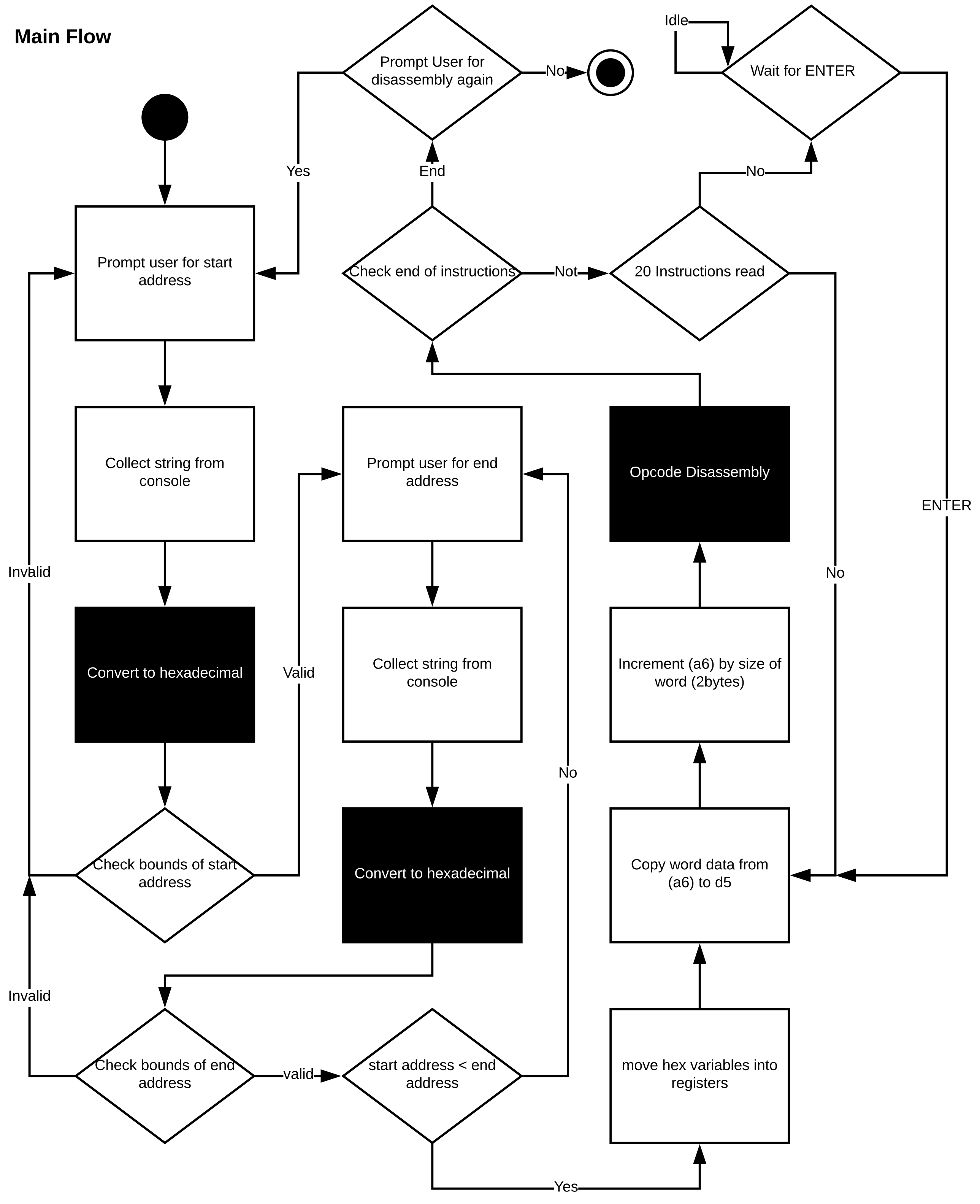
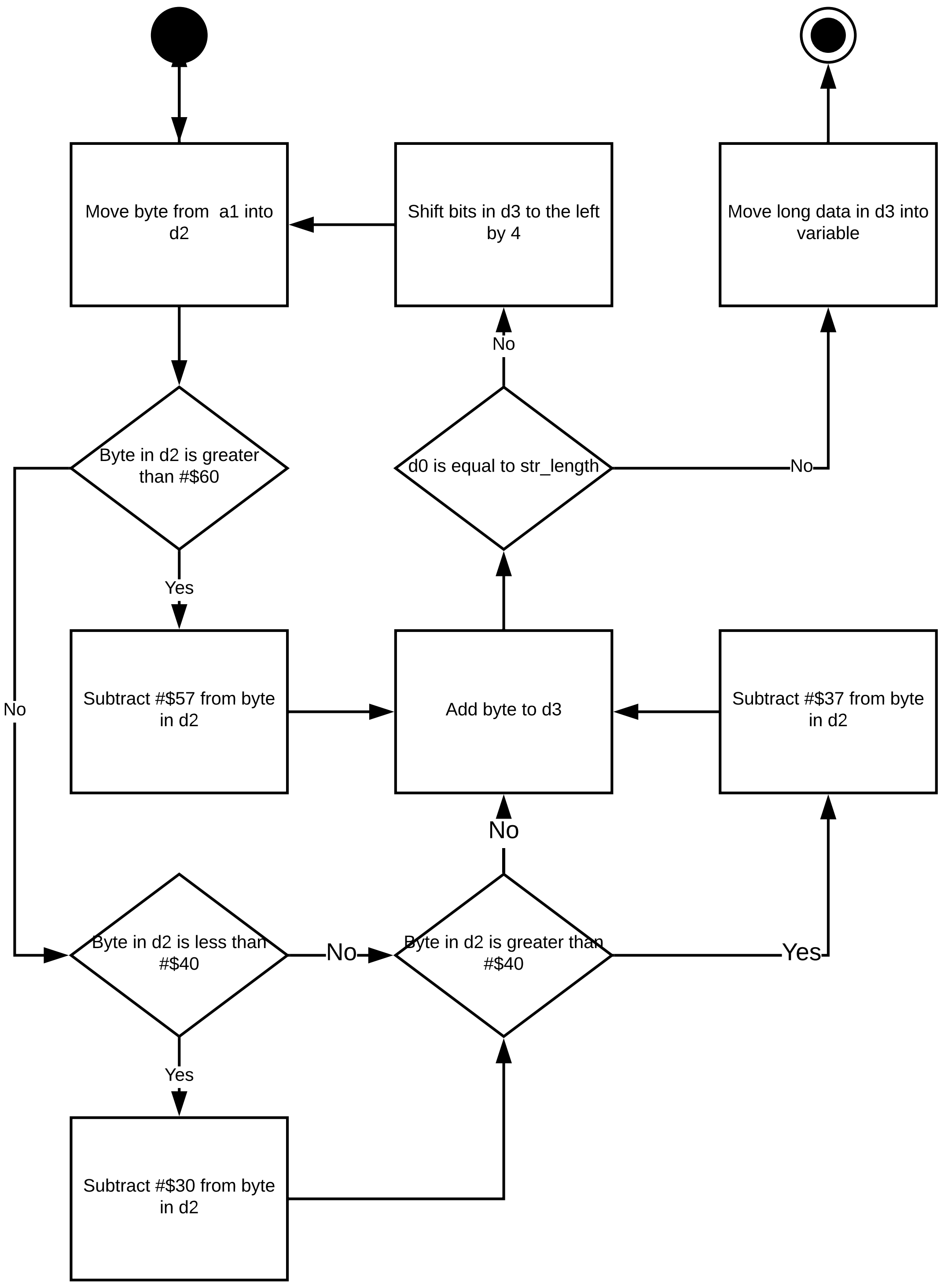
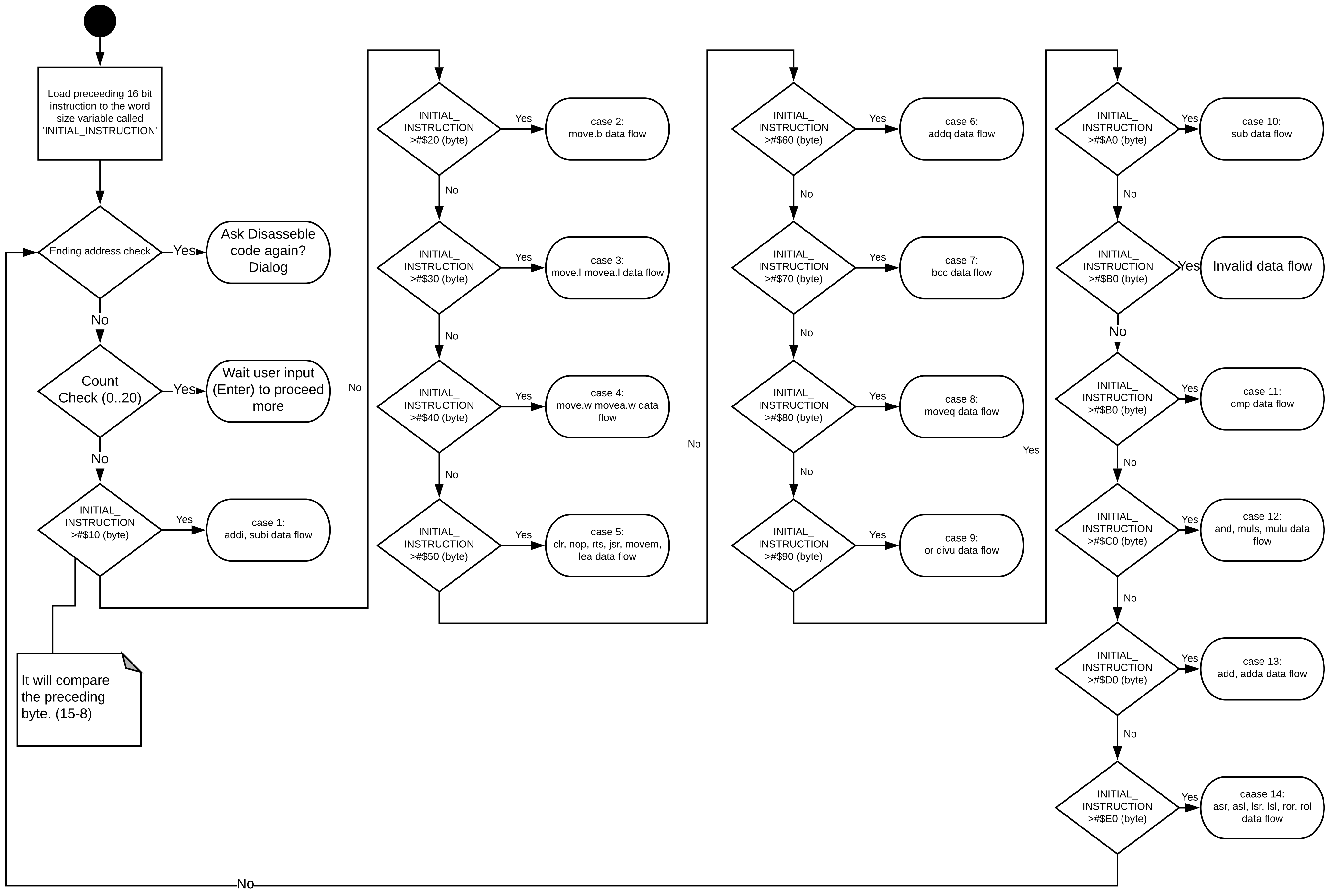
Main Flow


Prompt User for disassembly again
No
Idle
Wait for ENTER
Yes
End
No
Prompt user for start address
Check end of instructions
Not
20 Instructions read
Collect string from console
Prompt user for end address
Opcode Disassembly
ENTER
Invalid
No
Convert to hexadecimal
Valid
Collect string from console
Increment (a6) by size of word (2bytes)
No
Check bounds of start address
Convert to hexadecimal
Copy word data from (a6) to d5
Invalid
Check bounds of end address
valid
start address < end address
move hex variables into registers
Yes

