Chapter 3

Introduction to 68000 Instruction Sets

Expected Outcomes

- Explain the type of addressing modes in the 68000
- Compare the different instruction sets in the 68000

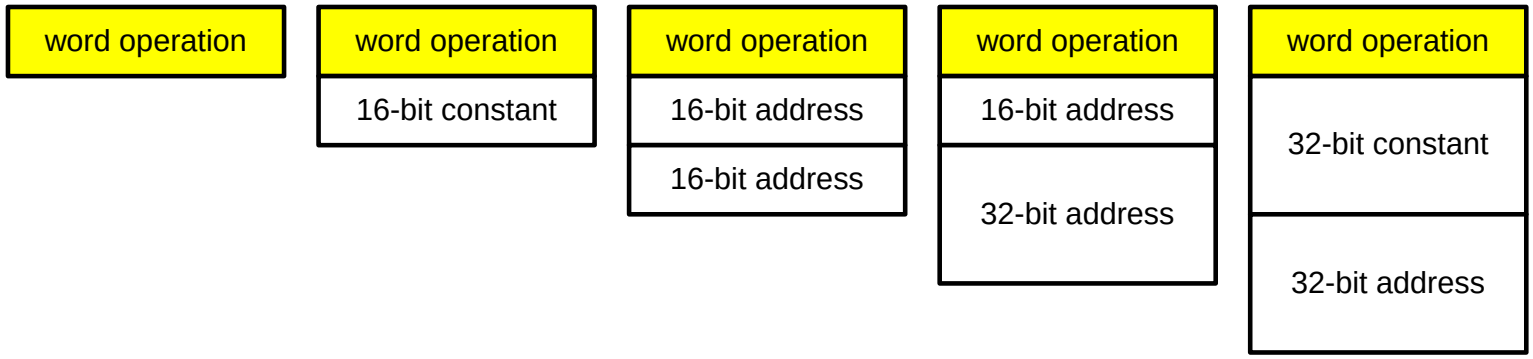
Introduction

- There are two type of instruction sets
- **Machine code**
  - String of bits, complex and the only code understand by CPU
  - Can be 2 – 10 byte size for each instruction
- **Assembly code**
  - Simple and easy to understand
  - 54 basic instructions but can be more than 1000 if all variations are considered

Machine Code	Assembly
0001110010100011	CLR.W D0
1110011011100011	MOVE.B $100, D2
1110011001101010	EXG D2, A1
1010000110101001	MULS D1, D7
1100001110010101	ADD.L #$5463,D3
0011111001101011	

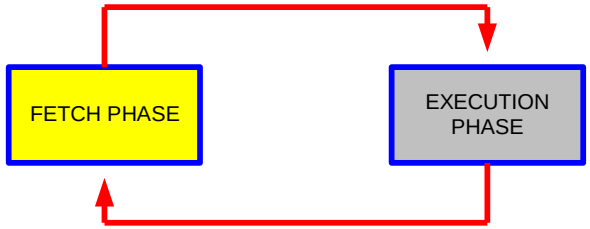
Machine Code Instruction

- The instruction at least must be 2 byte long and not more than 10 byte long
- The first word is known as **word operation**:
  - Instruction type (add, multiply etc)
  - Data size (byte, word or longword)
  - Numbers of byte in the instruction
  - Additional information (operand) to access data



