Diagram 5-1. Operand Prefetching During End Op
Diagram 5-2. Instruction Requests During End Op

Diagram 5-3. Instruction Requests During Early End Op

S-2.3 (7/70)
Diagram 5-4. Branch Requests
Diagram 5-5. Selection of I-Fetch Sequence
Diagram 5.7. One-Cycle RR I-Fetch

F

Clock
R1 (or R2 if Branch) to LAL
LS to T
T to PAA(32-63)
R to E
R2 (or R1 if Branch) to LAL
LS to S and T
PAL(32-63) to A and B
PAL(60-63) to D
Increment IC(21,22)
Q to E (Next Instruction)
Diagram 5-8. Two-Cycle RR-I-Fetch

- From SDBO
- A
- B
- C
- D
- E
- F
- G
- H
Diagram 5-10. One-Cycle RX, RS, and SI-I-Fetch
Diagram 5-11. Two-Cycle Indexed RX I-Fetch

- B2 Field to LAL
- LS to T
- T to E
- E to F
- F to PAA(2D-60) Only if B2 ≠ 0
- D2 Field to PAM(2D-60)
- X2 Field to LAL
- LS to T
- PAL(2D-60) to B
- Increment IC(21, 20)
- T to PAA(2D-60)
- B to PAM(2D-60)
- E1 Field to LAL
- PAL(4D-60) to D
- LS to S and T
- Q to R (Next Instruction)
Diagram 5-12. Two-Cycle Non-Indexed RX, BS, and SI I-Fetch
Diagram 5-14. SS I-Fetch (Sheet 1 of 2)
Diagram 5-15. I-Fetch Sequences (Sheet 1 of 2)
### Typical Micro-Orders

<table>
<thead>
<tr>
<th>Action</th>
<th>ALD</th>
<th>n-2</th>
<th>n-1</th>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set IF1 Trigger</td>
<td>K0101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Sequence 1 (See Note 1)</td>
<td>K0101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Sequence 2</td>
<td>K0111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Sequence 3</td>
<td>K0111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: X = Z = Not IF2, where Z is defined in Note 3.

### 1-Fetch Sequencers

<table>
<thead>
<tr>
<th>Action</th>
<th>ALD</th>
<th>n-2</th>
<th>n-1</th>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF Sequence 1 (See Note 3)</td>
<td>K0101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Sequence 2</td>
<td>K0111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Sequence 3</td>
<td>K0111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 1-Fetch Request Tgr</td>
<td>K0701</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IC + B → FAL</td>
<td>K0201</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FAL → IC</td>
<td>D6021</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS → G (See Note 2)</td>
<td>D6031</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 2-Fetch Sequencers

<table>
<thead>
<tr>
<th>Action</th>
<th>ALD</th>
<th>n-2</th>
<th>n-1</th>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set IF1 Trigger</td>
<td>K0101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Sequence 1 (See Note 3)</td>
<td>K0101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Sequence 2</td>
<td>K0111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Sequence 3</td>
<td>K0111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 1-Fetch Request Tgr</td>
<td>K0701</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IC + B → FAL</td>
<td>K0201</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FAL → IC</td>
<td>D6021</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS → G (See Note 2)</td>
<td>D6031</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Notes:
1. X = Z = Not IF2, where Z is defined in Note 3.
2. Also, gate QB-13 to RB-13 IF IC(21, 22) = 00 (ALD DB7F).
3. Z = (Not [Predicate branch] + Predicate 55 + Execute) + (IC(21, 22) + 10 + [Predicate not 85]).
4. These actions are inhibited by 'Blank 1-Fetch' trigger.

Diagram 5-15. 1-Fetch Sequencers (Sheet 2 of 2)
Diagram 5-16. Block 1-Fetch Trigger

- Start
- STAT O
- NEOP or BEOF
- Block 1-Fetch
- Exit

1. Block I-Fetch Sequences
2. Initial (P)Stop of I-Cycle
3. Block External Transfer
4. Block Storage Sequences During I-Fetch
5. Block PAL-n#7, I Transfer
6. Block Decoding of FI through PF Micro-orders
7. Block Initial Instruction Address Test

Explanations:
- NEOP = Not End of Program
- BEOF = Block End of Program
- STAT O = Status Register
- NEOP or BEOF = NEOP or BEOF
- Exit = Exit
- Manual Cell Reset Priority
- Machine Check Cell Priority
- Time Check Step Priority
- LPSM = Last Time in Progress
- Manual Cell Step Priority
- Supervisor Call Cell Priority
- Manual Cell Reset Priority
- Program Interrupt Type Off
- Program Store Compare Type
Diagram 5-17. Timer Exceptional Condition
Diagram 5-18. CPU Store In Progress Exceptional Condition

1. Inhibit Next I/O Address
2. Force ROSAR to 00H (hex)

a. 1st I/O Cycle
   - Force address to ROSAR.

b. CPU Store In Progress
   - Set $0000, if latch trigger.
   - Blocks not unlocked.

Diagram 5-18

7203-02 FEMDM (7/70) 5-18
Diagram 5-19. Machine Check Interruption
Diagram 5-23. Supervisor Call Interruption
Diagram 5-26. Common Interruption Routine

- Conditions at start of this routine:
  1. Address of old PSW is in D.
  2. Interruption code is in X'(6-31).
  3. STAT is not set if a machine check interruption is in progress.