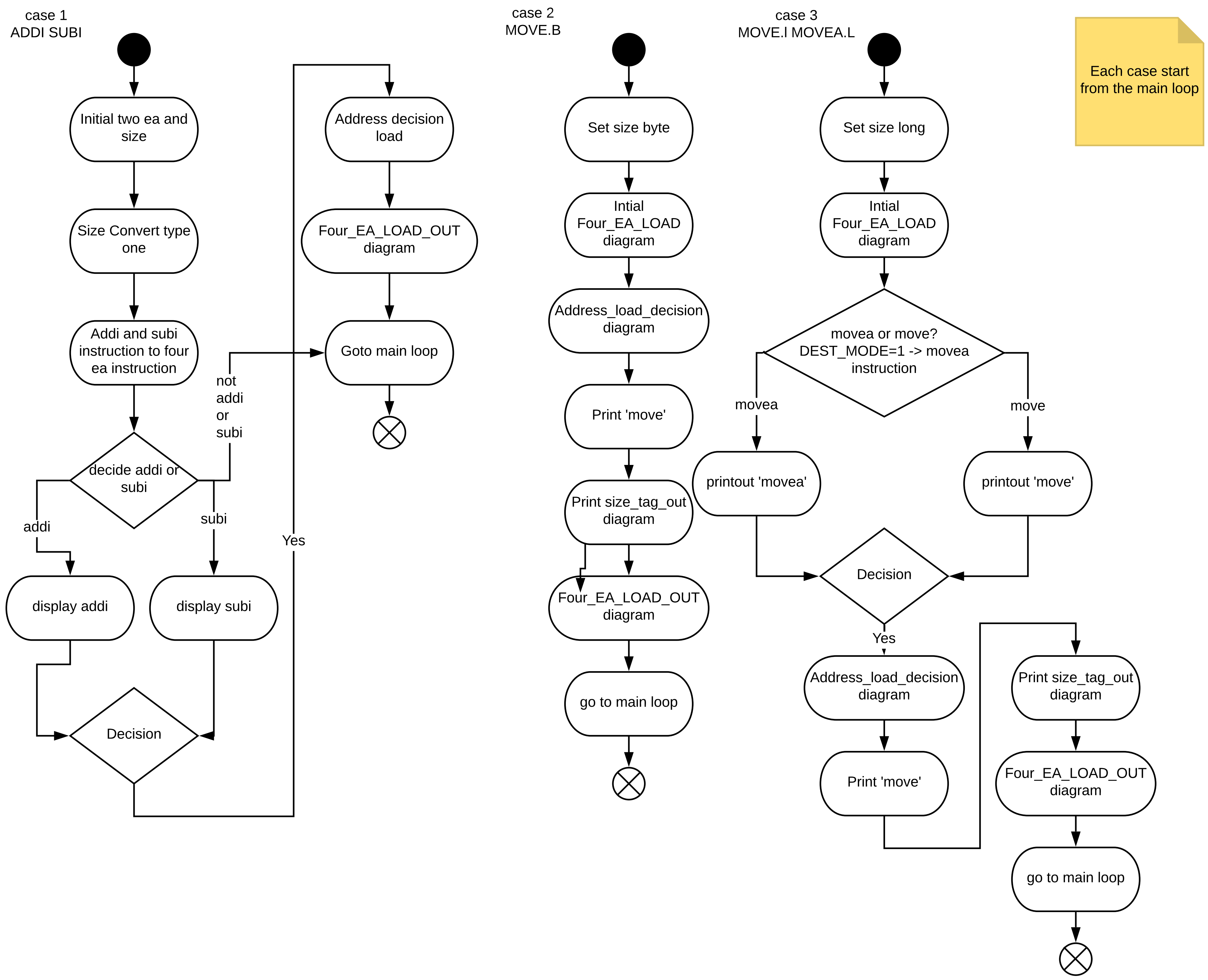
String to Hex

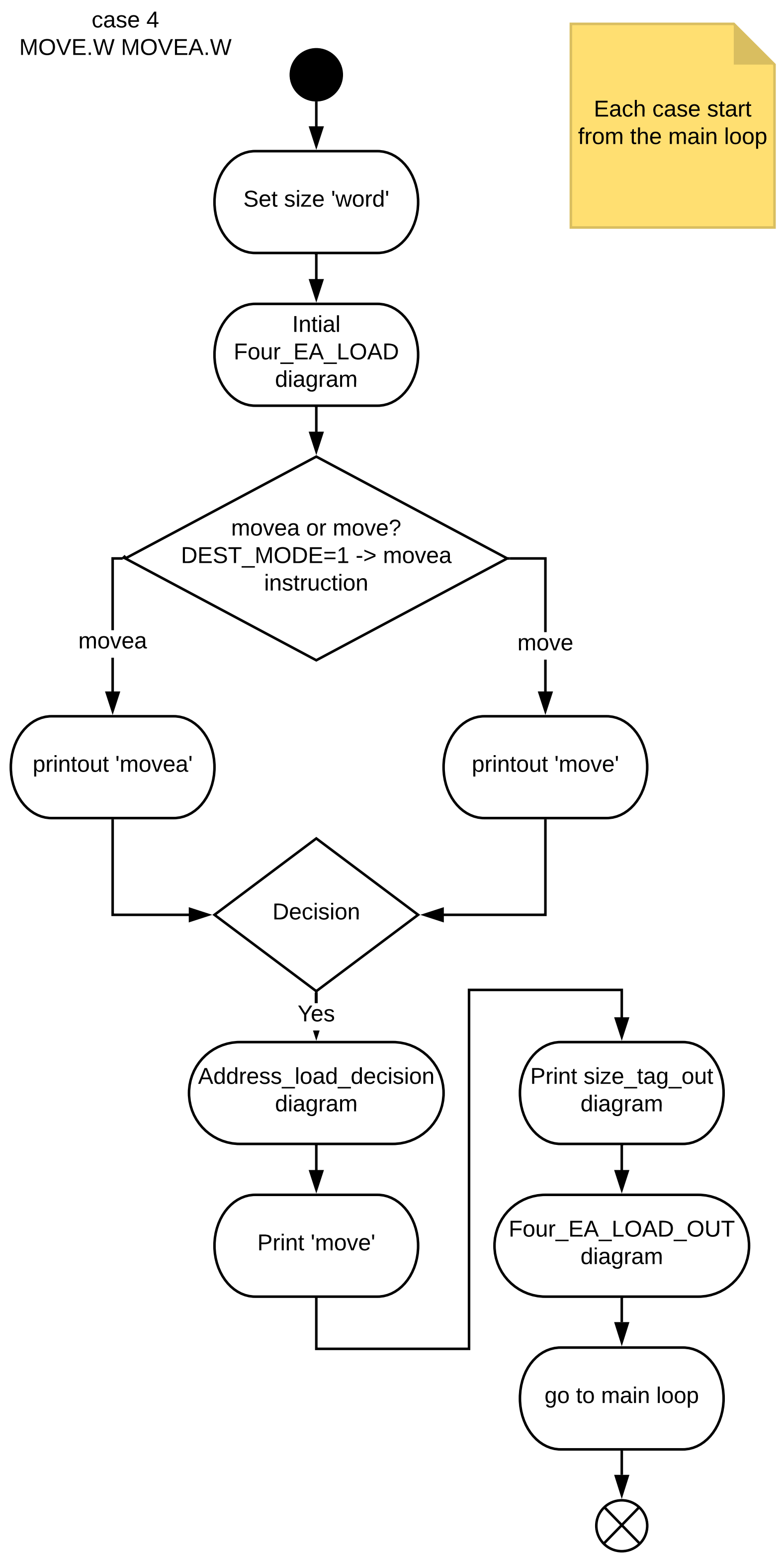

Move byte from a1 into d2
Shift bits in d3 to the left by 4
Move long data in d3 into variable
Byte in d2 is greater than #$60
d0 is equal to str_length
No
No
Yes
No
Subtract #$57 from byte in d2
Add byte to d3
Subtract #$37 from byte in d2
No
Byte in d2 is less than #$40
No
Byte in d2 is greater than #$40
Yes
Yes
Subtract #$30 from byte in d2

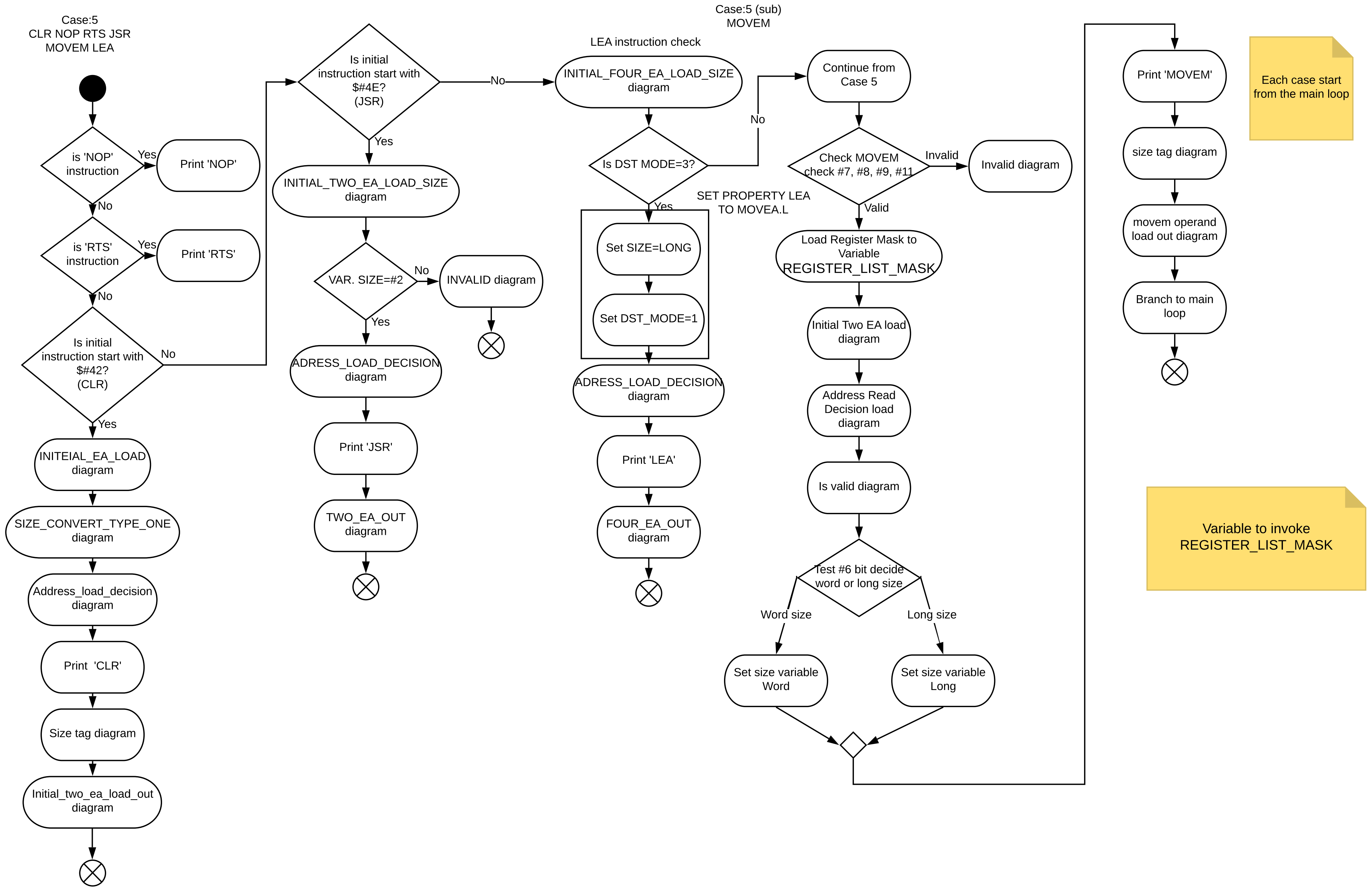
Opcode Dissassembly
(Main loop)



Load preceeding 16 bit instruction to the word size variable called 'INITIAL_INSTRUCTION'
Ending address check
Yes
Ask Disasseble code again? Dialog
No
Count Check (0..20)
Yes
Wait user input (Enter) to proceed more
No
INITIAL_ INSTRUCTION >#$10 (byte)
Yes
case 1: addi, subi data flow
It will compare the preceding byte. (15-8)
No
INITIAL_ INSTRUCTION >#$20 (byte)
Yes
case 2: move.b data flow
No
INITIAL_ INSTRUCTION >#$30 (byte)
Yes
case 3: move.l movea.l data flow
No
INITIAL_ INSTRUCTION >#$40 (byte)
Yes
case 4: move.w movea.w data flow
No
INITIAL_ INSTRUCTION >#$50 (byte)
Yes
case 5: clr, nop, rts, jsr, movem, lea data flow
No
INITIAL_ INSTRUCTION >#$60 (byte)
Yes
case 6: addq data flow
No
INITIAL_ INSTRUCTION >#$70 (byte)
Yes
case 7: bcc data flow
No
INITIAL_ INSTRUCTION >#$80 (byte)
Yes
case 8: moveq data flow
No
INITIAL_ INSTRUCTION >#$90 (byte)
Yes
case 9: or divu data flow
Yes
INITIAL_ INSTRUCTION >#$A0 (byte)
Yes
case 10: sub data flow
No
INITIAL_ INSTRUCTION >#$B0 (byte)
Yes
Invalid data flow
No
INITIAL_ INSTRUCTION >#$B0 (byte)
Yes
case 11: cmp data flow
No
INITIAL_ INSTRUCTION >#$C0 (byte)
Yes
case 12: and, muls, mulu data flow
No
INITIAL_ INSTRUCTION >#$D0 (byte)
Yes
case 13: add, adda data flow
No
INITIAL_ INSTRUCTION >#$E0 (byte)
Yes
caase 14: asr, asl, lsr, lsl, ror, rol data flow
No


case 1
ADDI SUBI
Initial two ea and size
Size Convert type one
Addi and subi instruction to four ea instruction
decide addi or subi
addi
subi
not addi or subi
display addi
display subi
Decision
Yes
Address decision load
Four_EA_LOAD_OUT diagram
Goto main loop
case 2
MOVE.B
Set size byte
Intial Four_EA_LOAD diagram
Address_load_decision diagram
Print 'move'
Print size_tag_out diagram
Four_EA_LOAD_OUT diagram
go to main loop
case 3
MOVE.I MOVEA.L
Set size long
Intial Four_EA_LOAD diagram
movea or move? DEST_MODE=1 -> movea instruction
movea
move
printout 'movea'
printout 'move'
Decision
Yes
Address_load_decision diagram
Print 'move'
Print size_tag_out diagram
Four_EA_LOAD_OUT diagram
go to main loop
Each case start from the main loop

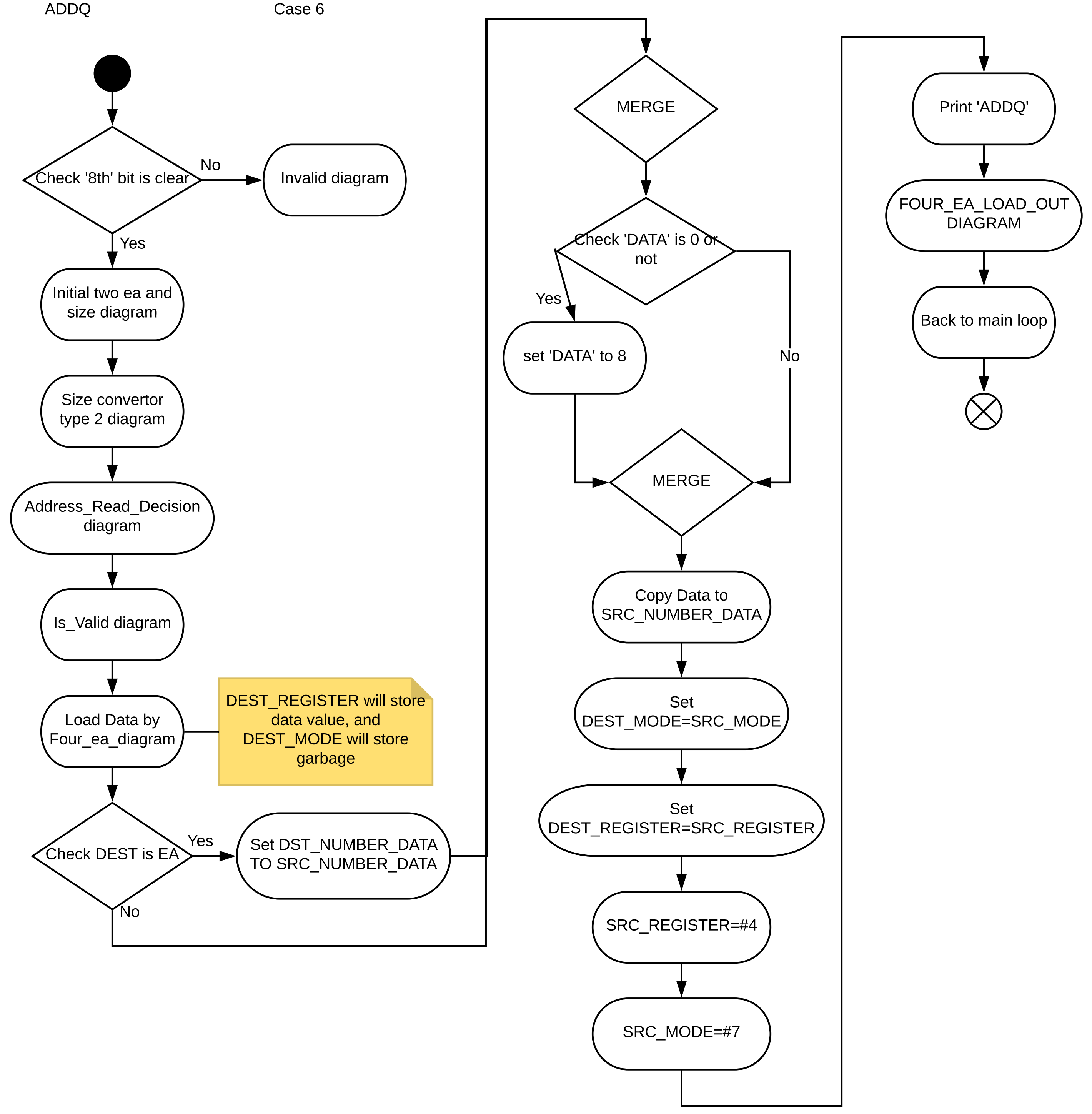

case 4
MOVE.W MOVEA.W
Each case start from the main loop
Set size 'word'
Intial Four_EA_LOAD diagram
movea or move? DEST_MODE=1 -> movea instruction
movea
move
printout 'movea'
printout 'move'
Decision
Yes
Address_load_decision diagram
Print 'move'
Print size_tag_out diagram
Four_EA_LOAD_OUT diagram
go to main loop


