-point Divide FP Exp Overflow FP Exp Underflow Significance FP Divide	X C X C X C X C m X C X C X C m X C m X C X C			 X 	The highest internal priority interrupt is indicated in interrupt code.
Supervisor Call	2	-	Supervisor Call Op		X		X	Interrupt code is set from the bits 8 to 15 of the instructions
External	3	- - - - - - - - -	External Signal 1 External Signal 2 External Signal 3 External Signal 4 External Signal 5 External Signal 6 Interrupt Key Timer		X mo X mo X mo X mo X mo X mo X mo X mo	X C X C X C X C X C X C X C X C	 X 	One mask bit controls all of the External interrupts All accepted signals are indicated in interrupt code at the same time.
External Machine Check	4	1 1 2 2 3 3	Channel 2 Interface Chk Channel 2 Channel Ctl Channel 1 Interface Chk Channel 1 Channel Ctl Channel 0 Interface Chk Channel 0 Channel Ctl	X C X C X C X C X C X C	X X X X X X		 X 	Auxiliary latch is operative. Interrupt code (PSW1 bits 27 & 28) indicate channel control check.
Input Output	5	1 2 3	HSMPX Channel 2 HSMPX Channel 1 MPX Channel		X mo X mo X mo	X X X	X	Interrupt code is the channel address and the device address

NOTES: C: Indicates that this latch is used to set the interrupt code.
m: The mask bit should be 1 to set the latch.
mo: The output line is activated if mask bit of 1 is applied.
X: Indicates that this latch is on.

Figure 3A-25. Summary of Interrupt Control Latches

instruction execution. If another interrupt request is pending, it will be taken at this time. Otherwise, the next instruction is fetched, from the storage location defined by the new value of the IC, and executed. Machine status is indicated by the new PSW, which now becomes a current PSW for subsequent operations.

ACCEPTING THE INTERRUPT

- When multiple interrupts are present, only one can be dealt with at one time.
- Simultaneously occurring interrupts of different class are accepted in a predetermined priority.
- Some interrupts can be prevented from occurring by mask bits in the current PSW.

The system used to establish the priority, and the system of accepting each of the different classes of interrupt are described in the following sections. The latches and conditions used are summarized in Figure 3A-25.

Machine-Check Interrupt

The machine check interrupt has the highest priority.

This interrupt can be masked on or off. If the mask bit is 0 the interrupt is ignored; if the mask bit is 1, a machine check causes a machine-check interrupt.

The CPU errors which cause the machine-check interrupts are:

1. SDR parity check
2. Sequence check
3. Cycle control check
4. Console-forced machine check.

The 'allow machine check' line must be conditioned to allow a machine check to cause a machine-check interrupt. For this line to be conditioned, the machine-check mask must be 1, and the check control switch on the console must not be in either the stop or the disable position.

When a machine check occurs, the 'allow machine check' signal combines with the CPU error signal and inhibits any wrapping of the compute clock that may be occurring. At the next CC6 an 'end execute' signal is generated which immediately terminates the current operation. The signal generated by the CPU error also initiates the machine-check interrupt request, and is used to turn on the machine-check interrupt latch to define the interrupt cycles being performed.

Once the machine-check interrupt cycles are started, the CPU error latches are reset. This allows the machine to check for further CPU errors that may occur during the machine-check interrupt cycles and subsequent operations.

If a machine-check interrupt is requested during any interrupt cycle, the current interrupt cycle is terminated at the next WC4 time by resetting the interrupt latch and forcing the 'end execute' signal. If the class of the terminated interrupt is other than machine check, a machine-check interrupt is then taken.

At the start of the machine-check interrupt, the 'double error sample' latch is set, and remains on until near the end of the interrupt cycles. See FEMD, Figures 6079/80 (basic) and 6293/94 (accelerator). This latch is set to avoid the possibility of the machine permanently executing machine-check interrupt cycles, if a machine check occurs during the machine-check interrupt cycles. The latch causes a machine hardstop condition if any CPU errors occur during the machine-check interrupt cycles.