A

B

C

D

E

F

G

• Conditions at start of this routine:
  1. Address of old PSW is in D.
  2. Interruption code is in X'(6-31).
  3. STAT is not set if a machine check interruption is in progress.

A

B

C

D

E

F

G

Diagram 5-26. Common Interruption Routine

- Conditions at start of this routine:
  1. Address of old PSW is in D.
  2. Interruption code is in X'(6-31).
  3. STAT is not set if a machine check interruption is in progress.
Diagram 5-27: Manual Control Exceptional Conditions
Diagram 5-28. Program Store Compare Exceptional Condition
Diagram 5-29. Invalid Instruction Address Test Exceptional Condition (Sheet 1 of 2)
Diagram 5-29. Invalid Instruction Address Test Exceptional Condition (Sheet 3 of 2)
Diagram 5-30: Test for Q-Register Refill Exceptional Condition
Diagram 5-106. Load Negative, LNR (11)

- **RR format:**
  - **Rr: Rr**
  - Purpose: Load 2nd operand (unchanged if negative, Z's complemented if positive) in GPR per R2 into 1st operand location (in GPR, per R1).
  - **Conditions at start of execution:**
    1. Instructions in E.
    2. 1st operand is in A, B, and D (not used).
    3. 2nd operand is in S and T.

- **Diagram:**
  - A: Q8001
  - B: Transfer 3rd operand from T to GPR per (T=11).
  - C: Transfer 3rd operand from T to GPR per (T=11).
  - D: Set CC to 0.
  - If T(32) = 1, set CC to 1.
  - End op.

---

---
Diagram 5-108. Fixed-Point Add/Type Instructions (Sheet 2 of 2)
Diagram 5-109. Fixed-Point Multiply (Sheet 1 of 3)
Diagram 5-109. Fixed-Point Multiply (Sheet 2 of 3)
Diagram 5-109. Fixed-Point Multiply (Sheet 3 of 3)
Diagram 5-10. Fixed-Point Divide (Sheet 1 of 6)

- Divide instruction (DIV) format: R0, R1, R2
- Purpose: Divide 1st operand (in GPR, per R1 and R2) by 2nd operand (in GPR, per R2) and place result into 1st operand location (remainder in GPR per R1); quotient in GPR per R1.
- Conditions on start of execution:
  1. Instruction is in T.
  2. High-order half of dividend (1st operand) is in A, B, and D.
  3. Divider (2nd operand) is in S and T.

A
- Prepare specification test and select dividend divisor select...
- G0001
- Specification test:
  - (11) = 1
  - Yes
  - No
- Diagram 5-22
- Program interruption.

B
- Data low-order dividend to 5 and high-order dividend to 0, in true form. Data divisor to T.
- Place 1's into A.
- Transfer GPR to T per (E1=1) = 1.
- Transfers lower-order bits of dividend from LS.
- Set STC to 000.
- 2's complement T and transfer to LS.
- Yes (Negative Dividend)
- No (Positive Dividend)
- STAT B or
- Transfer S to T.
- Places divisor into T.
- Transfer GPR to 5 per (E5=1) = 1.
- Transfers lower-order bits of dividend from LS to S.
- Set STAT O,
- Transfer LSWR to S.
- Transfer B to T.
- Places 2's complement of lower-order bits of dividend into S.
- Transfer 2's complement of T to B.
- Places 2's complement of higher-order bits of dividend into B.
- Transfer 2's complement of T and B.
- Places 2's complement of high-order bits of dividend into B.
- Transfer GPR to T per (E1=1).
- Places divisor into T.
- Add RAA and RAB, and transfer result to AB(24-07).
- DR (DI) Instruction Initialization

H

5-110, Sh 1  (7/70)
Diagram 5-110. Fixed-Point Divide (Sheet 4 of 6)
Establish proper sign and format for quotient and remainder, and store.

Yes (Negative Quotient) STAT C set

Addition to correct resultant remainder from last reduction cycle.

Sheet 6

Transfer 8 to PAA (32–60); Transfer 2's complement of T to PAA(23–60).

Add PAA and PAA, Transfer result to R.

Reset STAT C.

Yes (2's Complement Quotient) STAT C set

Sheet 6

Addition to correct resultant remainder from last reduction cycle.

Transfer 12(23–40) to PAA(32–60); Transfer 10(23–40) to PAA(32–60).

Add PAA and PAA, Transfer result to T.

Reset STAT C.

Yes (Positive Quotient) No (Positive Dividend)

2's complement T and transfer to R.

2's complements remainder to correct value.

No (Positive Dividend) STAT D set

Transfer S to T.

2's complement T.

Yes (Negative Quotient)

Transfer T to GPR per E[1]=1.

Store quotient.

Store remainder.

Termination. Quotient in 2's Complement Form

Set interruption code 7 and ‘program interrupt’ label.

1-cycle entry end up.

Diagram S-22

Program interruption.

Fixpoint divide check.