Case:5
CLR NOP RTS JSR
MOVEM LEA
is 'NOP' instruction
Yes
Print 'NOP'
No
is 'RTS' instruction
Yes
Print 'RTS'
No
Is initial instruction start with $#42? (CLR)
No
Yes
INITEIAL_EA_LOAD diagram
SIZE_CONVERT_TYPE_ONE diagram
Address_load_decision diagram
Print 'CLR'
Size tag diagram
Initial_two_ea_load_out diagram
Is initial instruction start with $#4E? (JSR)
No
Yes
INITIAL_TWO_EA_LOAD_SIZE diagram
VAR. SIZE=#2
No
INVALID diagram
Yes
ADRESS_LOAD_DECISION diagram
Print 'JSR'
TWO_EA_OUT diagram
LEA instruction check
INITIAL_FOUR_EA_LOAD_SIZE diagram
Is DST MODE=3?
No
Yes
SET PROPERTY LEA TO MOVEA.L
Set SIZE=LONG
Set DST_MODE=1
ADRESS_LOAD_DECISION diagram
Print 'LEA'
FOUR_EA_OUT diagram
Case:5 (sub)
MOVEM
Continue from Case 5
Check MOVEM check #7, #8, #9, #11
Invalid
Invalid diagram
Valid
Load Register Mask to Variable REGISTER_LIST_MASK
Initial Two EA load diagram
Address Read Decision load diagram
Is valid diagram
Test #6 bit decide word or long size
Word size
Long size
Set size variable Word
Set size variable Long
Print 'MOVEM'
size tag diagram
movem operand load out diagram
Branch to main loop
Each case start from the main loop
Variable to invoke REGISTER_LIST_MASK

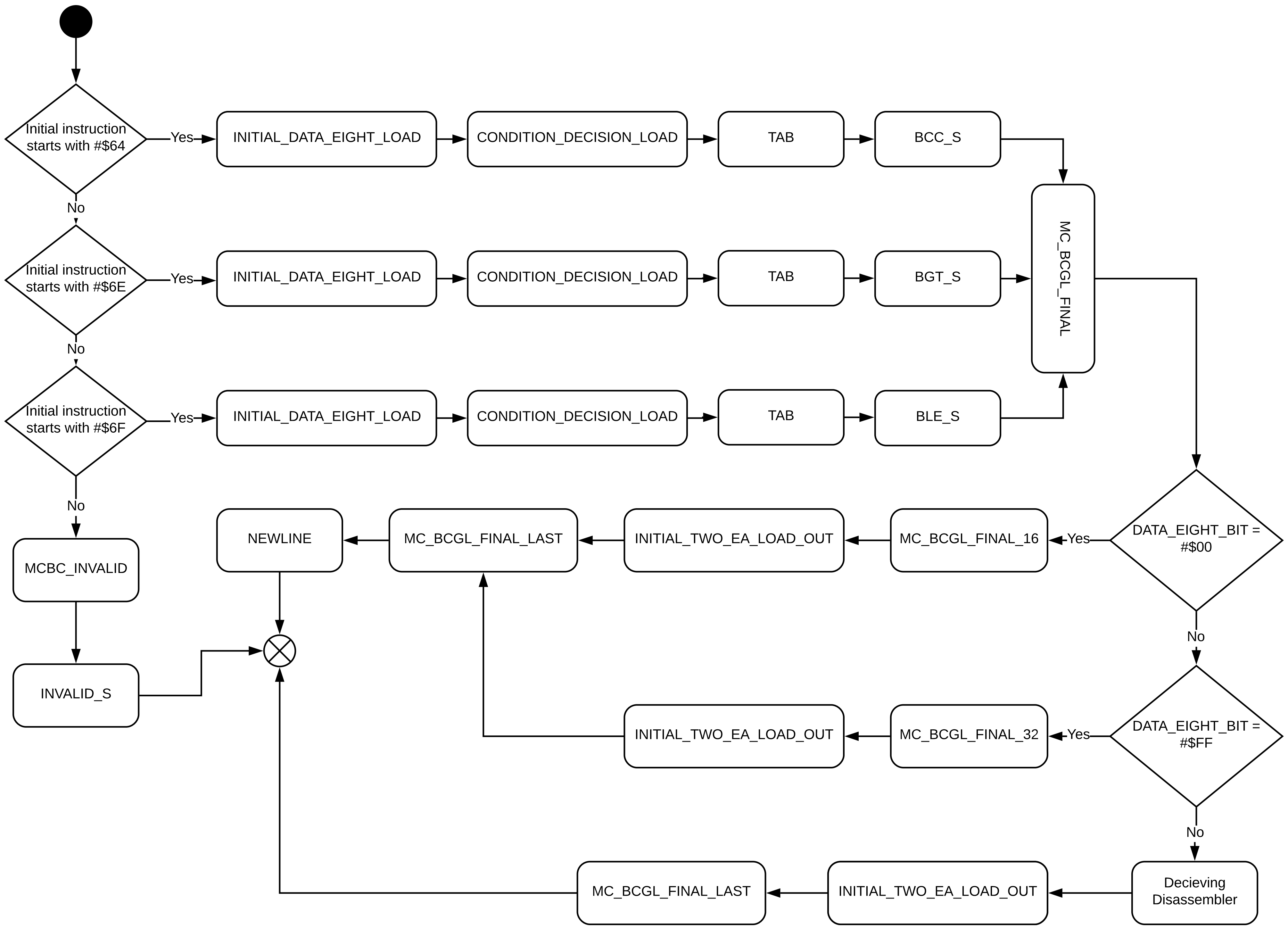
ADDQ

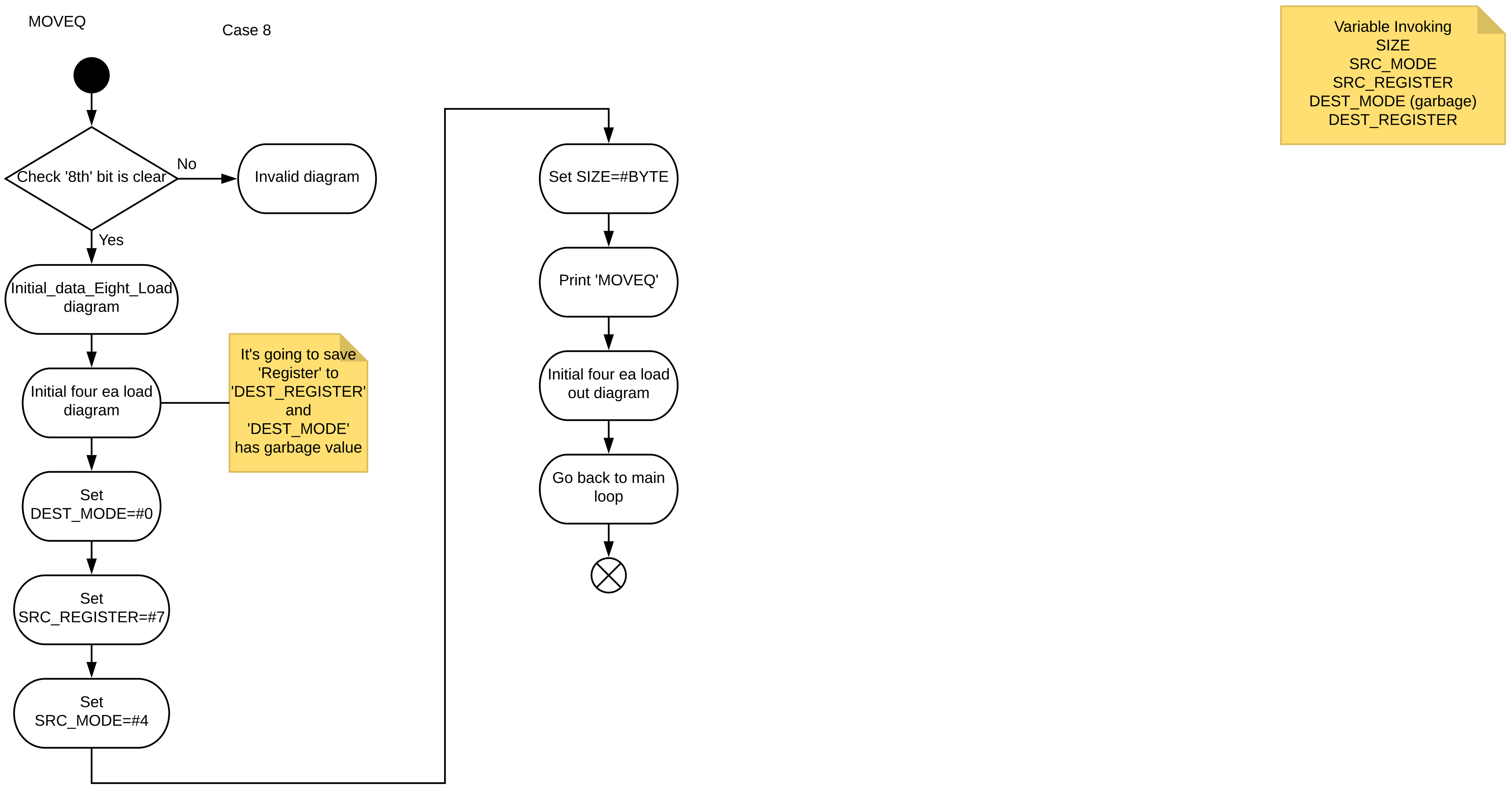
Case 6

To disassemble this,we invoke the existing functions, so, we make ADDQ function to have similar property with move.b, and we printout the operands.

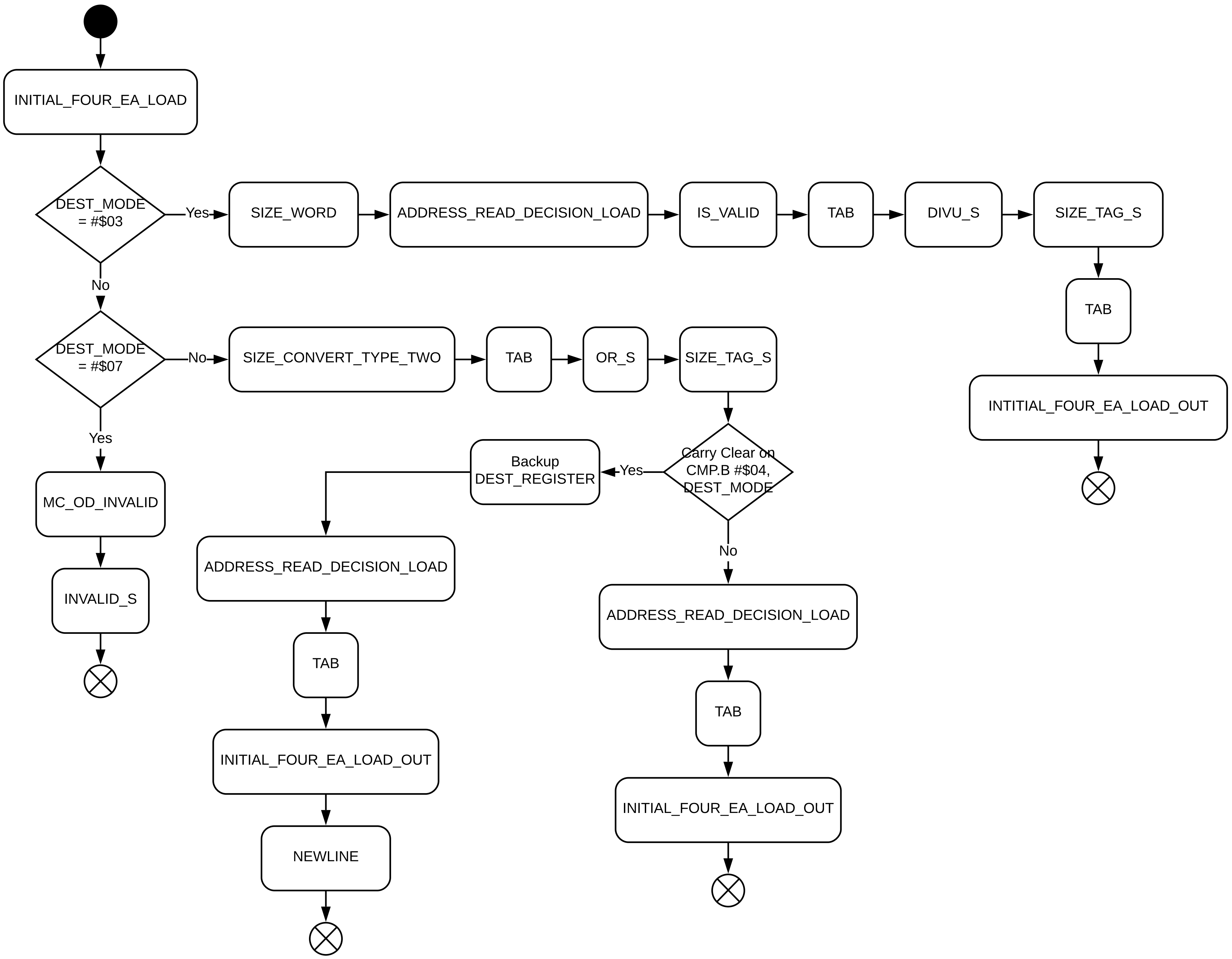

Check '8th' bit is clear
No
Invalid diagram
Yes
Initial two ea and size diagram
Size convertor type 2 diagram
Address_Read_Decision diagram
Is_Valid diagram
Load Data by Four_ea_diagram
DEST_REGISTER will store data value, and DEST_MODE will store garbage
Check DEST is EA
Yes
Set DST_NUMBER_DATA TO SRC_NUMBER_DATA
No
MERGE
Check 'DATA' is 0 or not
Yes
set 'DATA' to 8
No
MERGE
Copy Data to SRC_NUMBER_DATA
Set DEST_MODE=SRC_MODE
Set DEST_REGISTER=SRC_REGISTER
SRC_REGISTER=#4
SRC_MODE=#7
Print 'ADDQ'
FOUR_EA_LOAD_OUT DIAGRAM
Back to main loop