Program Interrupt

This class of interrupt may be caused by up to ten exception conditions; there is an internal priority for these exceptions, and the exception with the highest priority has to be identified.

Each exception sets an individual exception latch. Three of these exceptions need a mask bit of 1 to allow the latch to be set: fixed-point overflow, floating-point exponent underflow, and floating-point significance. If the mask bit is zero, the latches are prevented from setting and the interrupt is ignored. The exception latch for the exception with the highest internal priority is used to set the interrupt code. The exception latch outputs are OR'ed together, and used to request the interrupt and to set the program interrupt latch. This program interrupt latch is not set if the machine-check interrupt request latch is on. Thus, the program interrupt has a lower priority than the machine-check interrupt.

If a machine-check interrupt is taken in priority over a program interrupt, all the exception latches are reset during the last interrupt period of the machine-check interrupt. Accordingly, the program interrupt does not remain pending and all conditions are cleared.

During any interrupt cycle, a specification exception may occur if the protection feature is not installed, and a new PSW with a non-zero protection key is loaded. A program interrupt is taken immediately upon loading the new PSW, unless the machine-check interrupt is pending. The protection key is made zero when the PSW is stored. If the new PSW again has a non-zero protection key, program interrupts will continue to occur, and the process is

repeated with the CPU involved in a string of program interrupts.

This string can be broken only by Initial Program Loading (IPL) or system reset.

If the new PSW for the program interrupt has an unacceptable instruction address, another program interrupt occurs. As this second program interrupt introduces the same unacceptable instruction address, a string of program interrupts is established. This string may be broken as before by IPL or system reset, or by the machine-check, external, or I/O interrupts. If these interrupts also have an unacceptable new PSW, new supervisor information must be introduced by the IPL or by manual intervention.

When the machine encounters the following conditions, the operation is suppressed, the 'end execute' signal is forced and the program interrupt is taken:

1. Invalid op code is detected (operation specification).
2. Specification exception is detected.
3. Invalid address is set (address specification).
4. Privileged operation exception in the problem state.
5. Fixed-point divide exception.
6. Floating-point divide exception.

Program Interrupt Backup

ACCELERATOR only: On a basic machine (GPR's in core storage) there is sufficient time during instruction execution for a program interrupt to be taken, and the normal execution of the next instruction is inhibited.

MSLS GPR's execute instructions much faster and in many cases the next I-fetch is started while the previous instruction is terminating. In a fixed-point overflow situation, the program interrupt is recognized near the end of the current instruction, and is too late to stop the next instruction. There is time, however, to 'freeze' the I-fetch operation and prevent CPU data from being altered. A 'backup for interrupt' latch, set by fixed-point overflow, performs this function and produces a dummy I-fetch with the following effects:

Incrementing of the instruction counter is inhibited.
No address indexing occurs.
Set or reset of the op register is inhibited.
'End execute' is forced at the end of I-fetch.

The signal 'end execute' then permits the program interrupt to be taken.

Supervisor-Call Interrupt

During I-fetch of the supervisor-call instruction the supervisor-call request latch is set. This request latch initiates the interrupt cycles and sets the supervisor-call interrupt latch if both the machine-check and the program interrupt latches are off. These latches are reset during the last period of the interrupt cycles. Thus, if a machine-check condition occurs during the I-fetch of the supervisor-call instruction, the machine-check interrupt is taken in priority over the supervisor-call interrupt. The supervisor-call interrupt latch is reset during the machine-check interrupt, and the supervisor-call interrupt is cancelled.

External Interrupt

On the basic system there are only two external interrupt signals: interrupt key from the console, and timer update. They can be extended, with the addition of six signal lines, to a special interface which may be utilized by any equipment capable of producing signals with a duration of 0.5 to 0.1 microseconds.