E[32] + 1


F

End op.
Diagram 5-110. Fixed-Point Divide (Sheet 6 of 6)
Diagram 5-111. Convert to Binary, C/V (4F) (Sheet 1 of 2)
Diagram 5-111. Convert to Binary, CVG (4F) (Sheet 2 of 2)
Diagram 5-112. Convert to Decimal, CVD (48)
Diagram S-9

RX hit.

A

Q801

Specification test.

No

PA(0,60) = 0

Yes

Set Instruction code 6 and "program interrupt" task.

force end op.

Set 2's complement of D to PAA(40-53) and 7 to PA(34-43).

Transfer IC to PA(40-43).

Diagram S-22

Program interruption.

Add PA(40) and PA(50), shift right 4 into PA(40).

Yes

PA(40-60) = 0

No

Yes

IC(7-22) = 11

No

Set "PIC" trigger.

Yes

PA(60) = 0

No

D

E

Set mark triggers 4-7 and gate operand into right half of main storage word.

Set mark triggers 0-3 and gate operand into left half of main storage word.

End op.

F

Diagram S-22

No

Yes

Protection check

Main storage accepts data.

Set Instruction code 4 and "program interrupt" task during 3-fetch of next instruction.

Program interruption.

Late protection program

Interrogation occurs after execution of next instruction.

G

H

Diagram S-113. Store, ST (50)
Diagram 5-114. Store Halfword, STH (40)
Diagram 5-155. Store Multiple, STM (90) (Sheet 1 of 2)
Diagram 5-117. Shift Left Double, SLDA (SF) (Sheet 1 of 4)
Diagram 5-117. Shift Left Double, SLDA (BF) (Sheet 2 of 4)
Diagram 5-117. Shift Left, Double, SLDA (8F) (Sheet 3 of 4)
Diagram 5-117. Shift Left Double, SLDA (8F) (Sheet 4 of 4)
Diagram 5-118, Shift Right Single, SRA (6A) (Sheet 1 of 3)
Diagram 5-118. Shift Right Single, SRA (8A) (Sheet 2 of 3)
Diagram 5-11B. Shift Right Single, SRA (IA) (Sheet 3 of 3)
Diagram 5-119. Shift Right Double, SRDA (HE) (Sheet 2 of 4)
Diagram 5-19, Shift Right Double, SRDA (SE) (Sheet 3 of 4)
Diagram S-119. Shift Right Double, SRDA (R8) (Sheet 4 of 4)
Diagram 5-201. Save Signs and Insert Sign Functions, and OC Setting
Diagram S-22
Program Interruption.

Diagram S-4
RR In-Order.

Diagram S-202. Load, LER (3b) - Short Operands; Load, LDR (2b) - Long Operands
Diagram 5-203. Load, LE (78) - Short Operands; Load, LD (68) - Long Operands
Diagram 5-205. Load Positive, LPDR (20); Load Negative, LNDR (21); Load and Test, LTDR (22); Load Complement, LCDR (23) - Long Operands
Diagram 5-206. Floating-Point Add, Subtract, and Compare - Short Operands (Sheet 2 of 5)
Diagram 5-206. Floating Point Add, Subtract, and Compare - Short Operands (Sheet 3 of 5)
Diagram 5-206. Floating-Point Add, Subtract, and Compare - Short Operands (Sheet 4 of 5)
Diagram 5-207. Floating-Point Add, Subtract, and Compare - Long Operands (Sheet 1 of 5)
Diagram 5-207. Floating-Point Add, Subtract, and Compare - Long Operands (Sheet 2 of 3)
Diagram 5-207. Floating-Point Add, Subtract, and Compare - Long Operands (Sheet 3 of 5)
Diagram S-207. Floating-Point Add, Subtract, and Compare - Long Operands (Sheet 4 of 5)
Diagram 5-207. Floating-Point Add, Subtract, and Compare - Long Operands (Sheet 5 of 5)
Diagram 5-20R. Halve, HER (34) - Short Operands
Diagram 5-209. Hahn, HDR (24) - Long Operands
A. Sign and Characteristic Data Paths.

B. Multiplication Data Path.

C. Fraction Data Path.

D. Fraction Data Path.

Notes:
1. In 2865 floating-point multiply operations, rules of 1st and 2nd operands are reversed from rules defined in System/600 Principles of Operation, SEL, Form A22-0221-6. That is, 2nd operand is multiplied and 1st operand is multiplier. (Interchanging operand order does not change product. Read, however, still replaces 1st operand.)

2. For an EX instruction with normalized 1st operand, the 1st operand characteristic and sign are in S or T and the 2nd operand characteristic and sign are in A or B.