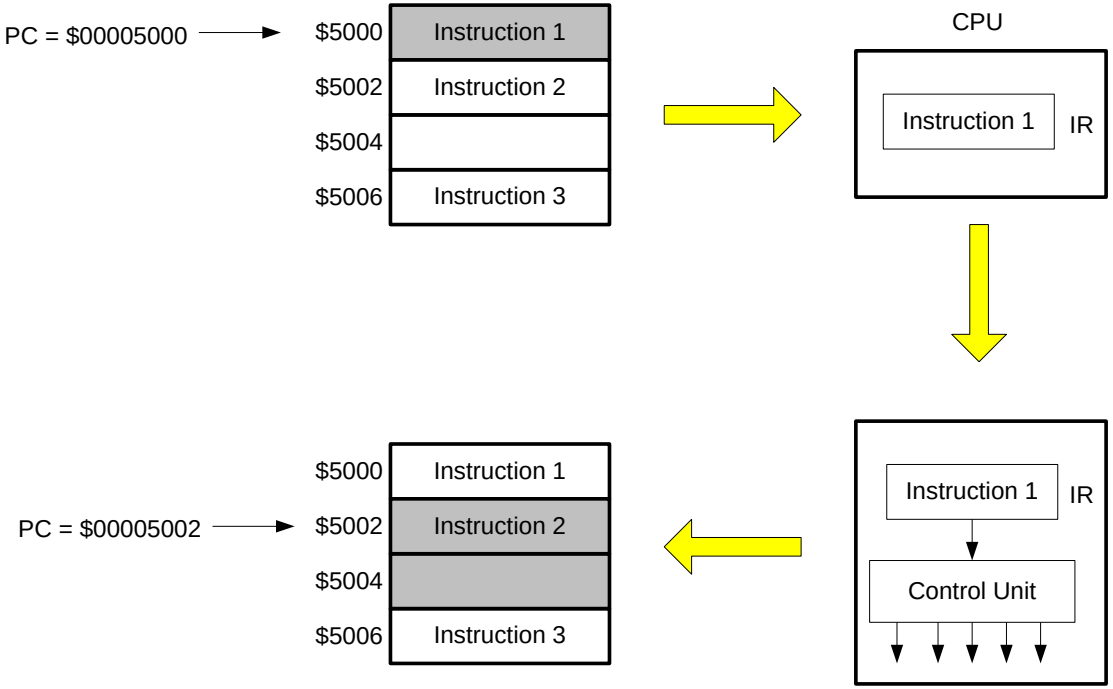
Instruction Cycle

- Microprocessor executes program continuously
- Two main phases
- **Fetch phase**
  - PC is pointed the address of instruction
  - Fetch the instruction from memory and place in IR
  - Control unit translate the instruction as it updates the PC
- **Execution phase**
  - Calculate the operand address
  - Fetch the operand
  - Execute the operation
  - Store the result
  - Return to fetch phase



Instruction Cycle

- PC always points the address of next instruction



Basic Instruction Sets

- Type of instruction sets
  - **Data Movement** such as `MOVE`, `MOVEA` and `MOVEQ`
  - **Integer Arithmetic** such as `ADD`, `MULU` and `DIVU`
  - **Boolean** such as `AND`, `EORI` and `NOT`
  - **Shift & Rotate** such as `ROL`, `ASL` and `LSR`
  - **Bit Manipulation** such `BSET`, `BCHG` and `BCLR`
  - **Binary Coded Decimal** such as `ABCD` and `NBCD`
  - **Program Flow** such as `BRA`, `JMP` and `BNE`
  - **System Control** such as `TRAP`

Data Movement

Instruction	Description	Example
MOVE	Copy an 8-,16- or 32-bit from one location/register to another location/register	`MOVE.B D0,D2` `MOVE.L #40,$1000`
MOVEA	Copy a source operand to an address reg. Only word or longword operation. Word operation will be sign-extended	`MOVEA.W #$400,A2` `MOVEA.L D0,A4`
MOVEQ	Copy 8-bit signed number to any data register. The data to be copied will be sign-extended to 32-bit	`MOVEQ #56,D4` `MOVEQ #-44,D7`
MOVEM	Transfers a group of registers specified by a list. Operates only on word or longword operation	`MOVEM D0-D3,-(A7)`

Integer Arithmetic

Instruction	Description	Example

ADDx
SUBx
	Add/Subtract the content of source to/from the content of destination and store the result in destination. One of the operand must be Data register	`ADD.B D0,D4`
		`SUB.W #43,D7`
		`ADD.L $100,D2`

MULU **MULS**	Multiply two 16-bit operands and place the result in destination (32-bit). One of the operand must be Data register. MULU is for unsigned number and MULS is for signed number	`MULU #$234,D3` `MULS $565,D6`
DIVU **DIVS**	For 32-bit dividend and 16-bit divisor where the former must be data register. The result of division is stored in low word of destination and the remainder in upper word of the destination	`DIVU D3,D5` `DIVS #$523,D3`

