

EECE416 :Microcomputer Fundamentals and Design

68000 Instruction Sets and Groups

Instruction Groups

- ⌘ Data Transfer Groups
- ⌘ Arithmetic Group
- ⌘ Logical Group
- ⌘ Shift and Rotate Group
- ⌘ Bit Manipulation Group
- ⌘ Binary Coded Decimal (BCD) Group
- ⌘ Program Control Group
- ⌘ System Control Group

Data Transfer Group

⌘ EXG: Exchange Register

⊗ EXG D3, D5

⊗ EXG A3, A5

⌘ MOVE.S (S: B, W, L)

⌘ MOVEA.L or MOVEA.W

⌘ MOVEM: Move Multiple Registers

⊗ MOVEM.L D0-D3, (A0)

⊗ MOVEM.L (A0), D0-D3

⌘ MOVEQ: Move Quick (sign-ext. 8-bit)

⊗ Moveq #\$B7, D4

⌘ SWAP: Swap Reg. Halves (Word)

⊗ SWAP D5 (with D5:3cff9100 ->91003cff)

Arithmetic Group

- ⌘ ADD: add binary
 - ⊗ Add.b, Add.w, Add.L
- ⌘ ADDA: add address
 - ⊗ Adda.w, adda.l
- ⌘ ADDI: add immediate (B, W, L)
 - ⊗ Addi # $\$10$,D2
- ⌘ ADDQ: add quick (data range 1 – 8)
- ⌘ CLR: clear operand (B, W, L)
 - ⊗ CLR D0
 - ⊗ CLR A2
 - ⊗ CLR TABLE
- ⌘ CMP: compare data (B, W, L) (DST-SRC and check Flag)
 - ⊗ CMP.W # $\$29AF$, D6 ;with D6=485C29AF
 - ⊗ Z flag?
- ⌘ CMPA: compare address (W [sign ext], L)
- ⌘ CMPI: compare immediate
 - ⊗ CMPI.W #5, (A3)

Arithmetic Group

- ⌘ DIVS: signed division (signed 32-bit (DST) divided by signed 16-bit (SRC))
 - ⊗ DST must be a data register
 - ⊗ After instruction: DST [UW(Remainder)+LW (Quotient)]
 - ⊗ DIVS D2, D3
- ⌘ DIVU: unsigned division (unsigned 32-bit (DST) divided by unsigned 16-bit (SRC))
 - ⊗ DST must be a data register
 - ⊗ After instruction: DST [UW(Remainder)+LW (Quotient)]
 - ⊗ DIVU D2, D3
- ⌘ MULS: signed multiplication (16-bit x 16-bit)
 - ⊗ DST must be data register
 - ⊗ MULS D4, D5
- ⌘ MULU: unsigned multiplication (16-bit x 16-bit)
 - ⊗ DST must be data register
 - ⊗ MULU D4, D5

Arithmetic Group

⌘ NEG: negate (2's complement of the DST) (B, W, L)

⊗ NEG.B D2

⌘ SUB: subtract binary (B, W, L) (DST-SRC → DST)

⊗ One of the Operands must be a Data Register

⊗ SUB.W D0, D1

⌘ SUBA: subtract Address (W[sign ext], L)

⌘ SUBI: subtract immediate (B, W, L)

⊗ SUBI.B #2C, D2

⌘ TAS: Test and set an operand

⊗ Lower 8-bits tested (CCR affected) → CCR Affected

⊗ Then, bit 7 is set

⊗ TAS D5; if D5=2CC3E500 → Z=1, and new D5=2CC3580

⌘ TST: Test an Operand

⊗ Same as TAS, except that DST is not changed

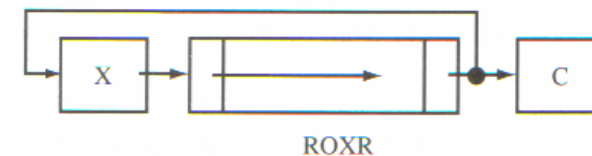
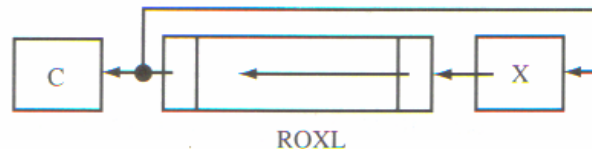
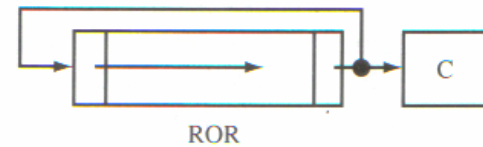
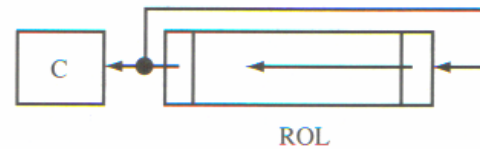
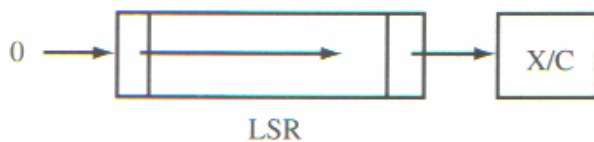
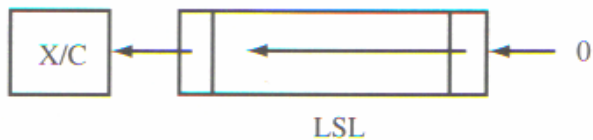
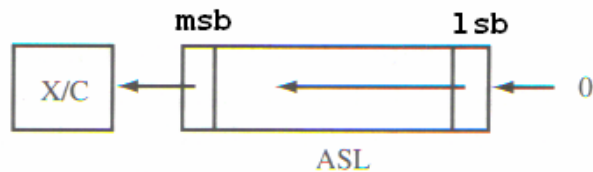
⊗ CCR affected