BCC BGT BLE		Case 7

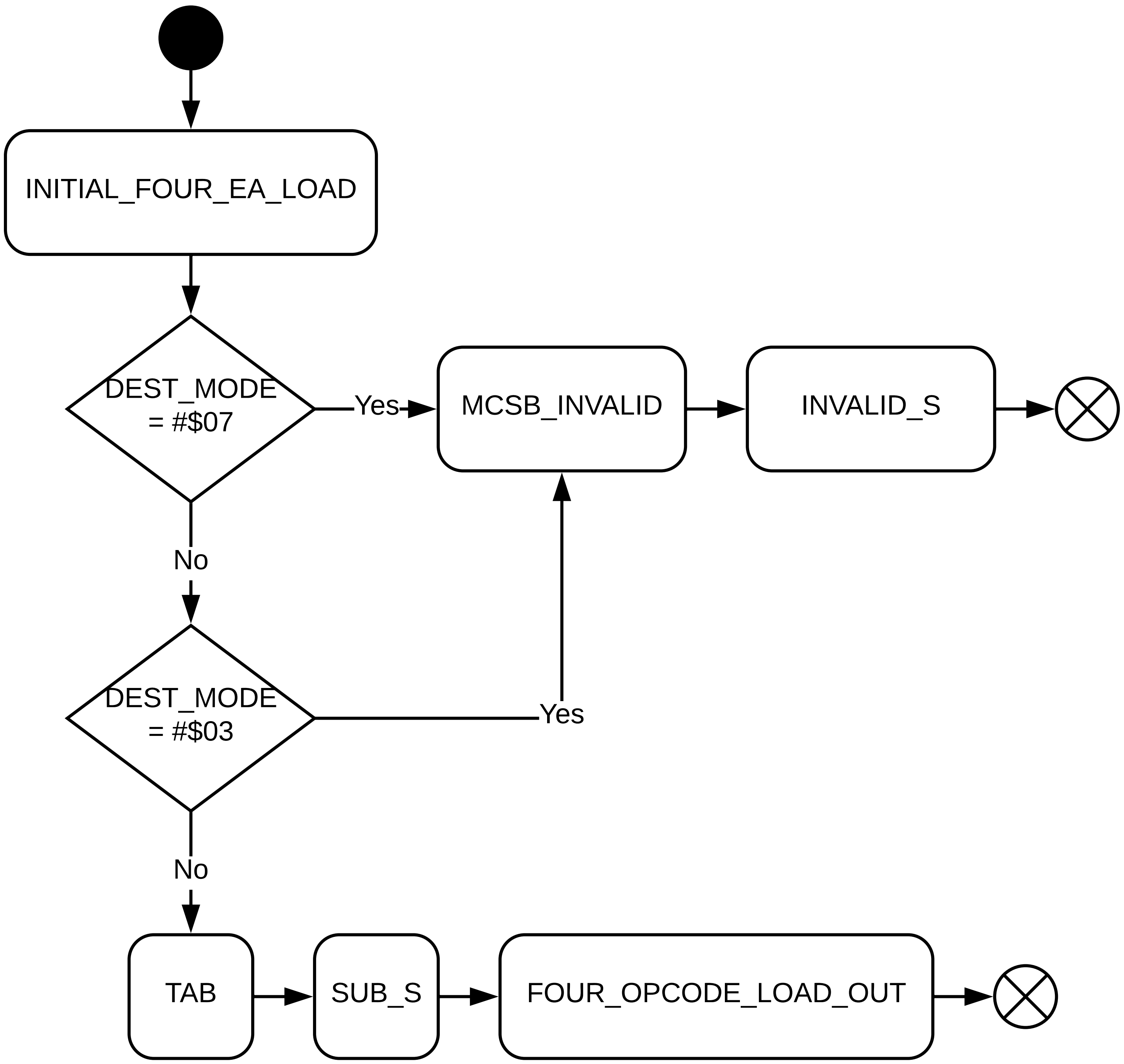



MOVEQ
Case 8
Check '8th' bit is clear
No
Invalid diagram
Yes
Initial_data_Eight_Load diagram
Initial four ea load diagram
It's going to save 'Register' to 'DEST_REGISTER' and 'DEST_MODE' has garbage value
Set DEST_MODE=#0
Set SRC_REGISTER=#7
Set SRC_MODE=#4
Set SIZE=#BYTE
Print 'MOVEQ'
Initial four ea load out diagram
Go back to main loop
Variable Invoking
SIZE
SRC_MODE
SRC_REGISTER
DEST_MODE (garbage)
DEST_REGISTER

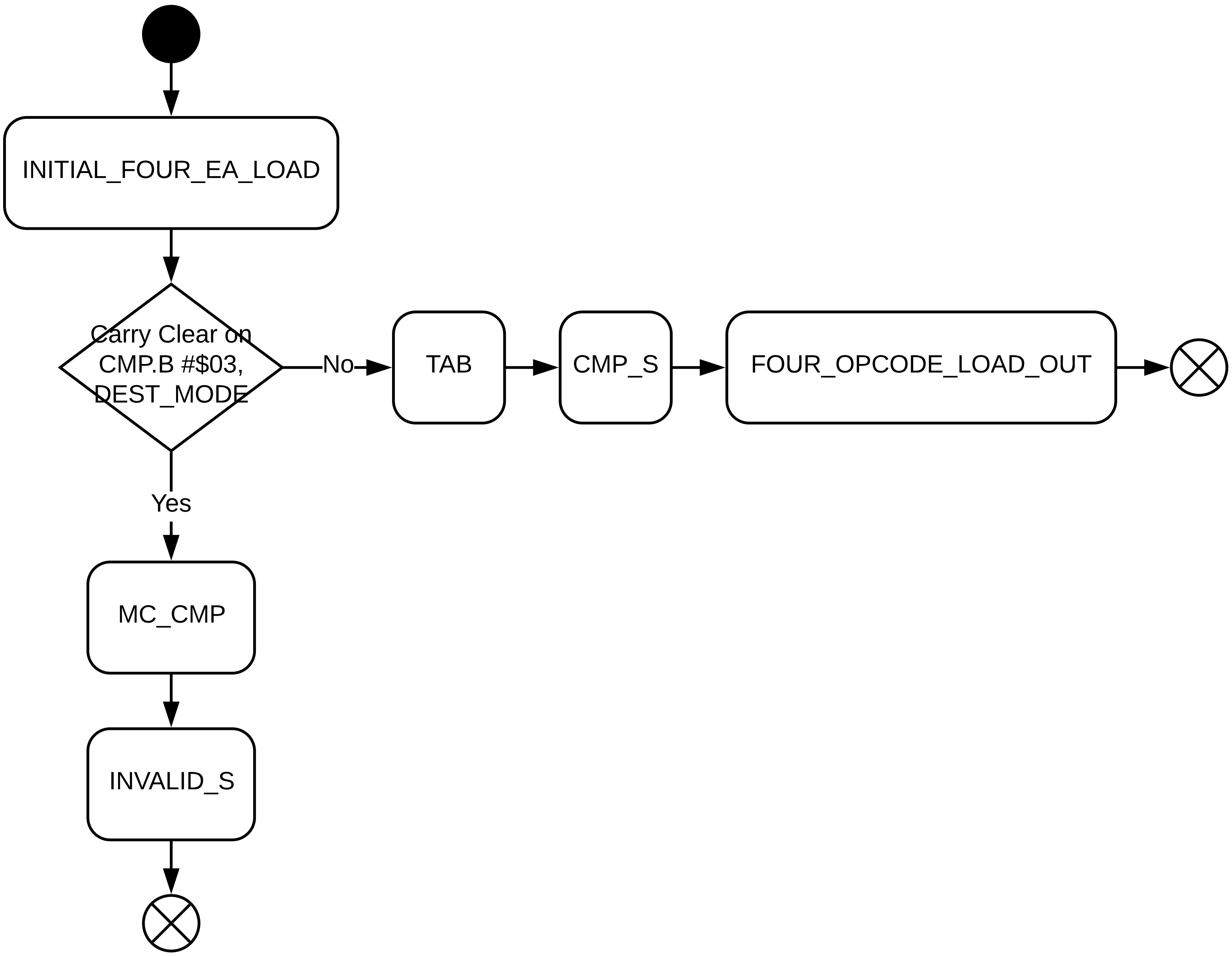
OR DIVU Case 9



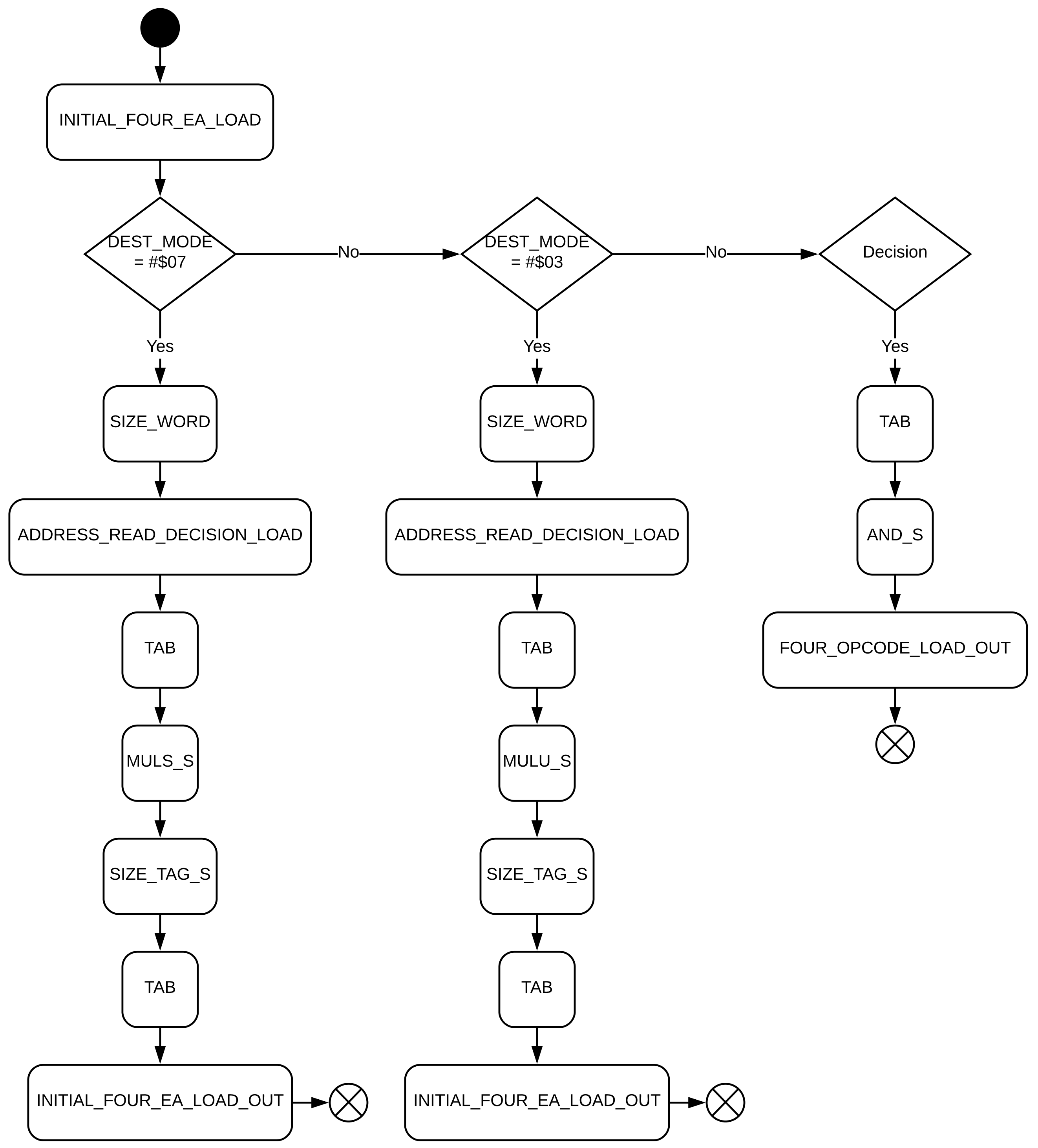
SUB Case 10


INITIAL_FOUR_EA_LOAD
DEST_MODE = #$07
Yes
MCSB_INVALID
INVALID_S
No
DEST_MODE = #$03
Yes
No
TAB
SUB_S
FOUR_OPCODE_LOAD_OUT