Integer Arithmetic

Instruction	Description	Example
EXT	Signed extended for data register to word or longword. `EXT.W` is sign extended from 8-bit to 16-bit where `EXT.L` is sign extended from 16-bit to 32-bit	`EXT.W D0` `EXT.L D3`

CLR	To place zeros to destination (memory location or data register). Invalid for address register or immediate mode	`CLR.B D6`
		`CLR.W $1234`
		`CLR.L $34568`

NEG	Obtain 2's complement the content of destination	`NEG.W D2` `NEG.B D7`

Boolean Operation

Instruction	Description	Example
AND **ANDI**	Bit-wise AND operation. Normally, it is used to clear certain bit of destination	`AND.B D2,D3` `ANDI.W #$3234,D7`
OR **ORI**	Bit-wise OR operation. Normally, it is used to set certain bit of destination	`OR.L D4,D7` `ORI.B #%1001,D0`
EOR **EORI**	Bit-wise exclusive OR operation. Normally used to toggle certain bit of destination	`EOR.B D1,D2` `EORI.L #$2344,D7`
NOT	1's complement operation	`NOT.L D3`

Bit Manipulation

Instruction	Description	Example
BSET	Bit test and set. The operation causes the Z-bit to be set if the specified bit is zero and forces the specified bit to be set	`BSET.L #30,D3`
BCLR	Bit test and clear. The operation causes the Z-bit to be set if the specified bit is zero and forces the specified bit to be clear	`BCLR.B #7,$1000`
BCHG	Bit test and change. The operation causes the Z-bit to be set if the specified bit is zero and toggle the specified bit	`BCHG.L D2,D6`
BTST	Bit test operation. The operation causes the Z-bit to be set if the specified bit is zero and the Z-bit to be clear is the specified bit is set	`BTST.B #4,D5`

Shift & Rotate

Instruction	Description	Example
ASL	Arithmetic Shift Left	`ASL.B #3,D3`
ASR	Arithmetic Shift Right	`ASR.L D2,D7`
LSL	Logical Shift Left	`LSL.L #6,D6`
LSR	Logical Shift Right	`LSR.L D2,D0`
ROR	Rotate to the right	`ROR.B #1,D0`
ROL	Rotate to the left	`ROL.L D2,D6`

Binary Coded Decimal

Instruction	Description	Example
ABCD	Add source operand and X-bit with the content of destination using BCD arithmetic. Only BYTE operation	`ABCD D6,D7`
SBCD	Subtract source operand and X-bit from the content of destination using BCD arithmetic. Only BYTE operation	`SBCD D4,D3`
NBCD	Subtract the destination operand and the X-bit from zero	`NBCD D0`

Program Flow

Instruction	Description	Example
BRA.S **BRA.L**	Unconditional branch that place PC to a new location. The offset is limited to 8- and 16-bit only	`BRA.S LABEL`
JMP	Like `BRA`, it is unconditional branch. It is relative addressing that allows the program to jump to any location in 68000 memory map	`JMP LABEL`
Bcc	Conditional Branch that is based on CCR flag	`BNE LABEL` `BCC LABEL`
BSR **JSR** **RTS**	`BSR(JSR)` is used to branch to subroutine. `RTS` is required at the end of subroutine to allow the program to return to main program	`BSR LABEL`

System Control

Instruction	Description
TRAP	It performs three operations. (1) Push PC and SR onto the stack (2) sets the execution mode to supervisor mode (3) load the PC with a new value from a vector table
STOP **RESET**	`STOP` load the SR with the immediate operand and stops the CPU. `RESET` asserts the CPU's RESET line for 124 cycles. If `STOP` and `RESET` are executed in user mode, a privilege violation occurs

Basics Instruction Sets

- Example of instruction sets

```
ABCD  Add decimal with extend
ADD   Add binary
AND   Logical AND
ASL   Arithmetic Shift Left
ASR   Arithmetic Shift Right
Bcc   Branch Conditionally
BCHG  Bit Test & Change
BCLR  Bit Test & Clear
BRA   Branch Always
BSET  Bit Test & Set
BSR   Branch to Subroutine
BTST  Bit Test
CHK   Check Register with Bounds
CLR   Clear Operands
```

Basics Instruction Sets

- Example of instruction sets (continues….)