Diagram 5-210. Floating-Point Multiply Data Paths
Diagram 5-211. Floating-Point Multiply, Short Operands (Sheet 1 of 4)
Diagram 5-211 Floating-Point Multiply, Short Operands (Sheet 4 of 4)
Diagram S-212. Floating-Point Multiply, Long Operands (Sheet 2 of 4)
Diagram S-212. Floating-Point Multiply, Long Operands (Sheet 4 of 4)
Diagram 5-214. Floating-Point Divide, Short Operands (Sheet 1 of 4)
Diagram 5-214. Floating-Point Divide, Short Operands (Sheet 3 of 4)
Diagram 5-214. Floating-Point Divide, Short Operands (Sheet 4 of 4)
Diagram 5-215. Floating-Point Divide, Long Operands (Sheet 1 of 3)
Diagram 5-215. Floating-Point Divide, Long Operands (Sheet 2 of 5)
Diagram 5-216. Store, STE (70) — Short Operands; Store, STD (60) — Long Operands
Diagram 3-302: True Add Sequence for Decimal Add, Subtract, and Compare (Sheet 2 of 3)
Diagram 5-102. True Add Sequence for Decimal Add, Subtract, and Compare (Sheet 3 of 3)
Diagram 5-303. Complement Add Sequence for Decimal Add, Subtract, and Compare (Sheet 1 of 3)
Diagram 5-103. Complement Add Sequence for Decimal Add, Subtract, and Compare (Sheet 2 of 3)
Diagram S-304. Zero and Add (Sheet 1 of 4)

Note: See Diagram S-302 for general data path.
A) Overall Flow Chart

B) General Data Path

Diagram 5-305. Decimal Multiply (Sheet 1 of 7)
MULTIPLIER LEFT-ADJUST SEQUENCE (G30M)
Objective:
Fetch full multiplier from storage and transfer to L2. High-order multiplier digit must occupy leftmost byte (byte 0) in L2.

A
Transfer instruction address from 0 to PAL, and hold.

During left-adjust sequence, instruction address is held in PAL by "MOS" accumulator.

B
Gate L2 from 0 to STC.

STC is set to where the rightmost byte of multiplier will be placed after left adjustment.

C
Move multiplier sign byte from A8 to STC.

D
Transfer performed via serial adder.

E
Set STC C-0 multiplier sign is minus. Set STC E-0 units digit is canceled.

F

G

Add 1 to E03-15.

Set STC to H11.

Gate E0-7 to E02-15 (see parallel adder).

Gate L2 from PAL to E03-15.

H

MULTIPLIER RIGHT-4 SHIFT TO DIP (SIGN) (G20M)
Objective:
Shift multiplier right 4 to eliminate sign and transfer to A.

Add 1 to E04-07 (see parallel adder).

Hold PAL to move last high-order multiplier digit.

Transfer last high-order multiplier digit from PAL (A0-07) to A0-07.

B04-07 serves as temporary storage during right-4 shift.

G04-07 or PMB04-07 and shift right 4.

G04-07 or PMB04-07 and shift right 4.

Transfer shifted result from PAL to B.

1. Add-011 contains high-order multiplier digit
2. Excluding last digit.

L2 must be restored to E03-15 before entering the multiply add or address loop.

L2 in E03-15 was reduced 3 before zero. It is now restored to zero.

To perform correct getting of L2 from E to 0.

L2 RESTORATION (G20M)
Objective:
S

Diagram 5-305. Decimal Multiply (Sheet 3 of 7)
Diagram 3-305. Decimal Multiply (Sheet 4 of 7)
Diagram 5-105. Decimal Multiply (Sheet 5 of 7)
Diagram 5-305. Decimal Multiply (Sheet 6 of 7)
Diagram 5-305: Decimal Multiply (Sheet 7 of 7)
A) Overall Flow Chart

B) General Data Path

Diagram 5-306. Decimal Divide (Sheet 1 of 9)
Diagram 5-14: SS-IFetch

G5 (Q153)
AND Q5041)

Objectives:
1. Move divisor into AB and set ARC to the low-order dividend byte. Save L1 and L2 in F.
2. Test dividend signs for validity.

Diagram 5-15

Gate 1st operand (dividend) from SD0 to ST.

Diagram 5-16

Add L3 to IC and request 2nd operand (divisor). Transfer D to STC.

Diagram 5-17

Transfer L1 and L2 from (IC-10) to (IC-9).

Reduce IC by 8 to address next double-word of divisor.

Diagram 5-18

Gate divisor from SD0 to AB. Set STAT 2 if dividend sign is negative; set STAT 1 if minus.

Diagram 5-19

Subtract L1 from D to set address of high-order byte. Reset STAT F.

Diagram 5-20

Gate L2 from E to STC.

Diagram 5-21

STC set to where highest divisor byte is placed after left adjustment.

Diagram 5-22

Set STAT D to record that divisor is 5 bytes or greater.

Diagram 5-23

Find next instruction address from LWRE to 5.

Diagram 5-24

Sheet 3

---

5-306, Sh 2 (7/70)
Diagram 5-306. Decimal Divide (Sheet 3 of 9)
Diagram 5-306. Decimal Divide (Sheet 4 of 9)
Diagram 5-306. Decimal Divide (Sheet 9 of 9)
### Diagram 5-14

**SS Format:**

<table>
<thead>
<tr>
<th>Op Code</th>
<th>L1</th>
<th>L2</th>
<th>B1</th>
<th>D0</th>
<th>D1</th>
<th>B2</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Op Code:**
1. Pack (PACK = F)
2. Unpack (UNPK = F)
3. Move With Offset (MOV = F)

**Purpose:**
1. Pack - Convert format of 2nd operand (in storage) from word to packed and place result into 1st operand location (in storage).
2. Unpack - Convert format of 2nd operand (in storage) from packed to word and place result into 1st operand location (in storage).
3. Move With Offset - Store 2nd operand (in storage) to left of and adjacent to low-order 4 bits of 1st operand (in storage).