There is a request latch and an accept latch for each external-interrupt signal. An incoming signal sets its appropriate request latch which stores the request for an interrupt until the machine accepts it. The output lines from the interrupt request latches are OR'ed together to set the external-interrupt latch at the beginning of the cycle where the interrupt is taken. The external-interrupt request line is subject to the one-state of mask bit 7. Thus, the external-interrupt latch cannot be set if the mask bit is zero or if a higher priority interrupt latch is on.

Shortly after the external-interrupt cycles are started, a time is established when all pending requests are accepted. The accept latch is set for each request latch that is on at the time. The accept latches are used to set the interrupt code in PSW 1 bits 16 to 31 and they represent the external-interrupt requests that have been accepted.

Request latches are not immediately reset, to allow the maximum-length input signal to fall. This ensures that one signal does not set the request latch twice.

The request latches are reset on the fourth interrupt cycle when the corresponding accept latch is on. This means that only the accepted requests have their request latches reset. Also, requests that arrive at the request latch while it is on, will be ignored if they occur before the request latch is reset. This complies with the architecture of System/360.

External Machine Check

An external machine check has an interrupt of priority class four and comes after an external interrupt and before an I/O interrupt. An external machine check affects all I/O operations on the relevant chan-

nel. Errors causing an external machine check are the interface control check and the channel control check. The conditions leading to these checks are set out in the Principles of Operation — Channels, Form Y33-0003.

Detection of an external machine check condition causes all interface activity to be terminated at the next single-cycle point, and the channel to be denied further access to main storage. The CPU and other channels, if not in error, continue to operate and the external machine check remains pending.

Conditions are set up for the external machine check line to set the I/O machine check request latch when the interrupt request sample appears at 'end execute' time. Also, to define that this is an external check, bit 16 of the old PSW 1 interrupt code is set to one. The channel that caused the error is identified by a bit in position 23 for channel 2, a bit in position 22 for channel 1 and a bit in position 21 for channel 0. Zeros in bit positions 27 and 28 of the old PSW IC indicate an interface control check. Ones in these positions indicate that the interrupt was due to a channel control check.

The on states of the I/O machine check request latch combined with the set interrupt latches, set the machine check interrupt latch on. From this point the operation proceeds as for any machine check. The old PSW 1 carries the external machine check identity in bit position 16.

I/O Interrupts

The system of accepting I/O interrupts is in many ways similar to that used for external interrupts.

There is both a channel interrupt request and an accept latch for each of the three available channels. When an interrupt request is sent from the channel, the corresponding channel request latch is set. The outputs of these latches are active only if the corresponding mask bit is a one.

If the output of any one of the three channel request latches is active, a request for I/O interrupt is generated at the next 'end execute' signal.

When the interrupt cycles are started, the I/O interrupt latch is turned on, provided that no other class of interrupt request is present. This gives I/O interrupts the lowest priority of all the classes of interrupts.

During the first interrupt cycle, the machine accepts the channel interrupt request by turning on the corresponding accept latch. When there is more than one channel request latch on, only one request is accepted. This acceptance is determined by internal priority with the accept latches. The multiplex channel 0 accept latch cannot turn on if either of the HS multiplex channel request latches is on, and the HS multiplex channel 1 accept latch cannot turn on if the HS multiplex channel 2 request latch is on.

The internal priority is therefore:

1. High speed multiplexor channel 2
2. High speed multiplexor channel 1
3. Multiplexor channel 0

Near the completion of the interrupt cycles, a signal is developed which is sent to the channel to cause a reset of the channel interrupt request.

INTERRUPT CYCLES

- An interrupt routine consists of storing the current PSW as an old PSW, and fetching a new PSW.
- Fixed-address locations are provided for both the old and new PSW's.
- The interrupt codes are developed and stored as a part of the old PSW.

An interrupt cycle consists of storing the current PSW as an old PSW, and fetching the new PSW as a current PSW. To exchange the PSW's, the CPU takes four interrupt storage cycles.