CMP	*Compare*
DBcc	*Decrement & Branch Conditionally*
DIVS	*Signed Divide*
DIVU	*Unsigned Divide*
EOR	*Exclusive OR*
EXG	*Exchange Registers*
EXT	*Sign Extend*
JMP	*Jump to Effective Address*
LEA	*Load Effective Address*
LINK	*Link Stack*
LSL	*Logical Shift Left*
LSR	*Logical Shift Right*
MOVE	*Move Source to Destination*

Basics Instruction Sets

- Example of instruction sets (continues….)

MULS	*Sign Multiply*
MULU	*Unsigned Multiply*
NBCD	*Negate Decimal with Extend*
NEG	*Negate*
NOP	*No Operation*
NOT	*One's Complement*
OR	*Logical OR*
PEA	*Push Effective Address*
RESET	*Reset External Device*
ROL	*Rotate Left*
ROR	*Rotate Right*
ROXL	*Rotate Left Through Extend*
ROXR	*Rotate Right Through Extend*

Basics Instruction Sets

- Example of instruction sets (continues…)

```
RTE     Return from Exception
RTR     Return & Store
RTS     Return from Subroutine
SBCD    Subtract Decimal with Extend
Scc     Set Conditionally
STOP    Stop Processor
SUB     Subtract Binary
SWAP    Swap Data Register Halves
TAS     Test & Set Operand
TRAP    Trap
TRAPV   Trap on Overflow
```

Variation of Instruction Set

- Some of examples variation instruction set
  - **ADD** such as `ADDI, ADDQ, ADDA, ADDX`
  - **CMP** such as `CMPI, CMPA, CMPM`
  - **MOVE** such as `MOVEQ, MOVEA, MOVEM, MOVEP`
  - **SUB** such as `SUBA, SUBQ, SUBI, SUBX`

Assembly Language

- General format

```
<label>     opcode<.field> <operand> <;comment>
```

- Instruction Format for assembly language consists of
  - **Label** – pointer to the instruction's memory location
  - **Opcode** - Operation code such as `MOVE, CLR`
  - **Field** – Define width of operand (B, W, L)
  - **Comment** – For documentation purposes
  - **Operands** – Data/address use in operation (source/destination)
    - There may be no operand or 1 operand or 2 operands

Operand

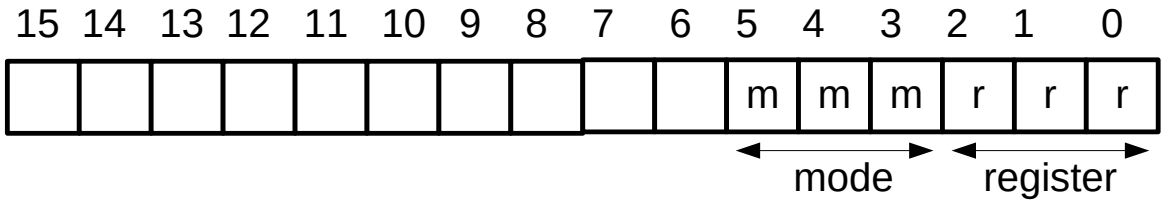
- Operand can be
  - Registers such as D0, D5, A1, A7
  - Constant
  - Memory location
- Examples of operand with basic addressing modes
  - Data register, Dn `MOVE.B D2,D4`
  - Register Indirect An `CLR.B (A0)`
  - Immediate `MOVE.L #$100,D2`
  - Absolute `CLR.L $1000`
- Immediate operand can be specified in several format such as hexadecimal, binary or even ASCII