**Conditions at end of Pack:**
1. Main storage request for doubleword containing low-order byte of 1st operand (destination) has been issued per D.
2. D contains low-order byte address (contents of GPR addressed by B1, B0 + D1 + D0) of 1st operand.
3. IC contains high-order byte address (contents of GPR addressed by B7, B0) of 2nd operand.

### Diagram 5-307. GES for Pack, Unpack, and Move With Offset
Diagram 5-30B. Pack, Not Word Overlap Sequence
Diagram 5-311. Unpack, Word Overlap Sequence
Diagram 5-312: Move With Offset, Not Word Overlap Sequence
Diagram 5-14

(All other SI
Logical Instructions)

GOTO

DIM

15

2

A

Transfer C to "PARA=62"

Transfer completion of D to "PARA=62"
and wait for C.

转移完成

of
D
and
wait
for
C.

SDF
wait right 4.

Test overlap
for "PARA=62"

Transfer D to STC

Transfer C to "PARA=62"

Transfer D to "PARA=62"
and continue.

SDF
wait right 4.

Test overlap
for "PARA=62"

Transfer C to "PARA=62"

Transfer D to "PARA=62"
and complete.

Shift wait right 4.

Test overlap
for "PARA=62"

Transfer C to "PARA=62"

Transfer D to "PARA=62"
and complete.

Shift wait right 4.

Test overlap
for "PARA=62"

Transfer C to "PARA=62"

Transfer D to "PARA=62"
and complete.

Shift wait right 4.

Test overlap
for "PARA=62"

Transfer C to "PARA=62"

Transfer D to "PARA=62"
and complete.

F

branch "non-execution
sequence per (E+5)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"

branch "non-execution
sequence per (E+7)"
Diagram 5-402. Logical Move Instructions

- **A. Move, MV (R2)**
  - **S1 format.**
  - Purpose: Place immediate operand (S2 of instruction) into 1st operand location (in storage).

- **B. Move, MVC (R2)**
  - **S2 format.**
  - Purpose: Place 2nd operand (in storage) into 1st operand address (in storage).
  - Fetch 1st number of bytes from source per 1st operand address.
  - Store into destination per 1st operand address.

- **C. Move, MVE (R2)**
  - **S2 format.**
  - Purpose: Place zone portion (high-order 4 bits) of each byte of 2nd operand (in storage) into high-order 4 bits of corresponding byte of 1st operand (in storage).

- **D. Move Zones, MVZ (R2)**
  - **S2 format.**
  - Purpose: Place zone portion (high-order 4 bits) of each byte of 2nd operand (in storage) into high-order 4 bits of corresponding byte of 1st operand (in storage).

Diagram 5-403. Logical Compare Instructions

- **A. Compare Logical, CLR (R2)**
  - **S1 format.**
  - Purpose: Binarly compare 1st operand (in DRF per R1) with 2nd operand (in DRF per R2) and set CC according to result.

- **B. Compare Logical, CIL (R2)**
  - **S2 format.**
  - Purpose: Binarly compare 1st operand (in DRF, per R1) with 2nd operand (in storage) and set CC according to result.
Diagram 5-404. Logical AND Instructions

A. AND, R (14)
- RR format.
- Purpose: AND 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and place result into 1st operand location.
- Fetch 1st operand from GPR per R1.
- Fetch 2nd operand from GPR per R2.
- AND.
- Store result into GPR per R1 and set CC.

B. AND, N (54)
- RX format.
- Purpose: AND 1st operand (in GPR, per R1) with 2nd operand (in storage) and place result into 1st operand location.
- Fetch 1st operand from GPR per R1.
- Fetch 2nd operand from main storage.
- AND.
- Store result into GPR per R1 and set CC.

Diagram 5-405. Logical OR Instructions

A. OR, R (16)
- RR format.
- Purpose: OR 1st operand (in GPR, per R1) with 2nd operand (in GPR, per R2) and place result into 1st operand location.
- Fetch 1st operand from GPR per R1.
- Fetch 2nd operand from GPR per R2.
- OR.
- Store result into GPR per R1 and set CC.

B. OR, O (56)
- RX format.
- Purpose: OR 1st operand (in GPR, per R1) with 2nd operand (in storage) and place result into 1st operand location.
- Fetch 1st operand from GPR per R1.
- Fetch 2nd operand from main storage.
- OR.
- Store result into GPR per R1 and set CC.

C. OR, N (56)
- SX format.
- Purpose: OR immediate operand (32 of instructions) with 1st operand (in storage) and place result into 1st operand location.
- Obtained immediate operand from E.
- Fetch 1st operand from main storage.
- OR.
- Store result into main storage per 1st operand address and set CC.