The cause and class of the interrupt are identified, so that the machine and the programmer can distinguish between the types of interrupts. The cause of the interrupt is identified by setting a bit or bits in the interrupt code of the old PSW; the class is identified by addressing separate predetermined locations for the new PSW and the old PSW (Figure 3A-26). Figure 3A-24 shows the cause of interrupts within each class, the corresponding interrupt code and, where applicable, mask bits.

ADDRESS				Length	Purpose
Dec	Hex	Binary			
24	18	0001	1000	Double Word	External Old PSW
32	20	0010	0000	Double Word	Supervisor Call Old PSW
40	28	0010	1000	Double Word	Program Old PSW
48	30	0011	0000	Double Word	Machine Check Old PSW
56	38	0011	1000	Double Word	Input/Output Old PSW
64	40	0100	0000	Double Word	Channel Status Word
88	58	0101	1000	Double Word	External New PSW
96	60	0110	0000	Double Word	Supervisor Call New PSW
104	68	0110	1000	Double Word	Program New PSW
112	70	0111	0000	Double Word	Machine Check New PSW
120	78	0111	1000	Double Word	Input/Output New PSW

Figure 3A-26. PSW and CSW Storage Locations

Interrupts are taken after an instruction comes to 'end execute' and before the next instruction is started.

To exchange the PSW's, the CPU takes four interrupt storage cycles: new PSW 1, old PSW 1, new PSW 2 and old PSW 2, periods.

For the timing and sequence of these periods, see FEMD, Figures 6079/80 (basic) and 6293/94 (accelerator).

The class of the interrupt is defined by the setting of the appropriate interrupt class latch in the new PSW 1 period. One of the functions of these latches is to determine the addressing of the respective PSW fixed locations.

Cycle 1, New PSW 1 Period

On this cycle, the new PSW 1 is fetched from storage. The appropriate new PSW fixed address is set into the SAR's, and the new PSW 1 is read out from storage into the SDR. The A, B, C and BX registers are reset and the SDR is gated via the ABC funnel to the B register.

At the completion of this cycle therefore, the new PSW 1 is contained in the B register.

For the external and I/O interrupts, the request latches are copied to the accept latches early in this cycle. For the machine-check interrupt, CPU error conditions are also reset early in the cycle.

At this time the I/O interrupt halts the interrupt storage cycle in order to store the CSW. This is described later under "Storing the CSW." At the completion of the store cycles, the I/O interrupt proceeds with the old PSW 1 period in the same way as all other types of interrupt.

Cycle 2, Old PSW 1 Period

On this cycle, the old PSW 1 is stored into the old PSW fixed address location, and the new PSW 1 fetched in the previous cycle is set into the PSW 1 register.

The old PSW 1 is stored via the SDR. Bits 0 to 15 of the SDR are set from bits 0 to 15 of the current PSW 1; bits 16 to 29 are not set except in the case of an I/O interrupt; bits 24 to 31 are the interrupt code and are set depending upon the cause of the interrupt (the pattern of these bits is listed in Figure 3A-24). The contents of the SDR are then stored into the old PSW 1 location, as described later under "Developing the Interrupt Code."

On the later portion of this cycle, the B register is gated to the PSW 1 register.

At the completion of this cycle, the current PSW 1 has been stored as an old PSW with the new interrupt code in the old PSW 1 location, and the new PSW 1 has been fetched and stored in the current PSW 1 register.

Cycle 3, New PSW 2 Period

On this cycle, the new PSW 2 is fetched and preparation is made to store the old PSW 2.

The new PSW 2 is fetched from the fixed address location defined by the type of interrupt, and set into the SDR. It is gated from there to the B register and placed in the BX register by a B to BX interchange. During this cycle, the current PSW 2 is gated via the HW funnel to the ABC funnel and to the A register.

Thus, at the completion of this cycle, the new PSW 2 is contained in the BX register, and the old PSW 2 is contained in the A register.