Logical Group

- ⌘ AND: AND logical (B, W, L)
 - ⊗ One of the Operands must be a data register
 - ⊗ `AND.W D0, D1`
- ⌘ ANDI: AND immediate (B, W, L)
 - ⊗ `ANDI.B #$F0, D7`
 - ⊗ What is this operation for?
- ⌘ OR: inclusive OR logical (B, W, L)
 - ⊗ `OR.L D3, D4`
- ⌘ ORI: OR immediate (B, W, L)
 - ⊗ How can the MSB of the lower byte of D1 be set?
 - ⊗ `ORI.B #$80, D1`
- ⌘ EOR: exclusive OR logical (B,W,L)
 - ⊗ `EOR.W D2, D3`
- ⌘ EORI: EOR immediate (B, W, L)
- ⌘ NOT: logical complement (B, W, L)
 - ⊗ `NOT.B D1` ;1's complement

Shift and Rotate Group

- ⌘ ASL, ASR, LSL, LSR, ROL, ROR, ROXL, ROXR (B, W, L)
- ⌘ Direction (Shift/Rotate) and Number
- ⌘ Use this for Data Register
- ⌘ Can be used for Memory (but with W only)
 - ⊗ ASL.B #4, D2
 - ⊗ LSR.W #6, D1
 - ⊗ ROL.L D4, D5 ;D4 has the count
 - ⊗ ROXR.B #5, D4



Bit Manipulation Group

⌘ BCHG: test a bit and change (B, L) Z flag

- ⊗ A bit is tested to see if it is 0 → if so Z=1
- ⊗ Then, the bit is complemented
- ⊗ BCHG.B #3, D1
- ⊗ If the bit in question is position from 8 to 31, the position must be placed into a data register
- ⊗ BCHG.L D0, D1

⌘ BCLR: test a bit and clear (Z flag)

- ⊗ BCLR D6, D7

⌘ BSET: test a bit and set (Z flag)

- ⊗ BSET #2, (A3)

⌘ BTST: test a bit (Z flag)

- ⊗ BTST #5, D4

Program Control Group

⌘ 1. Conditional Branch Instructions

☒ Signed Comparison

- ☒ BGT: branch if Greater Than ($>$)
- ☒ BGE: branch if Greater or Equal (\geq)
- ☒ BEQ: branch if Equal ($=$)
- ☒ BNE: branch if Not Equal (\neq)
- ☒ BLE: branch if Lower or Equal (\leq)
- ☒ BLT: branch if Lower Than ($<$)

☒ Unsigned Comparison

- ☒ BHI: branch if Higher ($>$)
- ☒ BCC: branch if Carry Clear (\geq)
- ☒ BEQ: branch if Equal ($=$)
- ☒ BNE: branch if Not Equal (\neq)
- ☒ BLS: branch if Lower or Same (\leq)
- ☒ BCS: branch if Carry Set ($<$)

☒ SUBQ #1, D0

BNE AGAIN ; branch if Z=0

Program Control Group

⌘ 2. Decrement-and-Branch Instruction

- ⊗ DBT: Decrement and branch if True
- ⊗ DBF: Decrement and branch if False
- ⊗ DBHI: Decrement and branch if High
- ⊗ DBLS, DBCC, DBCS, DBNE, DBVC, DBVS, DBPL
- ⊗ DBMI, DBGE, DBLT, DBGTE, DBLE
- ⊗ (ex)DBCS D5, TABLE

⊗ Condition Code Tests

Condition	Meaning	Flag Tested
T	True	None
F	False	None
HI	High	C+Z=0
LS	Lower or Same	C+Z=1
CC	Carry Clear	C=0
CS	Carry Set	C=1
NE	Not Equal	Z=0
EQ	Equal	Z=1
VC	Overflow Clear	V=0
VS	Overflow Set	V=1
PL	Plus	N=0
MI	Minus	N=1
GE	Greater or equal	N⊕V=0
LT	Less Than	N⊕V=1
GT	Greater Than	Z+(N⊕V)=0
LE	Less or equal	Z+(N⊕V)=1

Program Control Group

⌘ 3. Set-according-to-Condition

- ⊗ ST: Set if True
- ⊗ SF: Set if False
- ⊗ SHI, SLS, SCC, SNE, SEQ, SVC, SVS, SPL, SMI, etc
- ⊗ If true, the DST byte is set
- ⊗ If false, the DST byte is cleared
- ⊗ (ex) SEQ D3

⌘ 4. BRA (Branch Always)

- ⊗ Unconditional 32KB space

⌘ 5. BSR (Branch to Subroutine)

- ⊗ Return address is stored at the Stack (I.e., A7)

⌘ 6. JMP (jump)

- ⊗ Entire address space unconditional

⌘ 7. JSR (jump to subroutine)

- ⊗ Entire Memory Space

⌘ 8. RTR (return and restore condition codes)

⌘ 9. RTS (return from subroutine)