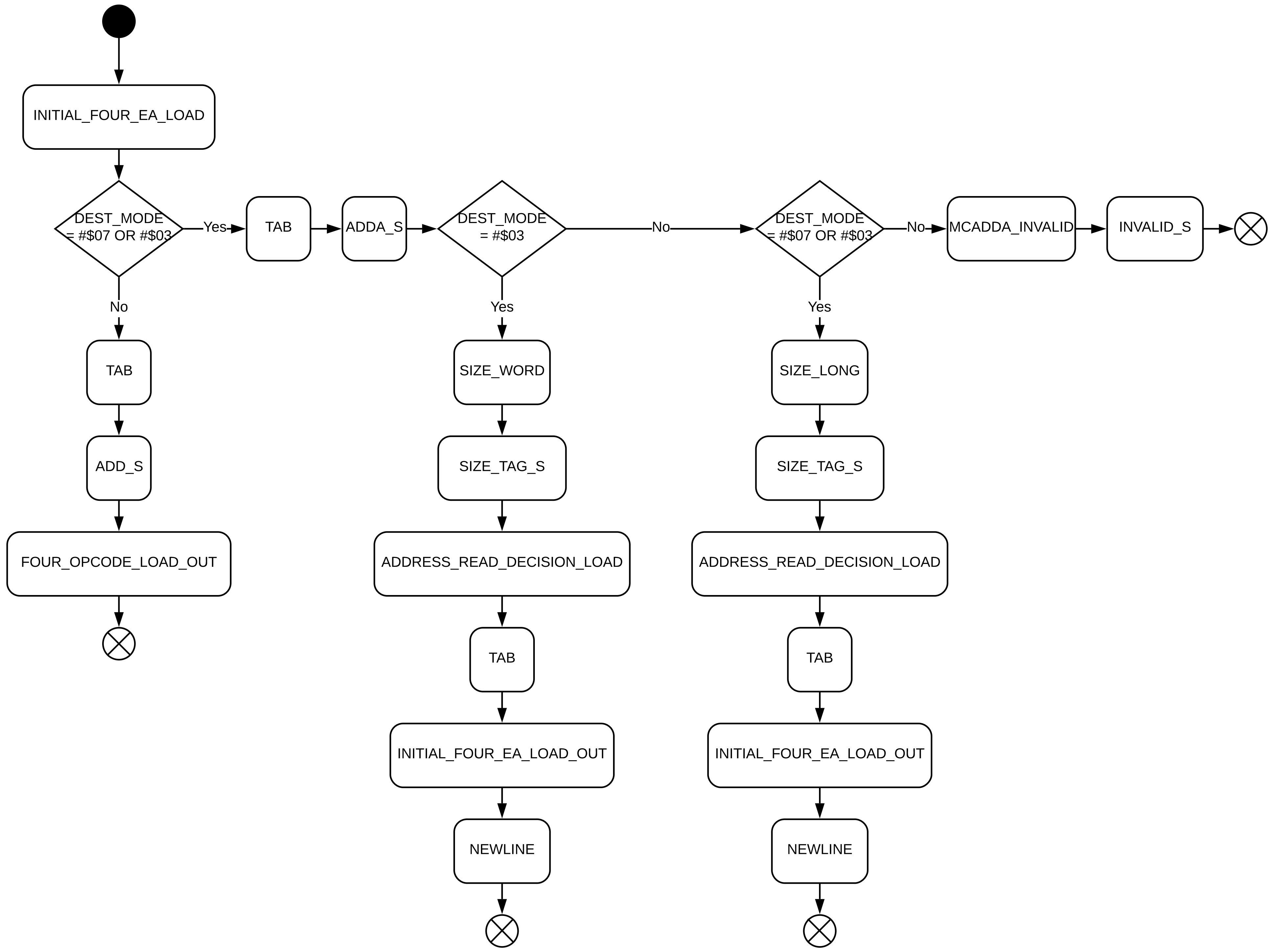
CMP Case 11

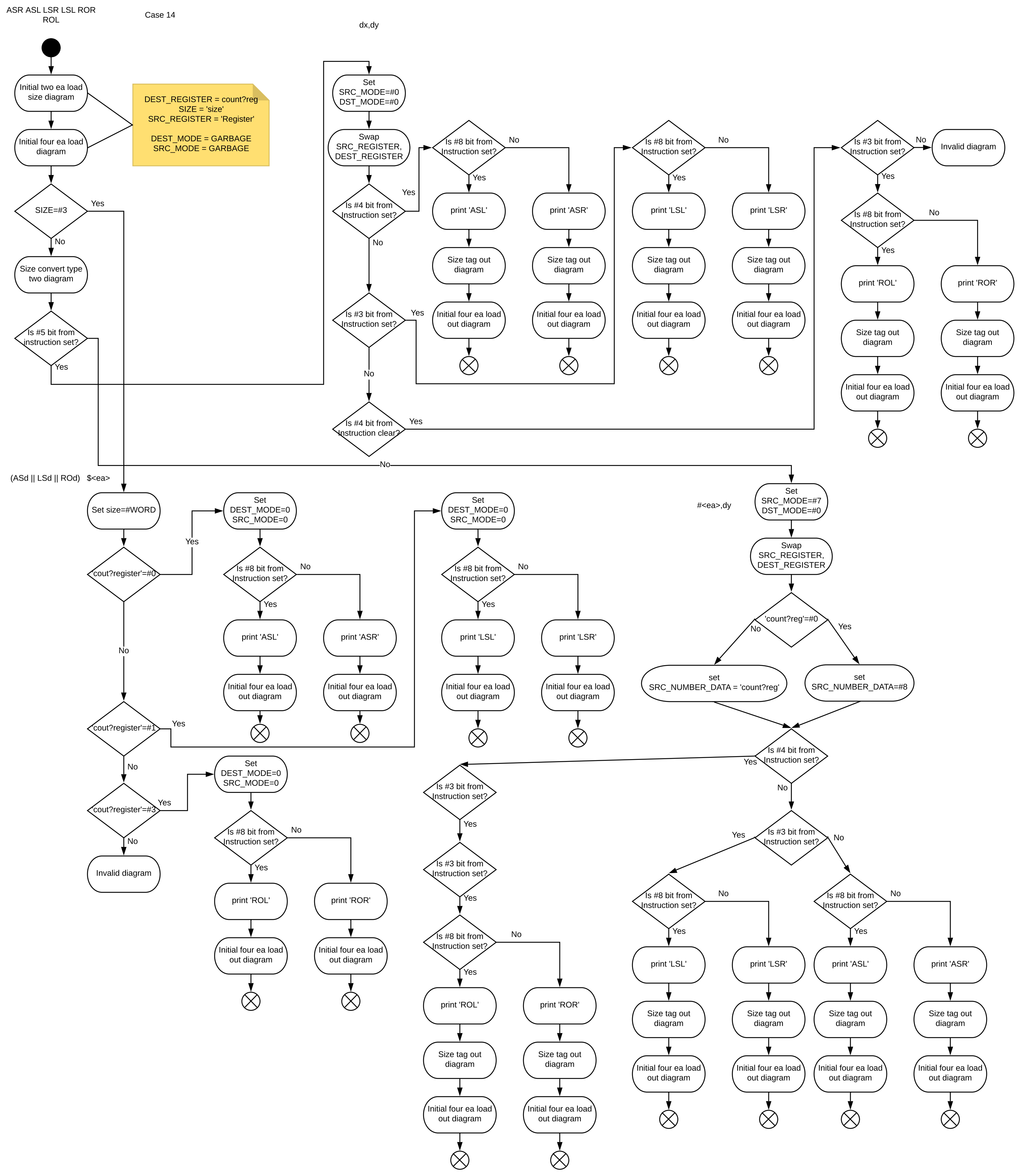

INITIAL_FOUR_EA_LOAD
Carry Clear on CMP.B #$03, DEST_MODE
No
TAB
CMP_S
FOUR_OPCODE_LOAD_OUT
Yes
MC_CMP
INVALID_S

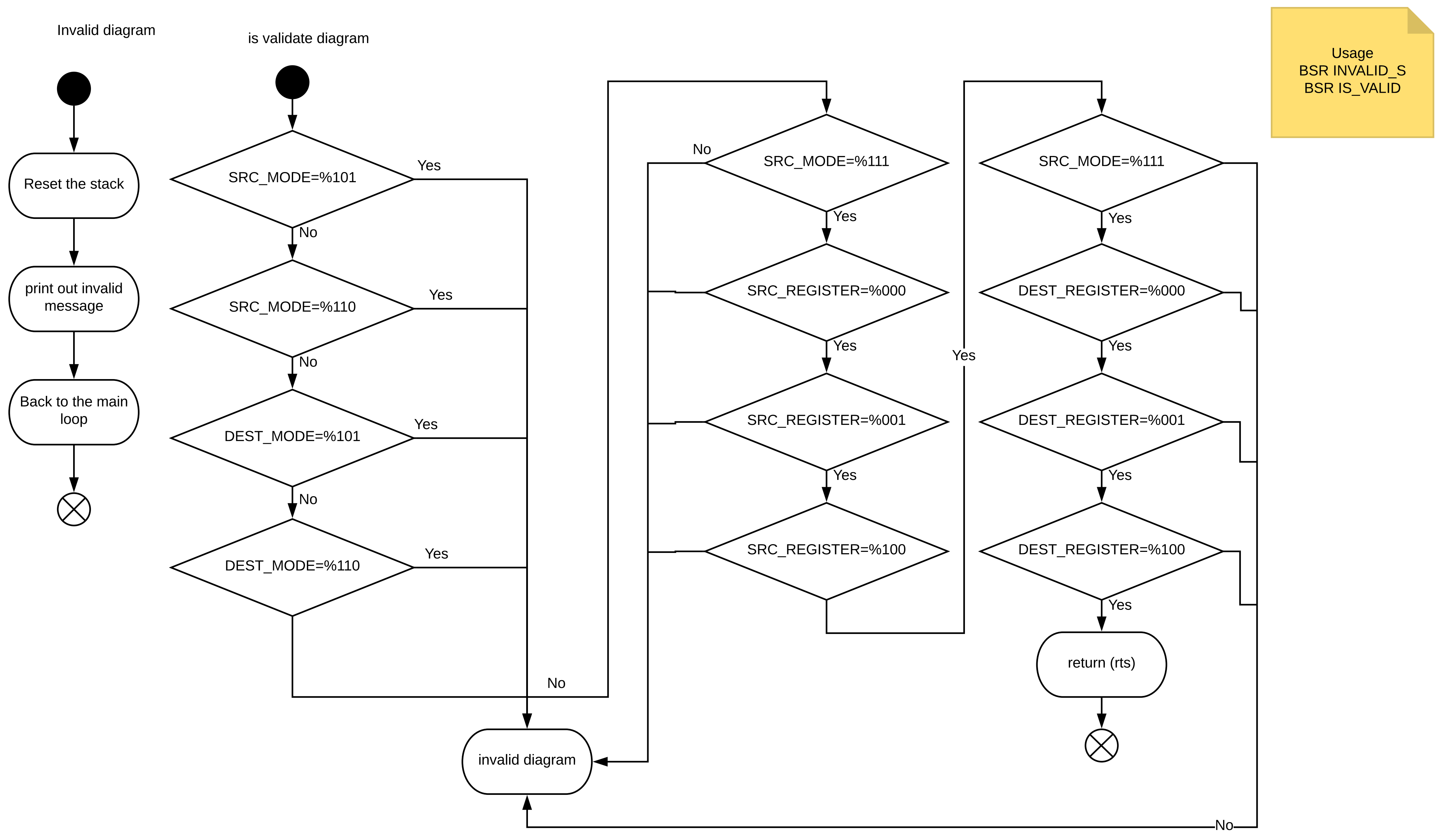
AND MULS MULU Case 12



ADD ADDA Case 13




ASR ASL LSR LSL ROR ROL
Case 14
dx,dy
Initial two ea load size diagram
Initial four ea load diagram
DEST_REGISTER = count?reg
SIZE = 'size'
SRC_REGISTER = 'Register'
DEST_MODE = GARBAGE
SRC_MODE = GARBAGE
SIZE=#3
Yes
No
Size convert type two diagram
Is #5 bit from instruction set?
Yes
Set SRC_MODE=#0 DST_MODE=#0
Swap SRC_REGISTER, DEST_REGISTER
Is #4 bit from Instruction set?
Yes
No
Is #8 bit from Instruction set?
No
Yes
print 'ASL'
print 'ASR'
Size tag out diagram
Initial four ea load out diagram
Is #3 bit from Instruction set?
Yes
No
Is #8 bit from Instruction set?
print 'LSL'
print 'LSR'
Is #4 bit from Instruction clear?
Yes
Is #3 bit from Instruction set?
Invalid diagram
Is #8 bit from Instruction set?
print 'ROL'
print 'ROR'
No
(ASd || LSd || ROd) $<ea>
Set size=#WORD
cout?register'=#0
Set DEST_MODE=0 SRC_MODE=0
print 'ASL'
print 'ASR'
cout?register'=#1
print 'LSL'
print 'LSR'
cout?register'=#3
Invalid diagram
print 'ROL'
print 'ROR'
#<ea>,dy
Set SRC_MODE=#7 DST_MODE=#0
Swap SRC_REGISTER, DEST_REGISTER
'count?reg'=#0
set SRC_NUMBER_DATA = 'count?reg'
set SRC_NUMBER_DATA=#8
Is #4 bit from Instruction set?
Is #3 bit from Instruction set?
Is #3 bit from Instruction set?
Is #8 bit from Instruction set?
print 'ROL'
print 'ROR'
Is #3 bit from Instruction set?
Is #8 bit from Instruction set?
print 'LSL'
print 'LSR'
print 'ASL'
print 'ASR'


Invalid diagram
is validate diagram
Usage
BSR INVALID_S
BSR IS_VALID
Reset the stack
print out invalid message
Back to the main loop
SRC_MODE=%101
Yes
No
SRC_MODE=%110
Yes
No
DEST_MODE=%101
Yes
No
DEST_MODE=%110
Yes
No
invalid diagram
SRC_MODE=%111
No
Yes
SRC_REGISTER=%000
Yes
SRC_REGISTER=%001
Yes
SRC_REGISTER=%100
Yes
SRC_MODE=%111
Yes
DEST_REGISTER=%000
Yes
DEST_REGISTER=%001
Yes
DEST_REGISTER=%100
Yes
return (rts)
No

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										Src Mode			Src Register		

Load initial two ea

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								size		Src Mode			Src Register		