Operand

Instruction	Comment
`BRA REPEAT`	PC branch to label called REPEAT
`NOP`	No operand
`CLR.W`	1 operand (destination)
`ADD.B #100,D7`	2 operand 100 is data source, D7 is destination
`MOVE.L D2, D4`	2 operand D2 is source, D4 is destination

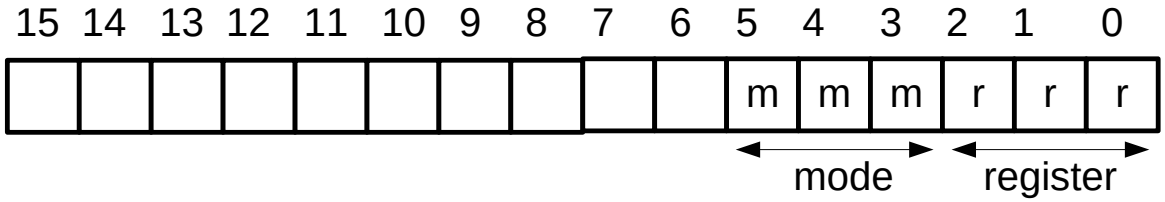
Effective Address

- Effective Address is the actual address of operand
- The value of effective address is determined by the address mode of the instruction
- The address mode determines the number of byte used in each instruction
- The effective address is composed of two 3-bit fields: the mode field and the register field



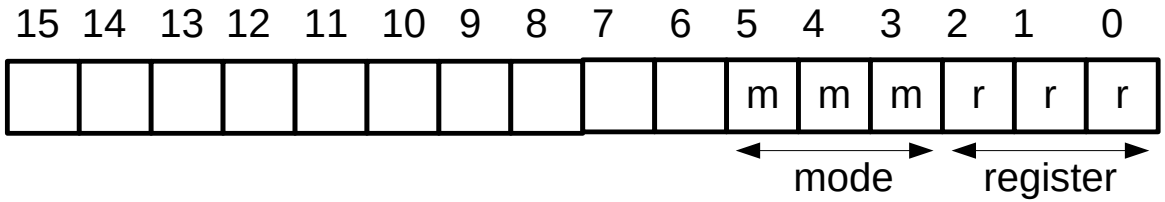
Effective Address

Mode	Register	Address Mode	Syntax
000	rrr	Direct data register	`Dn`
001	rrr	Direct address register	`An`
010	rrr	Address Register Indirect	`(An)`
011	rrr	ARI with post-decrement	`(An)+`
100	rrr	ARI with pre-decrement	`-(An)`
101	rrr	ARI with displacement	`N(An)`
110	rrr	ARI with index & displacement `N`	`(An,Xm)`



Effective Address

Mode	Register	Address Mode	Syntax
111	000	Absolute short	$XXXX
111	001	Absolute Long	$XXXXXXXX
111	010	PC with displacement	N(PC)
111	011	PC with Index	N(PC, Xm)
111	100	Immediate	#$XXXX