D. OR, OC (56)
- SX format.
- Purpose: OR immediate operand (32 of instructions) with 1st operand (in storage) and place result into 1st operand location.
- Fetch 1st operand from main storage.
- OR.
- Store result into main storage per 1st operand address and set CC.
- **SS format.**
  - **Problem:** Add 1st operand byte (argument) to 2nd operand address, use result as storage address, and place function byte from resulting storage address into corresponding 1st operand byte location.
  - **Steps:**
    1. **Select argument byte.**
    2. **Add argument byte to base address of function byte (2nd operand address).**
    3. **Fetch function byte per result address.**
    4. **Store function byte into argument byte location.**

**Diagram 5-410. Translate, TR (DC) and TRT (DD)**

---

- **SS format.**
  - **Problem:** Add 1st operand byte (argument) to 2nd operand address, use result as storage address, and place function byte from resulting storage address into corresponding 1st operand byte location. If zero, translate and load next argument byte; if nonzero, complete operation by inserting related argument address into GM1 and function byte into GM2.
  - **Steps:**
    1. **Select argument byte.**
    2. **Add argument byte to base address of function byte (2nd operand address).**
    3. **Fetch function byte per result address.**
    4. **Store function byte into GM2.**

**Diagram 5-410. Translate and Test, TRT (DD)**
A. Shift Left Single, SLL (89)

B. Shift Left Double, SLD (80)

C. Shift Right Single, SRL (88)

D. Shift Right Double, SSDL (BC)

Diagram 5-412. Logical Shift Instructions
Diagram 5-501. Branch On Condition, BCR (07): BC (47) (Sheet 1 of 2)
Diagram 5-501. Branch On Condition, BCR (07); BC (47) (Sheet 2 of 2)
Diagram 5-502. Branch and Link, BALR (05) (Sheet 1 of 2)
Diagram 5-9

Determine address of instruction (link address) following BAL instruction.

If set, indicates that the branch instruction is the subject instruction of an Execute Instruction.

If yes, Execute Instruction is located in the 3rd and 4th halfword.

If yes, branch instruction being executed was located in 3rd and 4th halfword of Q.

ABC ≠ 0

Store link data into Q, subroutine address of next sequential dualword following the Execute Instruction, refill Q, and transfer Q to 8 per DQ[1,20].

Store link data into IC, enable IC (register IC), refill Q, and gate Q to 8 per DQ[1,20].

DQ[1,20] = 11

Update IC and refill Q.

Normal end op.

Diagram 5-503. Branch and Link, BAL (45) (Sheet 1 of 2)
Diagram S-503. Branch and Link, BAL (45) (Sheet 2 of 2)
Diagram 5-54: Branch On Count, BCTR (05); BCT (46) (Sheet 1 of 2)
Diagram 5-505. Branch on Index High, BXLH (86); Branch on Index Low or Equal, BXLE (87) (Sheet 2 of 5)
Diagram 5-505. Branch on Index High, BXH (86); Branch on Index Low or Equal, BXLE (87) (Sheet 3 of 3)
Diagram 5-596. Execute, EK (Sheet 1 of 2)
Diagram 5-506. Execute, EX (44) (Sheet 2 of 2)
Diagram 5-6

A

Q+X01

164

Transfer 8 to T and O via parallel enter.

B

Issue early end-up
2 cycles early.

Transfer [34-39] to PSW-register.

Sets new CC and
program mask into
current PSW.

RI: I-Fetch.

Normal end-up.

Diagram 5-602. Set Program Mask, SPM (04)

- **RI Format:**
  - **04**
  - **01**

- Replace CC and program mask (bits 34-39) of current PSW with bits 2-7 of 1st operand (in GPW, per R1).

- Conditions at start of execution:
  1. Instruction is in E.
  2. 1st operand is in A, B, and D.
  3. 2nd operand is not used.

- Bits 2-7 of 1st operand may have been loaded from PSW-register by a previous Branch and Link instruction.

- Program mask format (set mask bit permits interruption):
  - Bit 36 = Floating-point overflow mask.
  - Bit 37 = Directed overflow mask.
  - Bit 38 = Excess underflow (floating-point)
    mask.
  - Bit 39 = Significance (floating-point) mask.
Diagram 5-13

SI = fetch

A

Q,631

DIAG/1?

Problem
State

Yes

Diagram 5-22

Program
Interrupt

No

B

Set ABC to value of
(D01-23) via parallel
adder.

ABC will gate new system
mask from input doubllexword.

Gate SDRO to AB.

C

Gate first byte of new
system mask from AB
(then serial adder) into ST
(0-15).

D

ABC/7

Yes

Request next doubllexword
(0-15) from storage.

Gate SDRO into AB.

E

Gate second byte of new
system mask from AB
(then serial adder) into ST
(16-31).

F

Gate S (0-7) to PSW (0-7)
and S16-19 to PSW (16-19).

Set new system mask into PSW.

G

End Op

H

Diagram 5-603. Set System Mask, SSM (60)
A
Q.X331
IA
Set " supervisor call" trigger.

B
bus early end-op
1 cycle before normal end-op.

Diagram 5-6
IR I-fetch.

Diagram 5-23
Supervisor call interruption.

Diagram 5-404. Supervisor Call SVC (6A)

C

D

E

F

G

H

IR Format:

<table>
<thead>
<tr>
<th>2A</th>
<th>2F</th>
<th>1</th>
</tr>
</thead>
</table>

Causes supervisor call interruption; replace old-PSW(32-31) with Inf/old (bits 9-15) of instruction, providing interruption code.

Conditions at start of execution:
1. Instruction is in E.
2. 0(9-15) contains interruption code.
A

Diagram S-6

Initiate.