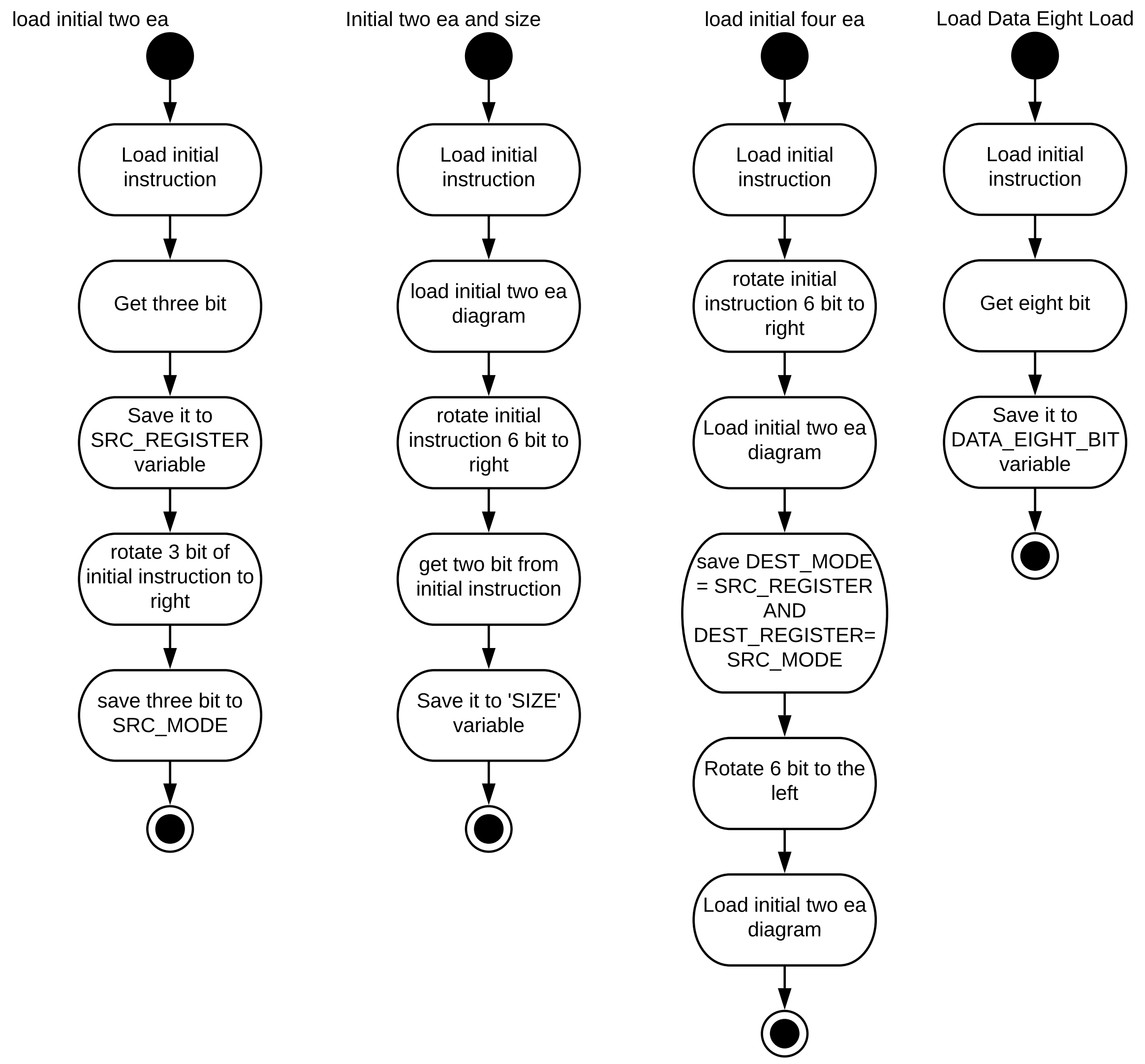
Load initial two ea and size

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				Dest Register			Dest Mode			Src Mode			Src Register		

Load initial four ea

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Data							

Load Data Eight Load

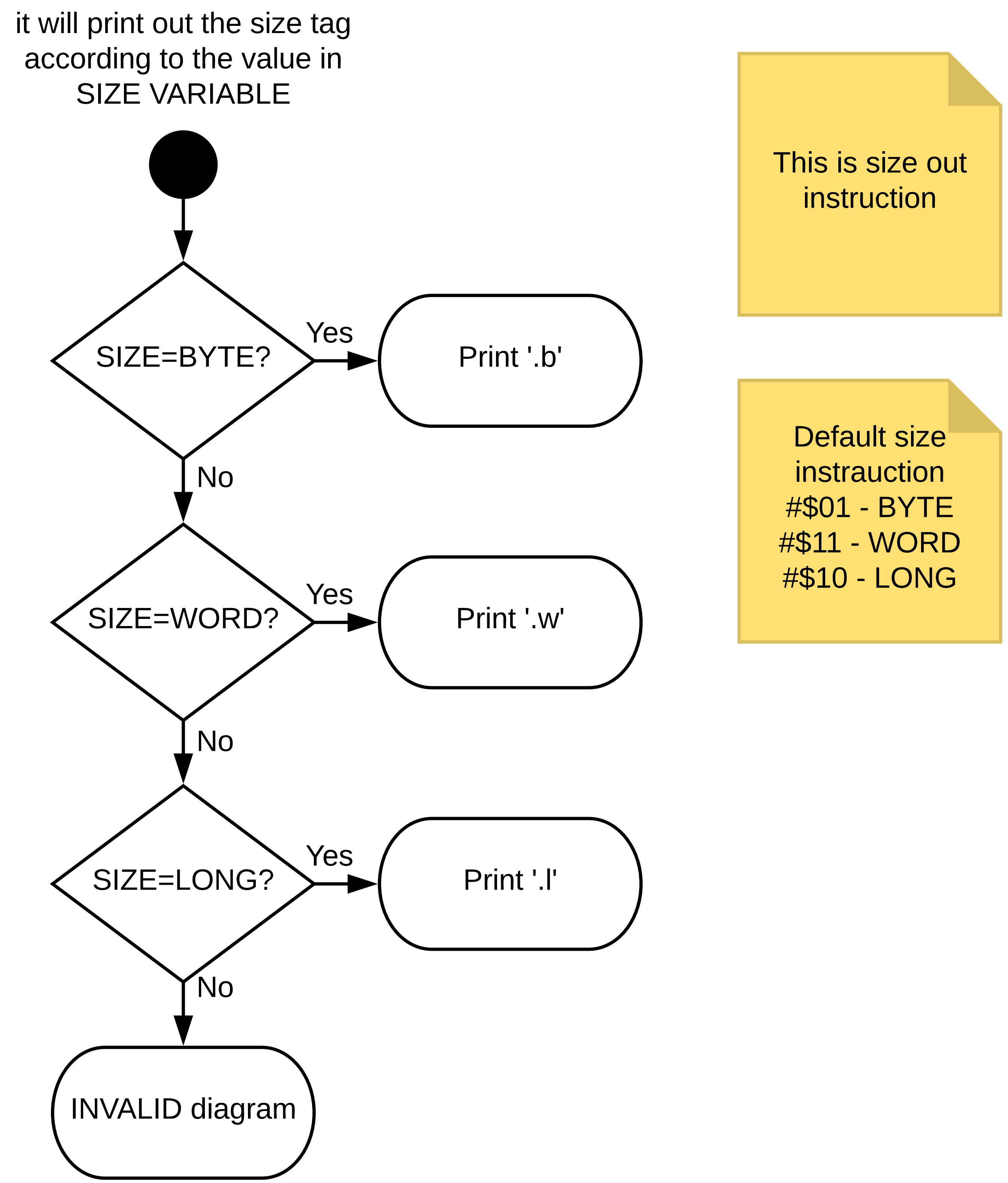

load initial two ea
Load initial instruction
Get three bit
Save it to SRC_REGISTER variable
rotate 3 bit of initial instruction to right
save three bit to SRC_MODE
Initial two ea and size
Load initial instruction
load initial two ea diagram
rotate initial instruction 6 bit to right
get two bit from initial instruction
Save it to 'SIZE' variable
load initial four ea
Load initial instruction
rotate initial instruction 6 bit to right
Load initial two ea diagram
save DEST_MODE = SRC_REGISTER AND DEST_REGISTER= SRC_MODE
Rotate 6 bit to the left
Load initial two ea diagram
Load Data Eight Load
Load initial instruction
Get eight bit
Save it to DATA_EIGHT_BIT variable

load initial two ea ->
BSR INITIAL_TWO_EA_LOAD
set ->
SRC_MODE
SRC_REGISTER

load initial two ea and size ->
BSR INITIAL_TWO_EA_LOAD_SIZE
set ->
SRC_MODE
SRC_REGISTER
SIZE

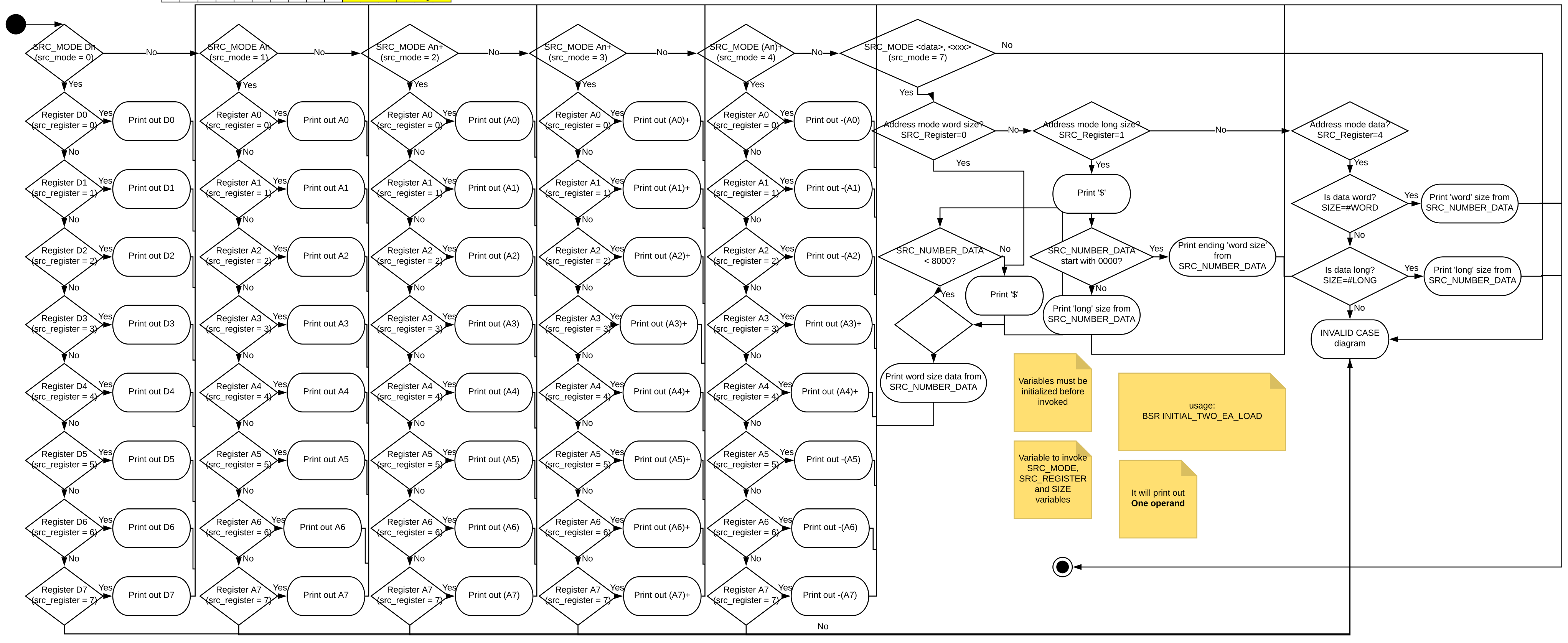
load initial four ea =
INITIAL_FOUR_EA_LOAD
SRC_MODE
SRC_REGISTER
DEST_MODE
DEST_REGISTER

Load Data Eight Load =
INITIAL_DATA_EIGHT_LOAD
DATA_EIGHT_BIT


it will print out the size tag according to the value in SIZE VARIABLE
SIZE=BYTE?
Yes
Print '.b'
No
SIZE=WORD?
Yes
Print '.w'
No
SIZE=LONG?
Yes
Print '.l'
No
INVALID diagram
This is size out instruction
Default size instrauction
#$01 - BYTE
#$11 - WORD
#$10 - LONG

Initial two ea out (one operand out)

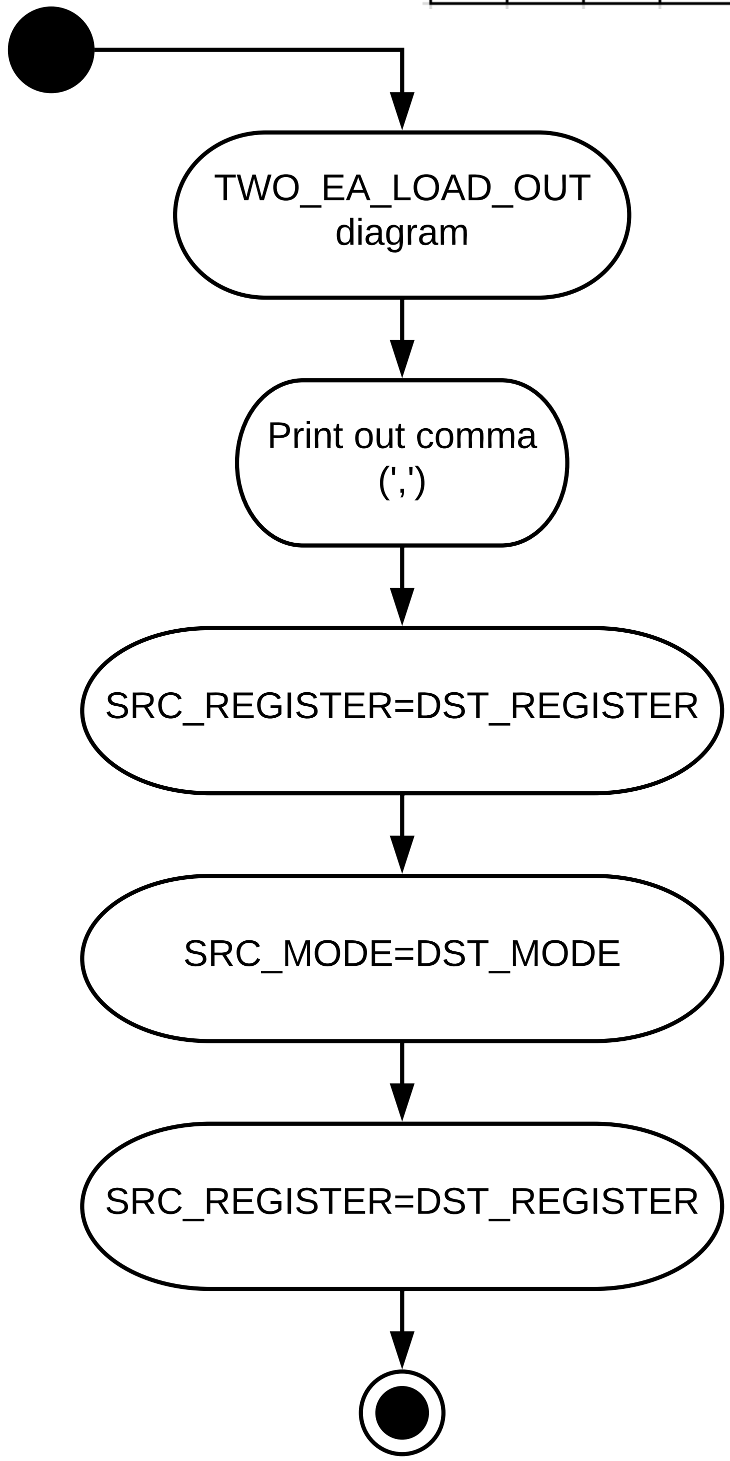
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										Src Mode			Src Register		