Cycle 4, Old PSW 2 Period

On this cycle, the old PSW 2 is stored and the new PSW 2 is set to the current PSW 2 register. The old PSW 2 is fed from the A register, via the ABC funnel and the B register, to the SDR, and stored in the fixed address location. During this cycle, the IC (PSW 2 bits 8 to 31) of the current PSW 2 is set from the BX register. After the B register has been set to the SDR, the B and BX registers are interchanged. PSW 2 bits 2 to 7 of the current PSW are then set from the B register.

Bits 0 and 1 of PSW 2 are the Instruction Length Code (ILC) and remain unchanged.

During this period the external and I/O request latches are reset if the corresponding accept latch is on. The accept latches are then reset.

Summary of Interrupt Cycles

At the completion of these cycles, the current PSW has been stored as an old PSW, and the new PSW has been fetched and set as the current PSW.

The old PSW contains the ILC of the instruction last fetched and the address of the next instruction that would have been executed if an interrupt had not occurred.

The ILC is a network originating from the op register. This register does not reset until the next I-fetch, and during the interrupt cycle, the op register contains the op code of the last instruction fetched before the interrupt. Thus, the ILC remains unchanged until the next I-fetch is taken.

I/O Interrupts

During the first interrupt cycle, the new PSW 1 period, the wait for I/O latch is turned on if the interrupt class is an I/O interrupt. This latch inhibits the turning on of IRPT cycle control, and signals to the appropriate channel (in conjunction with the accept latch) that the storing of the CSW is to proceed.

Storing the CSW

- Four C-cycles are taken to store the CSW.
- The CSW is stored into the fixed storage locations 40 to 47 hex.
- If the subchannel is busy, the whole of the CSW is stored.
- If the subchannel is free, only the status portion of the CSW is stored.
- If the CSW information is available in the channel area, it is stored from this area.
- If the CSW information is stacked in the device, the device is selected to obtain this information.

When the accept interrupt signal is sent to the channel, it starts to re-select the device in order to fetch the status. (The device is not re-selected for the high speed multiplexor channel if the subchannel is busy.) This selection is similar to initial selection during a start I/O operation. Four C-cycles are needed to store the CSW.

For multiplexor channel 0, the information is taken from the selected subchannel in the extension storage and loaded back into the main storage. The status portion, however, is loaded from the channel data register.

For the high speed multiplexor channels, the information is loaded direct from the registers for the selected subchannel. For further details refer to the descriptions of "CSW Storing" in the multiplexor channel 0 and high speed multiplexor channel sections in the Principles of Operation — Channels, Form Y33-0003.

Developing the Interrupt Code

During the old PSW 1 period of exchanging the PSW's, the interrupt code is developed and set into the old PSW 1.

Machine-Check Interrupt

Bit 16 of the interrupt code is set directly from 'I/O error' which identifies the machine check as external in origin. The on state of bit 16 also lowers the priority of the interrupt. Bits 21, 22 and 23 are set from the channel check request latches. Bits 27 and 28, which signify the type of error, are set from the exception latches. If the machine has more than one check latch on at the same time, both bits may be set at the same cycle.

Program Interrupt

The program-interrupt code is set from the exception latches.

As it is possible to have more than one exception latch on, the exception latch with the highest internal priority is used to set the appropriate interrupt code. (See Figures 3A-24, 25.)

All exception latches are reset during the interrupt cycles so that the exceptions not identified in the interrupt code are not held pending.

Bits 16 to 27 of the interrupt code are always made zero.

Supervisor-Call Interrupt

Bits 24 to 31 of the interrupt code are obtained from the Ra and Rb registers, which are set from bits 8 to 15 of the instruction during the I-fetch.

Bits 16 to 23 of the interrupt code are made zero.

External Interrupt

The accept latches are used to set the external-interrupt codes, bits 24 to 31. If the latches are on, the corresponding bit positions are set to ones. If more than one accept latch is set, all the corresponding interrupt code bits are set at the same time on this interrupt cycle. (See Figure 3A-24.)