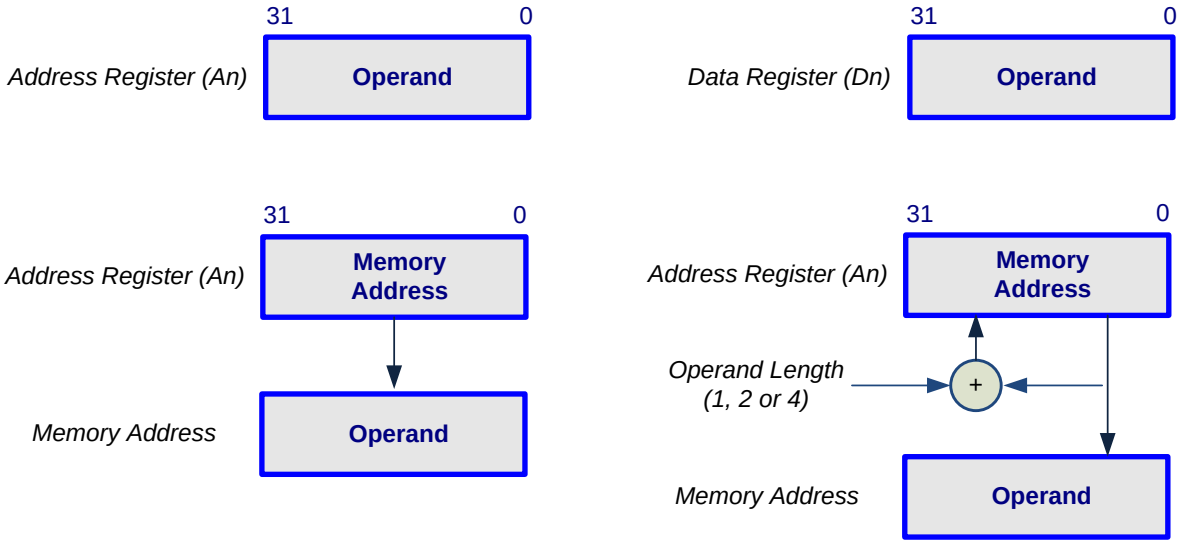


Addressing Modes

- **Absolute Short** – specifies 16-bit address in a single extension word and it is sign-extended to 32-bit during execution
- **Absolute Long** – requires 32-bit to specify 24-bit address
- **Immediate** – requires the instruction include the operand as integral part itself
- **Direct Data Register** – the operand is data register itself
- **Direct Address Register** – the operand is address register itself
- **Address Register Indirect** – use address register as a pointer to a target memory location
- **Address Register Indirect with Pre-decrement** – use address register as a pointer and decrement accordingly to point a target memory location

Addressing Modes

- Direct Data Register
- Direct Address Register
- Address Register Indirect
- Address Register Indirect with Post-increment

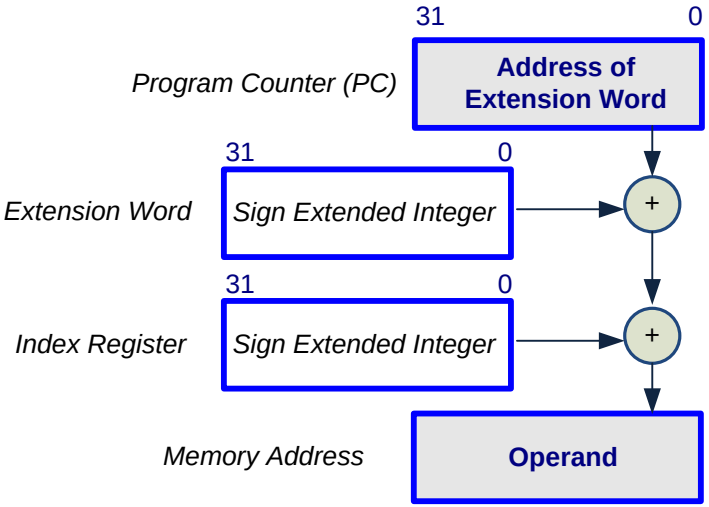


Addressing Modes

- **Address Register Indirect with Post-increment** – use address register as a pointer to point a target memory location and increment accordingly
- **Address Register Indirect with Index and displacement** – use address register, data register/address register and 8-bit offset as a pointer to point a target memory location
- **Address Register Indirect with displacement** – use address register and 16-bit offset as a pointer to point a target memory location
- **Program Counter with displacement** – use PC and 16-bit offset as a pointer to point a target memory location
- **Program Counter with index**– use PC and index as a pointer to point a target memory location