SRC_MODE Dn (src_mode = 0)
SRC_MODE An (src_mode = 1)
SRC_MODE An+ (src_mode = 2)
SRC_MODE An+ (src_mode = 3)
SRC_MODE (An)+ (src_mode = 4)
SRC_MODE <data>, <xxx> (src_mode = 7)
Yes
No
Register D0 (src_register = 0)
Register D1 (src_register = 1)
Register D2 (src_register = 2)
Register D3 (src_register = 3)
Register D4 (src_register = 4)
Register D5 (src_register = 5)
Register D6 (src_register = 6)
Register D7 (src_register = 7)
Print out D0
Print out D1
Print out D2
Print out D3
Print out D4
Print out D5
Print out D6
Print out D7
Register A0 (src_register = 0)
Register A1 (src_register = 1)
Register A2 (src_register = 2)
Register A3 (src_register = 3)
Register A4 (src_register = 4)
Register A5 (src_register = 5)
Register A6 (src_register = 6)
Register A7 (src_register = 7)
Print out A0
Print out A1
Print out A2
Print out A3
Print out A4
Print out A5
Print out A6
Print out A7
Print out (A0)
Print out (A1)
Print out (A2)
Print out (A3)
Print out (A4)
Print out (A5)
Print out (A6)
Print out (A7)
Print out (A0)+
Print out (A1)+
Print out (A2)+
Print out (A3)+
Print out (A4)+
Print out (A5)+
Print out (A6)+
Print out (A7)+
Print out -(A0)
Print out -(A1)
Print out -(A2)
Print out (A3)+
Print out (A4)+
Print out -(A5)
Print out -(A6)
Print out -(A7)
Address mode word size? SRC_Register=0
Address mode long size? SRC_Register=1
Address mode data? SRC_Register=4
Print '$'
SRC_NUMBER_DATA < 8000?
Print '$'
Print word size data from SRC_NUMBER_DATA
SRC_NUMBER_DATA start with 0000?
Print ending 'word size' from SRC_NUMBER_DATA
Print 'long' size from SRC_NUMBER_DATA
Is data word? SIZE=#WORD
Print 'word' size from SRC_NUMBER_DATA
Is data long? SIZE=#LONG
Print 'long' size from SRC_NUMBER_DATA
INVALID CASE diagram
Variables must be initialized before invoked
Variable to invoke SRC_MODE, SRC_REGISTER and SIZE variables
usage: BSR INITIAL_TWO_EA_LOAD
It will print out **One operand**

Initial four ea out

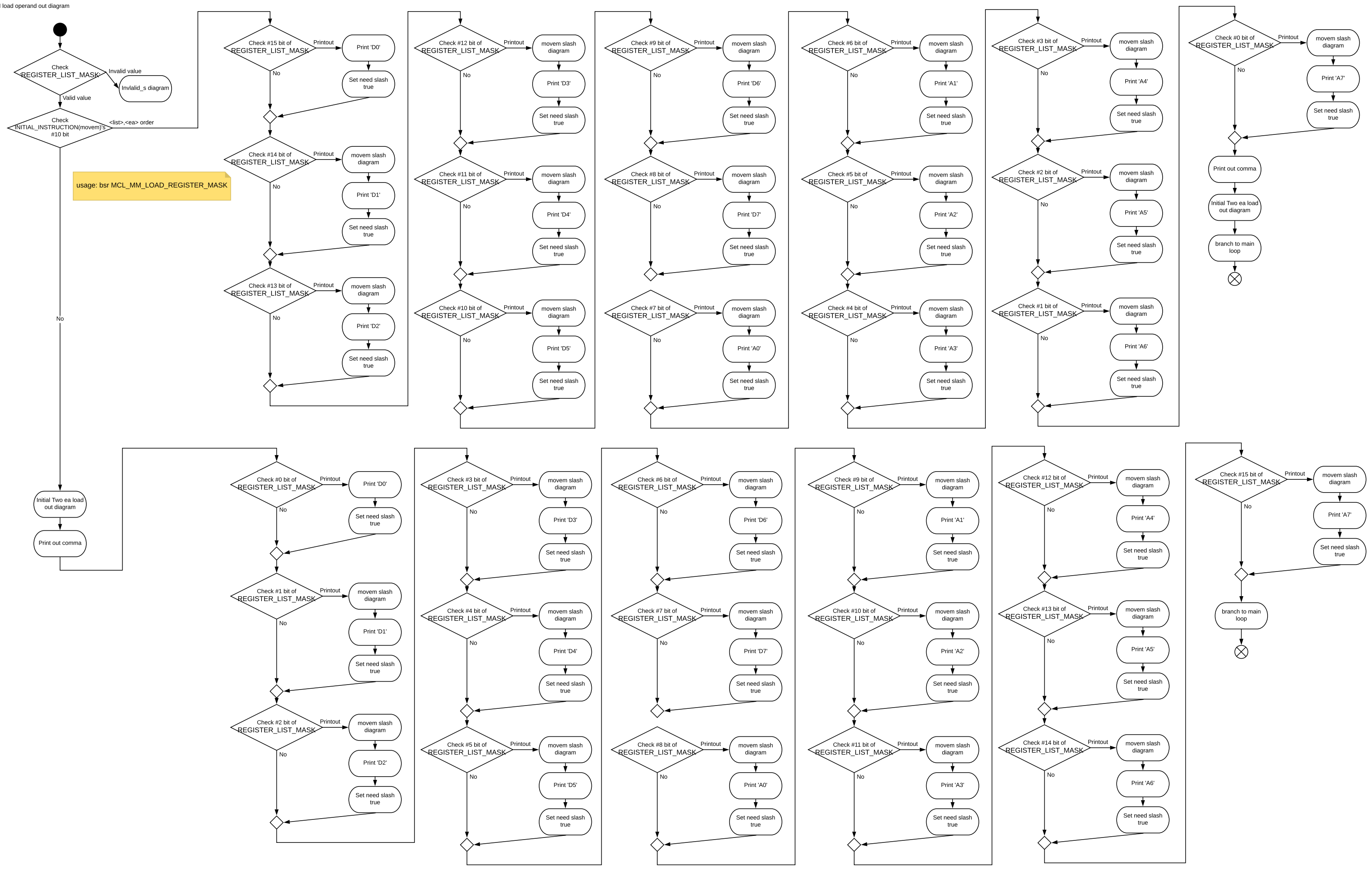
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				Dest Register			Dest Mode			Src Mode			Src Register		



Usage:
BSR INITIAL_FOUR_EA_LOAD _OUT

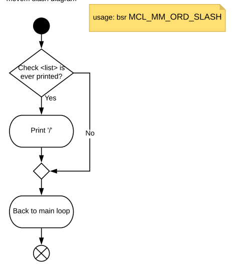
It will print out
two operands

MOVEM load operand out diagram

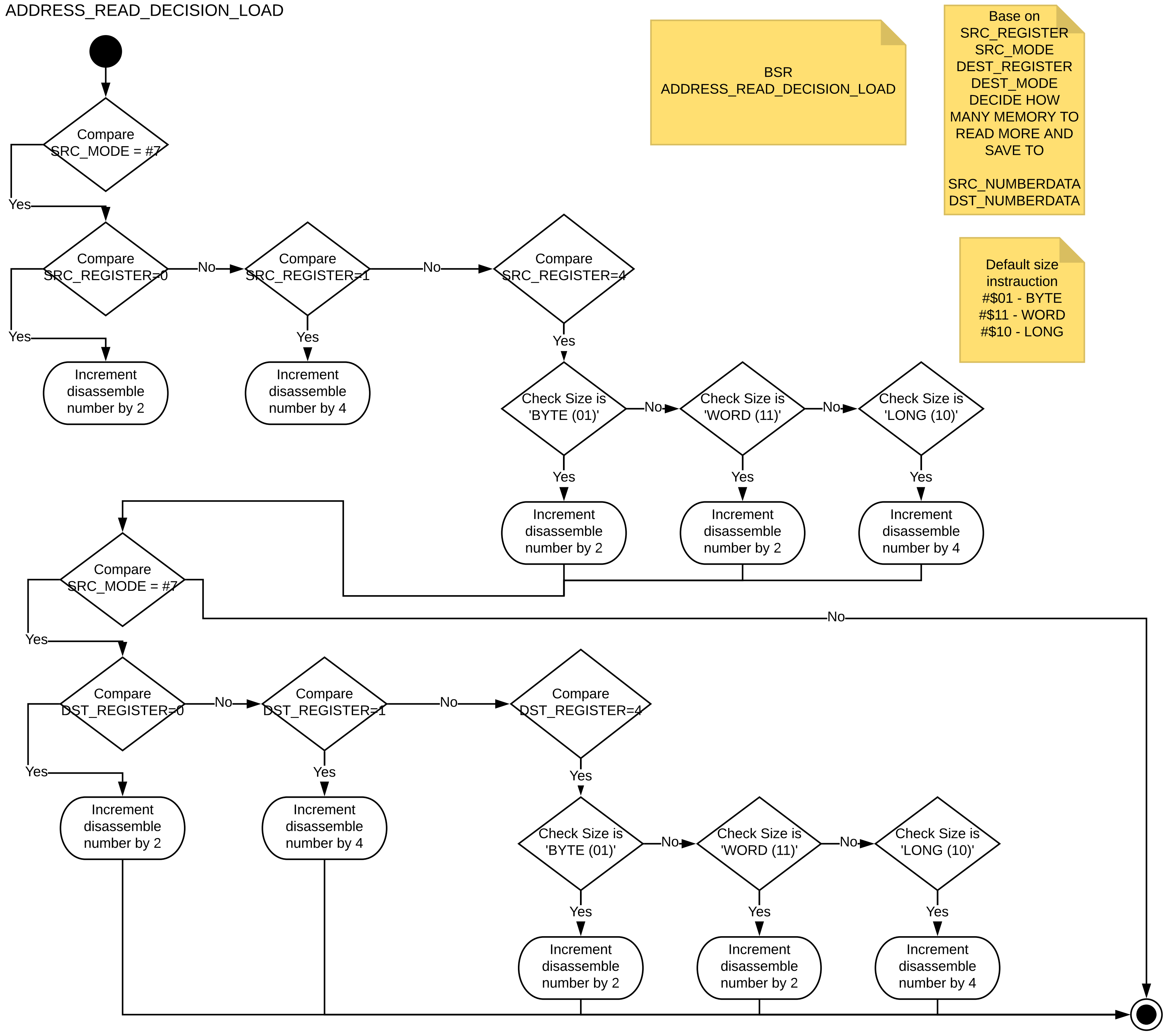


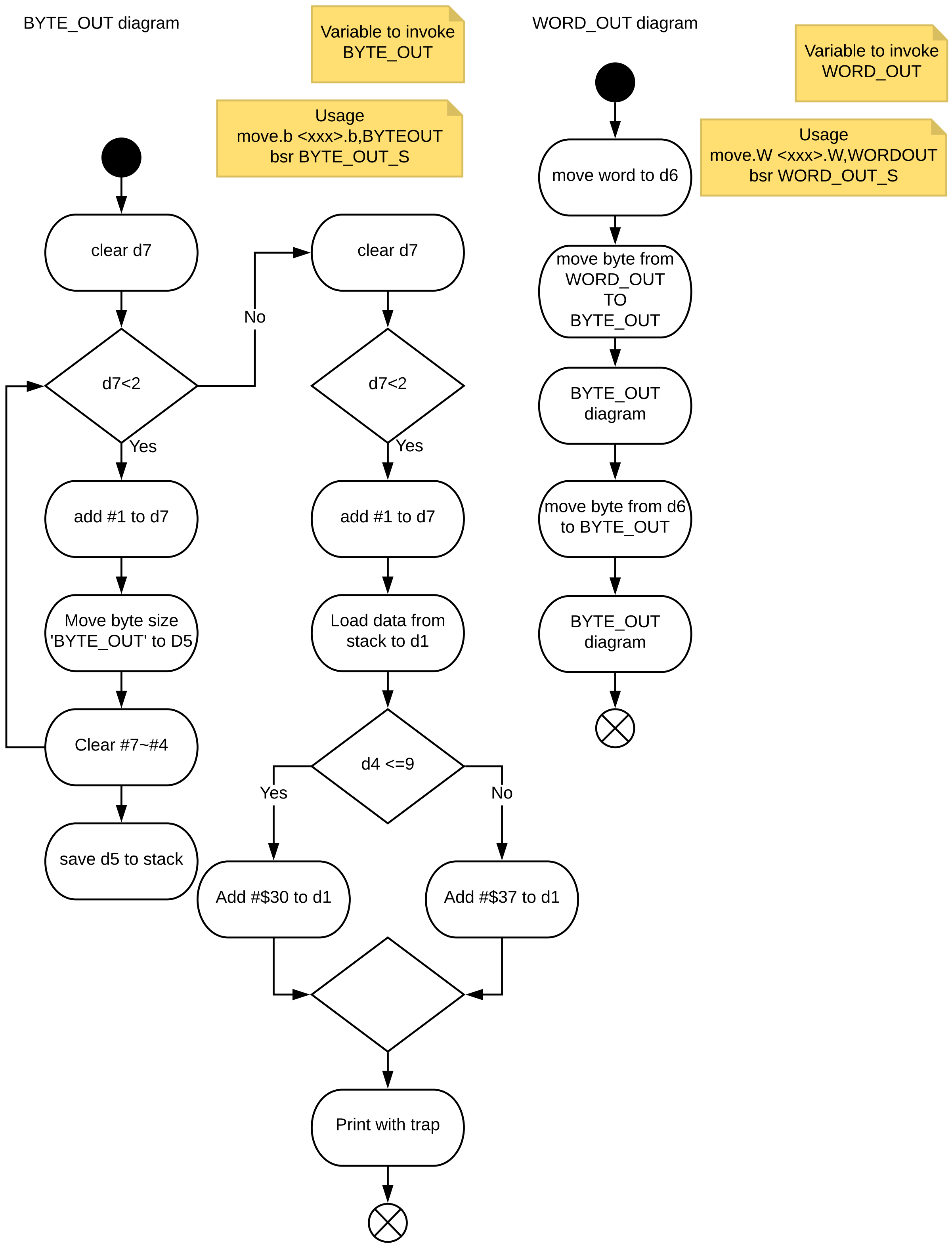
Check REGISTER_LIST_MASK
Invalid value
Invlalid_s diagram
Valid value
Check INITIAL_INSTRUCTION(movem)'s #10 bit
<list>,<ea> order
usage: bsr MCL_MM_LOAD_REGISTER_MASK
No
Initial Two ea load out diagram
Print out comma
Check #15 bit of REGISTER_LIST_MASK
Printout
Print 'D0'
Set need slash true
Check #14 bit of REGISTER_LIST_MASK
movem slash diagram
Print 'D1'
Check #13 bit of REGISTER_LIST_MASK
Print 'D2'
Check #12 bit of REGISTER_LIST_MASK
Print 'D3'
Check #11 bit of REGISTER_LIST_MASK
Print 'D4'
Check #10 bit of REGISTER_LIST_MASK
Print 'D5'
Check #9 bit of REGISTER_LIST_MASK
Print 'D6'
Check #8 bit of REGISTER_LIST_MASK
Print 'D7'
Check #7 bit of REGISTER_LIST_MASK
Print 'A0'
Check #6 bit of REGISTER_LIST_MASK
Print 'A1'
Check #5 bit of REGISTER_LIST_MASK
Print 'A2'
Check #4 bit of REGISTER_LIST_MASK
Print 'A3'
Check #3 bit of REGISTER_LIST_MASK
Print 'A4'
Check #2 bit of REGISTER_LIST_MASK
Print 'A5'
Check #1 bit of REGISTER_LIST_MASK
Print 'A6'
Check #0 bit of REGISTER_LIST_MASK
Print 'A7'
Print out comma
Initial Two ea load out diagram
branch to main loop
Check #0 bit of REGISTER_LIST_MASK
Print 'D0'
Check #1 bit of REGISTER_LIST_MASK
Print 'D1'
Check #2 bit of REGISTER_LIST_MASK
Print 'D2'
Check #3 bit of REGISTER_LIST_MASK
Print 'D3'
Check #4 bit of REGISTER_LIST_MASK
Print 'D4'
Check #5 bit of REGISTER_LIST_MASK
Print 'D5'
Check #6 bit of REGISTER_LIST_MASK
Print 'D6'
Check #7 bit of REGISTER_LIST_MASK
Print 'D7'
Check #8 bit of REGISTER_LIST_MASK
Print 'A0'
Check #9 bit of REGISTER_LIST_MASK
Print 'A1'
Check #10 bit of REGISTER_LIST_MASK
Print 'A2'
Check #11 bit of REGISTER_LIST_MASK
Print 'A3'
Check #12 bit of REGISTER_LIST_MASK
Print 'A4'
Check #13 bit of REGISTER_LIST_MASK
Print 'A5'
Check #14 bit of REGISTER_LIST_MASK
Print 'A6'
Check #15 bit of REGISTER_LIST_MASK
Print 'A7'
branch to main loop