Bits 16 to 23 of the interrupt code are made zeros.

Input/Output Interrupt

Bits 21 to 23 and 24 to 31 of the I/O interrupt code are the channel address and the device address respectively, of the unit causing the interrupt. Bits 16 to 20 are made zeros.

Note that the condition in the device causing the I/O interrupt is identified in the status section of the CSW stored during the I/O interrupt.

TERMINATING THE INTERRUPT CYCLES

An interrupt is normally terminated by an 'end execute' signal in the same way as for instruction execution. The occurrence of a machine check during the interrupt cycles also terminates the interrupt by developing an 'end execute' signal, unless the machine-check occurs during a machine-check interrupt, in which case a hardstop occurs.

If there is another interrupt pending when the 'end execute' signal is developed, it is taken, otherwise a new instruction is fetched, using the IC field of the new PSW loaded during the interrupt cycles.

Special consideration must be given to the cancellation of an I/O interrupt, as there are some

conditions under which the interrupt is terminated prior to the time at which the CSW store cycles would normally occur.

Cancellation of an I/O Interrupt

- An interrupt is cancelled if the interface is working in burst mode at the time the channel tries to re-select the device.

- If the interrupt is cancelled, the interrupt latches in the channel and subchannel are reset.

- The I/O cancel interrupt signal forces the end-execute latch on; the CPU continues with the next sequential instruction.

If the interface is working in the burst mode when the channel or subchannel tries to re-select the device, the interrupt is cancelled (rather than hold the CPU waiting for an interrupt until the burst mode operation is completed).

During the first interrupt cycle the new PSW 1 was fetched from main storage and loaded into the B register. At that time, no information in either the old or current PSW was altered.

The I/O cancel interrupt line sets the end-execute latch, and the CPU continues with the next sequential instruction.

The I/O cancel interrupt line also resets the interrupt latches in the channel and subchannel. When the interface is next free, the device again sends a 'request in' and the channel-interrupt latch is again set.

CUT ALONG LINE

COMMENT SHEET

System/360, Model 44, Principles of Operation, Processing Unit
Field Engineering Theory of Operation Y33-0002

FROM

NAME _____ OFFICE/DEPT NO. _____

CITY/STATE _____ DATE _____

To make this manual more useful to you, we want your comments: what additional information should be included in the manual; what description or figure could be clarified; what subject requires more explanation; what presentation is particularly helpful to you; and so forth.

How do you rate this manual: Excellent ____ Good ____ Fair ____ Poor ____

Suggestions from IBM Employees giving specific solutions intended for award considerations should be submitted through the IBM Suggestion Plan.

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), STAPLE AND MAIL.

fold fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

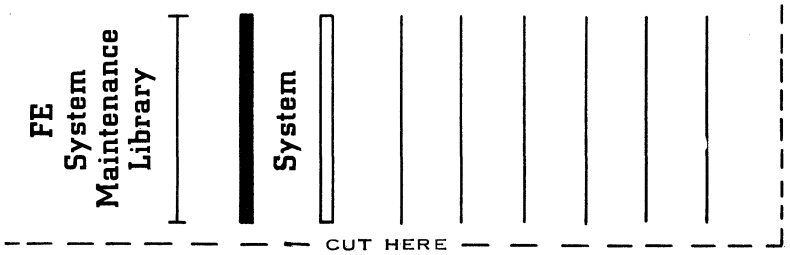
POSTAGE WILL BE PAID BY . . .

IBM CORPORATION

P.O. BOX 390
POUGHKEEPSIE, N.Y. 12602

ATTENTION: FE MANUALS, DEPT. B96

fold fold



Y33-0002-0

S/360 Model 44 Printed in U.S.A. Y33-0002-0

IBM

International Business Machines Corporation
Field Engineering Division
112 East Post Road, White Plains, N.Y. 10601