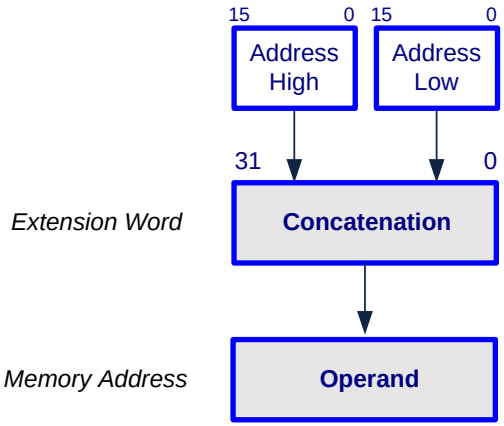
Addressing Modes

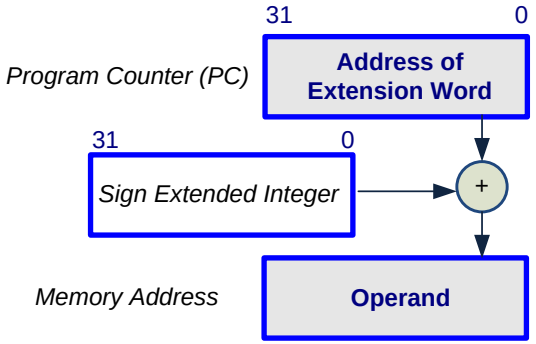
Program Counter with index



Addressing Modes

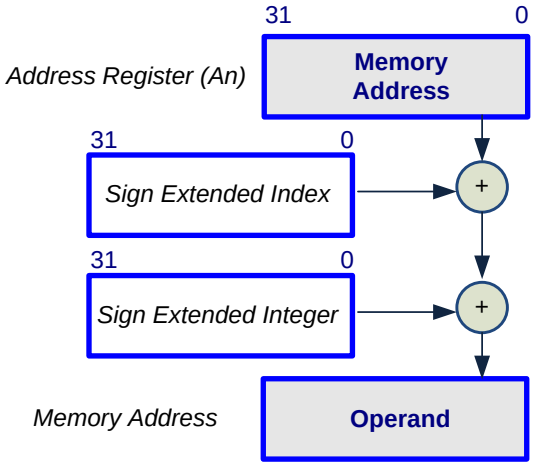
- Absolute Long
- Program Counter with displacement

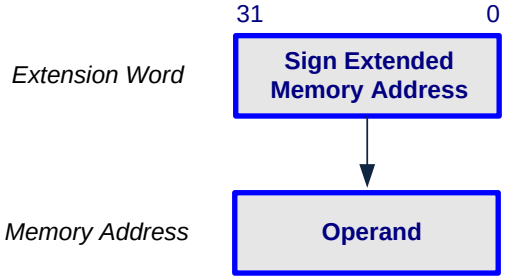




Addressing Modes

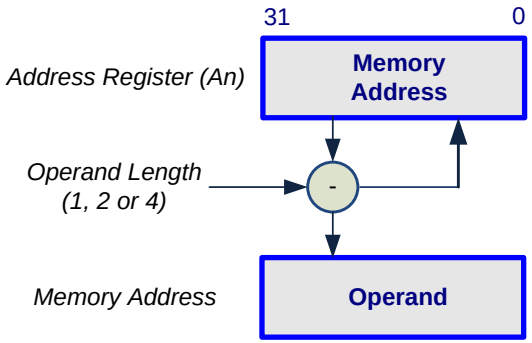
- Absolute Short
- Address Register Indirect with Index and displacement

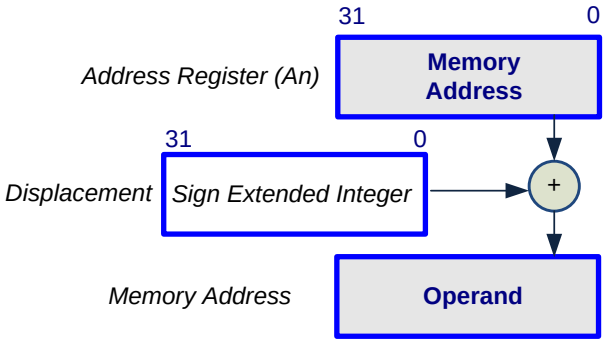




Addressing Modes

- Address Register Indirect with displacement
- Address Register Indirect with Pre-decrement





Self-Test

- **Exercise**
  Names and briefly explain the difference between addressing modes in MC68000
- **Exercise**
  Write the general format for instruction in MC68000 Assembly Language. Briefly explain the role for each format
- **Exercise**
  The size of each instruction set can be 2 – 10 bytes. Why?
- **Exercise**
  Briefly explain the type of instruction sets in MC68000.