movem slash diagram



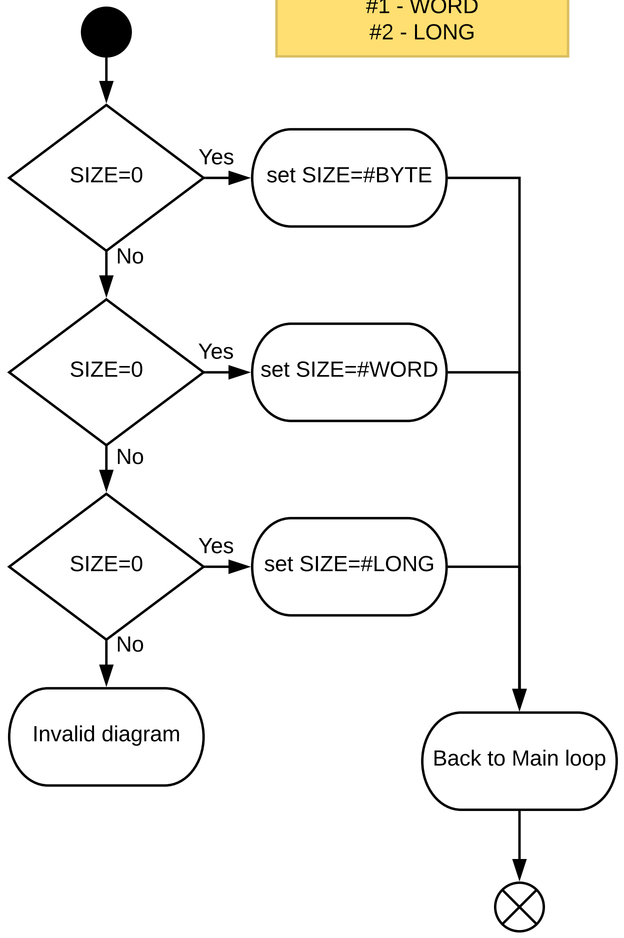
ADDRESS_READ_DECISION_LOAD


BSR
ADDRESS_READ_DECISION_LOAD
Base on
SRC_REGISTER
SRC_MODE
DEST_REGISTER
DEST_MODE
DECIDE HOW MANY MEMORY TO READ MORE AND SAVE TO
SRC_NUMBERDATA
DST_NUMBERDATA
Default size instraction
#$01 - BYTE
#$11 - WORD
#$10 - LONG
Compare SRC_MODE = #7
Yes
Compare SRC_REGISTER=0
No
Compare SRC_REGISTER=1
No
Compare SRC_REGISTER=4
Yes
Increment disassemble number by 2
Yes
Increment disassemble number by 4
Yes
Check Size is 'BYTE (01)'
No
Check Size is 'WORD (11)'
No
Check Size is 'LONG (10)'
Yes
Increment disassemble number by 2
Yes
Increment disassemble number by 2
Yes
Increment disassemble number by 4
Compare SRC_MODE = #7
No
Yes
Compare DST_REGISTER=0
No
Compare DST_REGISTER=1
No
Compare DST_REGISTER=4
Yes
Increment disassemble number by 2
Yes
Increment disassemble number by 4
Yes
Check Size is 'BYTE (01)'
No
Check Size is 'WORD (11)'
No
Check Size is 'LONG (10)'
Yes
Increment disassemble number by 2
Yes
Increment disassemble number by 2
Yes
Increment disassemble number by 4


BYTE_OUT diagram
Variable to invoke BYTE_OUT
Usage
move.b <xxx>.b,BYTEOUT
bsr BYTE_OUT_S
clear d7
d7<2
No
Yes
add #1 to d7
Move byte size 'BYTE_OUT' to D5
Clear #7~#4
save d5 to stack
clear d7
d7<2
Yes
add #1 to d7
Load data from stack to d1
d4 <=9
Yes
No
Add #$30 to d1
Add #$37 to d1
Print with trap
WORD_OUT diagram
Variable to invoke WORD_OUT
Usage
move.W <xxx>.W,WORDOUT
bsr WORD_OUT_S
move word to d6
move byte from WORD_OUT TO BYTE_OUT
BYTE_OUT diagram
move byte from d6 to BYTE_OUT
BYTE_OUT diagram

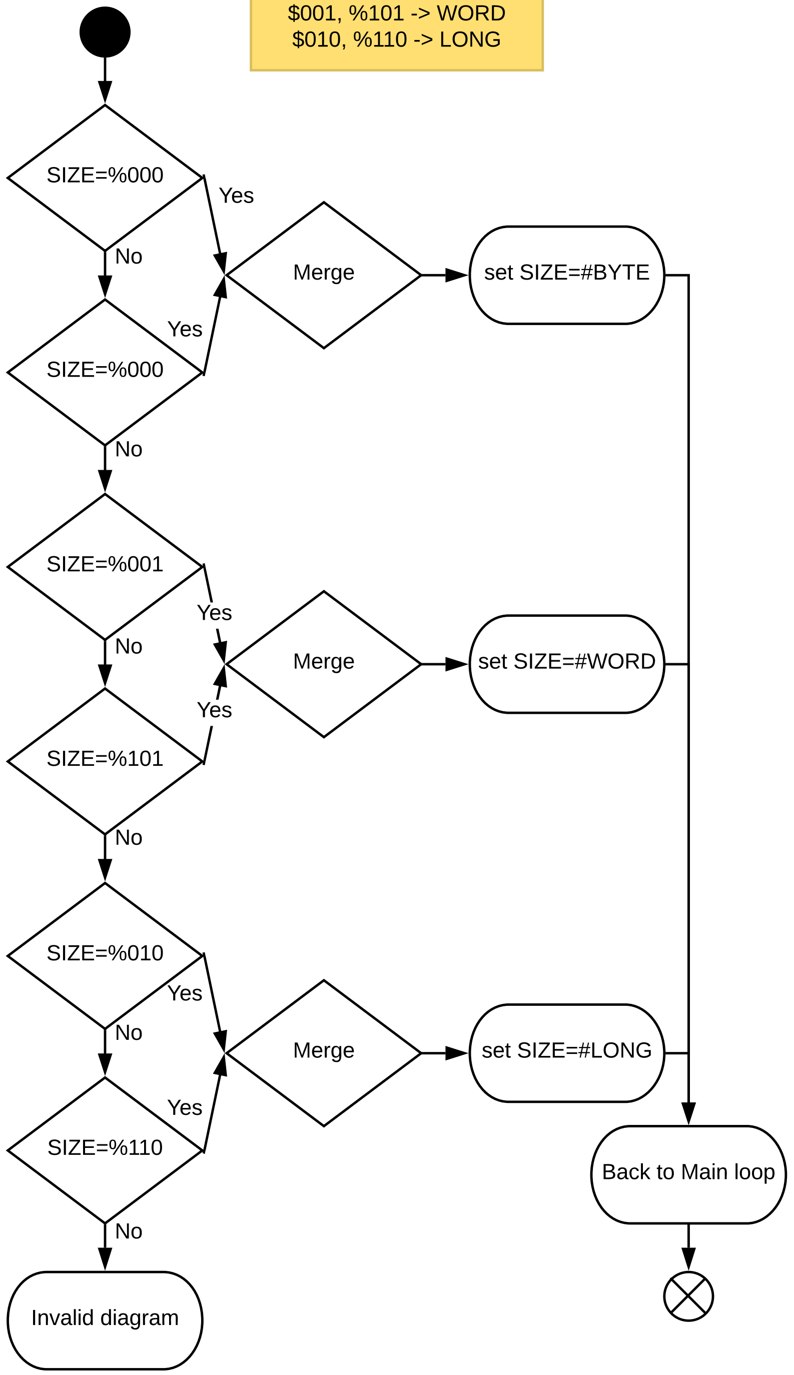
Size Convertor type 1

Size Convertor type 1
#0 - BYTE
#1 - WORD
#2 - LONG



Size Convertor type 2

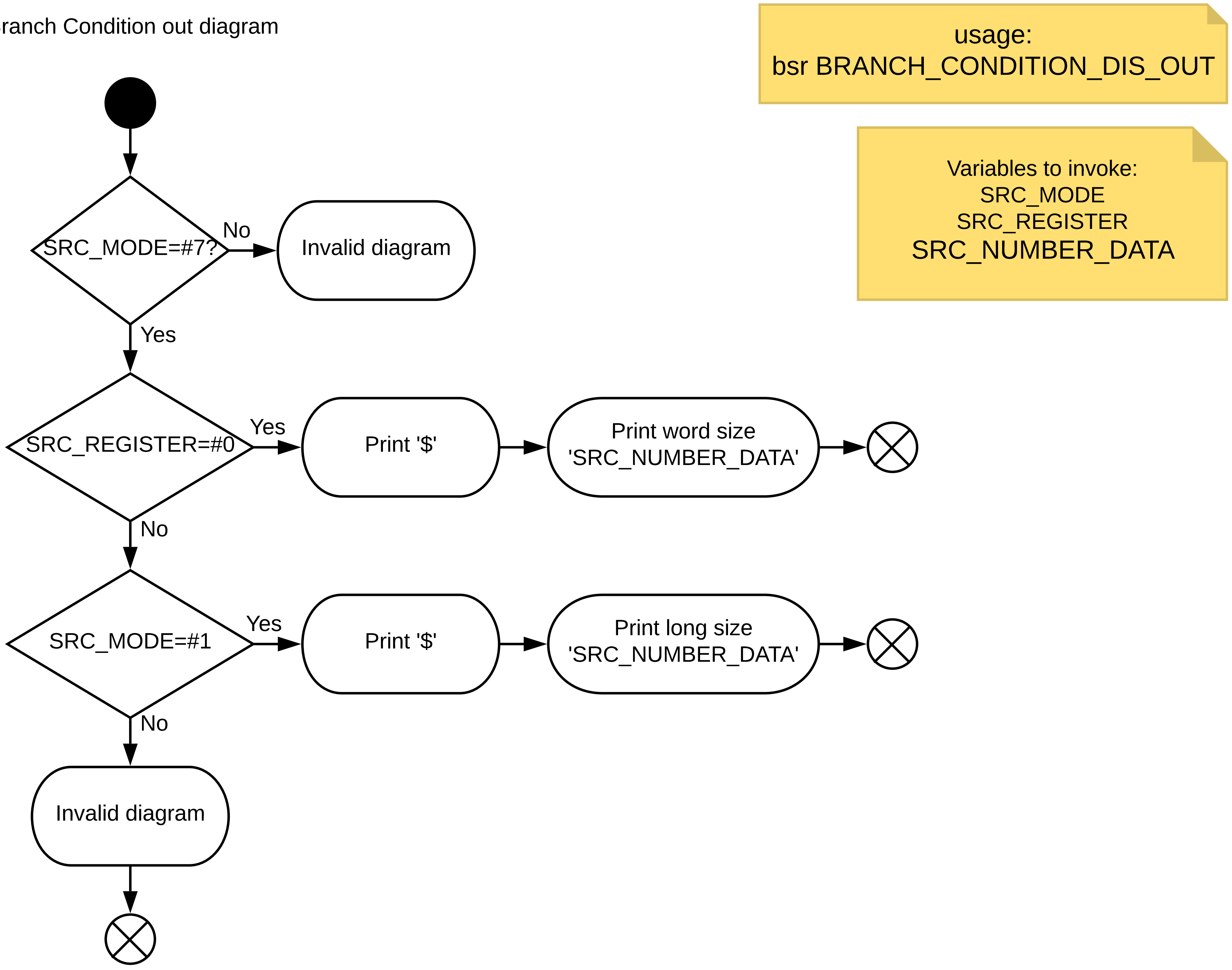
Size convertor type 2
$000, %100 -> BYTE
$001, %101 -> WORD
$010, %110 -> LONG



Default size instrauction
#$01 - BYTE
#$11 - WORD
#$10 - LONG

Variable to invoke
SIZE
CONST VARIABLE
BYTE
WORD
LONG

Branch Condition out diagram


usage:
bsr BRANCH_CONDITION_DIS_OUT
Variables to invoke:
SRC_MODE
SRC_REGISTER
SRC_NUMBER_DATA
SRC_MODE=#7?
No
Invalid diagram
Yes
SRC_REGISTER=#0
Yes
Print '$'
Print word size 'SRC_NUMBER_DATA'
No
SRC_MODE=#1
Yes
Print '$'
Print long size 'SRC_NUMBER_DATA'
No
Invalid diagram