B

Decompress ABC and STC.

C

Transfer S to A via parallel adder. Transfer A minus S to D via parallel adder.

D

Privileged operation test.

E

Add S to D via parallel adder.

F

Set key into even address.

G

Set if key has been set for the last time.

H

Diagram S-65. Set Storage Key, SSK (03)
Diagram 5-66. Insert Storage Key, ISK (09)

- OR if less TSE=403 via serial adder.
- Add B to D via parallel adder, and issue 3-cycle storage request per D.
- Set 'Insert key' trigger and STAT D.
- 'Insert key' signal is sent to storage, and CE clock is stopped.
- Code 'key out' kick to POP, B=4, and restart CE clock.
- Fetch key from file address.
- Separation of previous operation mode necessary by default interface maintenance etc.
- Test if last fetch was made.
- Set STAT D.
- OR if less TSE=403 via serial adder.
- Fetch key twice more.
- Transfer T to GMR per ERI-111.
- TSE=403 contains old and event keys OR'ed together.
- Normal end-up.

Diagram 5-22. Program initiation.

- Fetch key from even address.
- OR if less TSE=403 via serial adder.
- Allows construction of STAT D loop.
- Add B to D via parallel adder, and issue 3-cycle storage request per D.
- Set 'Insert key' trigger and STAT D.
- 'Insert key' signal is sent to storage, and CE clock is stopped.
- Code 'key out' kick to POP, B=4, and restart CE clock.
- Fetch key from file address.
- Separation of previous operation mode necessary by default interface maintenance etc.
- Test if last fetch was made.
- Set STAT D.
- OR if less TSE=403 via serial adder.
- Fetch key twice more.
- Transfer T to GMR per ERI-111.
- TSE=403 contains old and event keys OR'ed together.
- Normal end-up.

- Insert storage protection key for 2048-byte storage block, addressed by bits 8-39 of 3rd operand (in GMR, per E92), into bits 36-39 of 1st operand (in GMR, per E91).
- Conditions of state of execution:
  1. Instruction is I, E, or.
  2. 1st operand is in A, B, and D.
  3. 2nd operand is in E and T.
  4. STC=0 and ABC=0.
- Bits 8-39 of 1st operand are not changed; storage protection key form block of onboard storage is inserted into bits 26-39; bits 29-31 are set to 0's.
- Bits 8-39 of 2nd operand designate which block of 2048 bytes in main storage is to have its key inserted into 1st operand.
- 'Insert key' trigger: ALD MAC 181; STAT 0, ALD 20301.
Diagram 5-607. Write Direct, WRD (84)
Diagram 5-608. Read Direct, RDD (85)
Diagram 5-609. Diagnose (83) (Sheet 3 of 3)
### Diagram S-701: I/O Instructions

**Objective:**
- Analyze the I/O instructions for the DEC PDP-11 computer
- Focus on the General register

**Objective D:**
- General I/O instructions
- Load and store I/O registers

**Objective C:**
- Load and store I/O registers

**Objective B:**
- Retrieve and modify I/O registers
- Load and store I/O registers

**Objective A:**
- Analyze the I/O instructions for the DEC PDP-11 computer
- Focus on the General register

**External Inputs:**
- 8-bit external register
- 8-bit external register

**External Outputs:**
- 8-bit external register
- 8-bit external register

**Notes:**
- The current I/O address is determined by examining the contents of I/O registers.
- The I/O address is then stored in an external register for control.

**Condition Code:**
- The condition code is set based on the I/O address.

**Instructions:**
- For each instruction, the condition code is determined by examining the control register.
- The condition code is then stored in an external register for control.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Condition Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1C</td>
<td>One</td>
</tr>
<tr>
<td>1D</td>
<td>T0</td>
</tr>
<tr>
<td>1E</td>
<td>H0</td>
</tr>
<tr>
<td>1F</td>
<td>I0</td>
</tr>
<tr>
<td>1G</td>
<td>SPDI</td>
</tr>
<tr>
<td>1H</td>
<td>Available</td>
</tr>
</tbody>
</table>

**References:**
- For a detailed list of instructions and I/O addresses, refer to the DEC PDP-11 manual.
Diagram 5-9

RR 1-1=eh

Diagram 5-801. Load Identity, LI (OC)

Objectives:
Core identity in F14-L7.
Enclave identity to f.
Core identity in a GPR (38-31).
Diagram 5-83: Delay, DLY (08)

- **Purpose:** Provides a variable delay (256 uses x N1) dependent on the value of N1.
- **Conditions at the beginning of execution:** Instruction and N are in E.

### Diagram Details

- **Step 1 and Step 2:** Microprocessor step CE clock for 3 cycles.
- **Delay 1 uses:**
  - Add 1 to F register.
  - Check for pending interrupt.
  - If yes, follow the next step; if no, check if value equals 256.
    - If yes, subtract 1 from E16-15.
    - If no, timer update.
- **Timer update:**
  - Yes: STAT A on.
- **Save count in K register:**
- **Turn on STAT A:**
- **Execute timer update microprogram End Op.:**
- **End Op.:**
Diagram S-805. Load PSBAR, LPSB (A1)
Diagram 5-806. Move Word, MVW (D8) (Sheet 1 of 3)
Diagram S-806. Move Word, MYW (DB) (Sheet 2 of 3)
Diagram 5-807.  Start I/O Processor, SIOP (9A)
Diagram 5-80. Set Address Translator, SATR (0D), Execution in Issuing CE (Sheet 1 of 6)
Diagram 5-408. Set Address Translator, SATR (0D), Execution in Issuing CE (Sheet 2 of 6)
Diagram 5-808. Set Address Translator, SATR (ID), Execution in issuing CE (Sheet 3 of 6)
Diagram 5-808. Set Address Translator, SATR (SD); Execution in Issuing CE (Sheet 4 of 6)
Diagram 5-808. Set Address Translator, SATR (0D), Execution in Issuing CE (Sheet 5 of 6)
Diagram 5-80B: Set Address Transistor, SATR (0D), Execution in Issuing CE (Sheet 6 of 6)
Diagram 5-809. Set Address Translator, SATR (0D), Execution in Receiving CE (Sheet 1 of 3)
Diagram S-809. Set Address Translator, SATR (00), Execution in Receiving CE (Sheet 2 of 3)
Diagram 5-409. Set Address Translator, SATR (ID), Execution in Receiving CE (Sheet 3 of 3)
A
- **Purpose:** To transfer configuration data to CCR of element(s) specified by selection mask.
- **Conditions at the end of 1-fetch:**
  1. A-reg has 1st word of configuration mask (contents of R1).
  2. T-reg has selection mask (contents of OR specified by R2).
- **Configuration and selection mask formats:**
  1. 9S0200 System

![Diagram of configuration and selection mask formats for 9S0200 System]

B

C
2. 9S030 System

![Diagram of configuration and selection mask formats for 9S030 System]

D

![Diagram 5-6: 9S 1-fetch]

E
- **Objectives:**
  - Sense T-reg
  - Sense external reg.
- **Legend:**
  - Square Bit
  - Reserved Bit
  - Initial Value
  - Initial Value Step Bit
  - Inhibit OE Step Bit

F

![Diagram 6-8: Sense T-reg]

G
- **Diagram 5-810: Set Configuration, SCON (01) (Sheet 1 of 6)**
Diagram 5-810. Set Configuration, SCON (01) (Sheet 2 of 6)
Diagram S-810. Set Configuration, SCON (01) (Sheet 3 of 6)
Diagram 5-810. Set Configuration, SCON (01) (Sheet 5 of 6)
Diagram 5-11. Test and Set, TS (93)
Diagram 5-901. Regen Bars, Simplified Flow Chart
Diagram 5-902. Repack Symbols, RPSB (IF) (Sheet 3 of 21)
Diagram 5-902. Repack Symbols, RPSB (9F) (Sheet 4 of 21)
Diagram 5-902. Repack Symbols, RPSR (OP) (Sheet 5 of 23)
Diagram S-902. Repack Symbols, RPSB (OF) (Sheet 6 of 21)
Diagram 5-902: Repack Symbols, RPSB OF (Sheet 10 of 21)

* 2-letter designations identify actual line/edge connection on CAS page.
Diagram 5-902. Repack Symbols, RPSB (OF) (Sheet 12 of 21)
Diagram 5-902. Repack Symbols, RPSB (OF) (Sheet 14 of 21)
Diagram 5-902. Repack Symbols, RPSB (OP) (Sheet 15 of 21)
Diagram 5-002. Repack Symbols, RPSB (OF) (Sheet 16 of 21)
Diagram 5-902. Repack Symbols, RPISB (OF) (Sheet 17 of 21)
Diagram 5-902. Repack Symbols, RPSB (OF) (Sheet 19 of 21)
Diagram 5-902. Repack Symbols, RPSB (OF) (Sheet 21 of 21)
Diagram S-903. Convert and Sort Symbols, CSS (02) (Sheet 2 of 10)
Diagram 5-903. Convert and Sort Symbols, CSS (02) (Sheet 3 of 10)
Diagram 5-903. Convert and Sort Symbols, CSS (02) (Sheet 4 of 10)
Diagram 5-903. Convert and Sort Symbols, CSS (02) (Sheet 7 of 10)
Diagram 5.904. Convert Weather Lines, Simplified Flowchart
Diagram S-905. Convert Weather Lines, CVWL (03) (Sheet 1 of 9)
Diagram 5-905. Convert Weather Lines, CVWL (03) (Sheet 2 of 9)
Diagram S-905. Convert Weather Lines, CVWL (03) (Sheet 3 of 9)
Multiply routine, used by both CVWL and CSS.

Objective:
Multiply both X and Y coordinates by the conversion constant.

AND (0A-47) with F(0-1) and the serial register.

Checking scaling constant 5/bits 24-25 to determine which scale.

Conversion Complete

Scale Multiplier

Global Local

ML MG M0

A

B

C

D

E

F

G

H

Diagram 5-905: Convert Weather Lines, CVWL (03) (Sheet 6 of 9)
Diagram S-905. Convert Weather Lines, CVWL (03) (Sheet 7 of 9)
Diagram S-905. Convert Weather Laces, CVWL (03) (Sheet 9 of 9)