

# SEE 3223 Microprocessor Systems

## 2: 68000 Architecture

Muhammad Mun'im Ahmad Zabidi (munim@utm.my)



# 68000 Architecture

## Aims

- To review the the architecture of the 68000 microprocessor.

## • Intended Learning Outcomes

- At the end of this module, students should be able to:
  - Briefly explain the history of microprocessor and the 68000 family
  - Describe the term programming model
  - Describe the programmer-visible registers in the 68000
  - Describe how the memory is accessed in the 68000
  - Describe and use the three simplest addressing modes of the 68000: direct, absolute and immediate
  - Be able to access and understand the information presented in the 68000 Programmer's Reference Manual

# 68000/ColdFire Background

- MC68000 introduced by Motorola in 1979.
- Notable sightings:
  - Used in the Sun, the first ever workstation
  - Used in the Macintosh, first ever GUI personal computer
  - Used in early versions of PalmPilot PDA
- Why 68000 for learning microprocessors?
  - Powerful & simple instruction set
  - Sophisticated interfacing capabilities
  - Able to support high-level language and operating systems
  - Flat memory map (versus segmented memory used in Intel 80x86)
  - The most popular  $\mu$ P in academia
- Internally, MC68000 has 32 bit data paths and 32-bit instructions
  - interfaces with external components using a 16-bit data bus. So a programmer considers it 32-bit chip while a system designer considers it a 16-bit chip.
  - Hence the “16-/32-bit chip” designation.
- The original 68000 was available in 64-bit DIP or 68-pin PLCC.



# 68000/ColdFire Versions

- 68000 family has many versions.
  - 680x0 means 68000, 68008, 68010, 68020, 68030, 68040 and 68060.
  - Newer versions are “upward compatible” with older versions.
  - The family is also affectionately called 68k or MC68k.
  - Most commonly found members are 68000, 68020, CPU32 and ColdFire.
- The family includes 16-bit peripherals chips.
  - The 68000 can use 68000-type peripherals chips for higher performance or older 6800-type peripherals for lower cost.
- ColdFire is the current version
  - ‘RISC’ ified 68000 processor core.
  - Smaller, less power used than normal 68020.
  - A ColdFire chip is an embedded processor with integrated peripheral
  - You can find it in some HP laserjet printers
- Today, the 68k family is made by **Freescale Semiconductors**.



# 68k Processor Family

	48-pin 68008 *	52-pin 68008 *	68000	68010 *	68020	68030	68040	68060
Data bus (bits)	8	8	8/16 **	16	32	32	32	32
Address Bus (bits)	20	22	24	24	32	32	32	32
Data cache (bytes)	-	-	-	-	-	256	4096	8192
Instruction cache (bytes)	-	-	-	-	256	256	4096	8192
Memory Management Unit	-	-	-	-	-	On-chip	On-chip	On-chip
Floating-Point Unit	-	-	-	-	Off-chip	Off-chip	On-chip	On-chip
Max Speed (MHz)	-	-	20	-	33	50	40	75
Performance (MIPS)	-	-	2	-	10	18	44	110

\* 68008 and 68010 are end-of-lifed (EOL) meaning no longer in production.

\*\* Original 68000 has 16-bit bus. Current 68000 has selectable 8- or 16-bit bus.

# 68000 Hardware

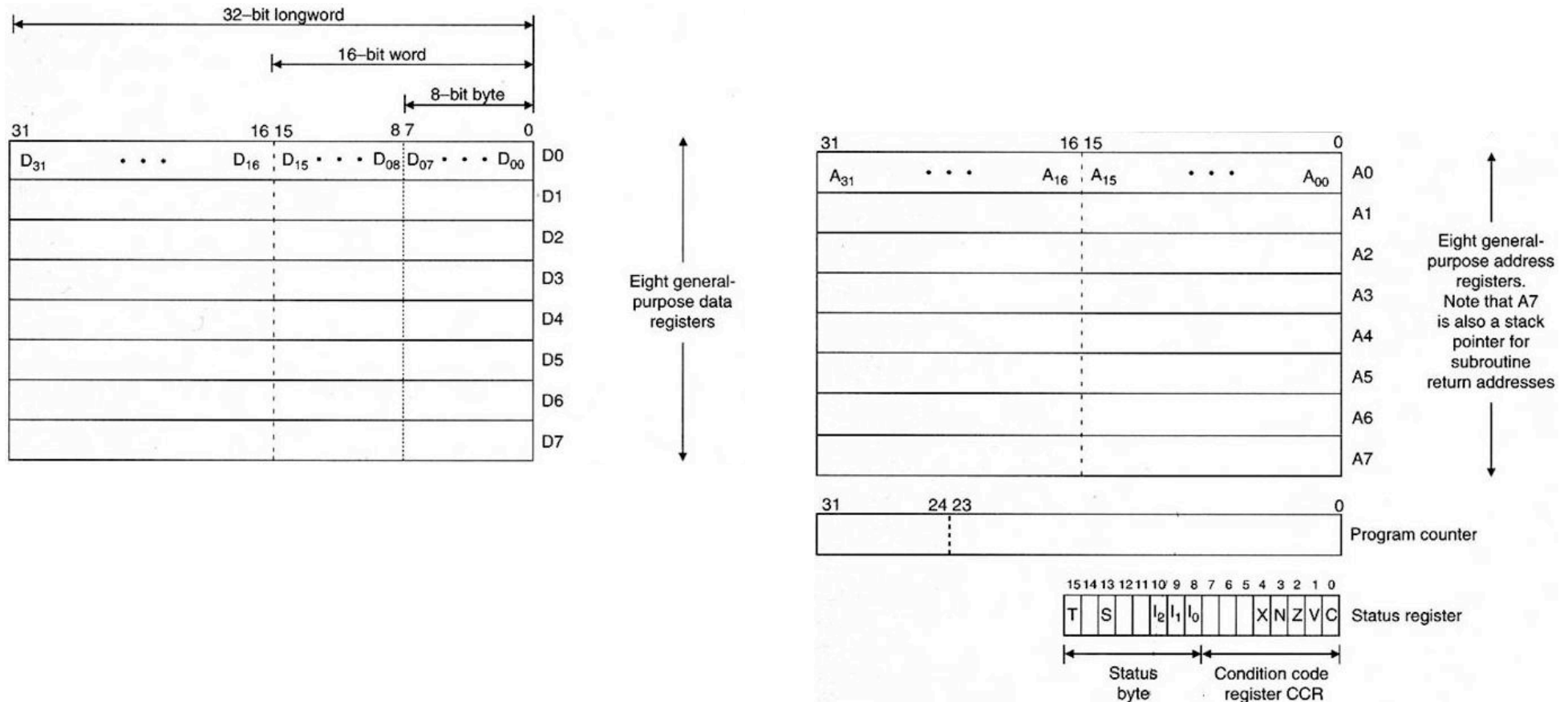
- Specifications
  - 32-bit data and address registers
  - 16-bit data bus
  - 24-bit address bus
  - 14 addressing modes
  - Memory-mapped input-output
  - Program counter
  - 56 instructions
  - 5 main data types
  - 7 interrupt levels
  - Clock speeds: 4 MHz to 12.5 MHz
  - Synchronous and asynchronous data transfers

D <sub>4</sub>	1		64	D <sub>5</sub>
D <sub>3</sub>	2		63	D <sub>6</sub>
D <sub>2</sub>	3		62	D <sub>7</sub>
D <sub>1</sub>	4		61	D <sub>8</sub>
D <sub>0</sub>	5		60	D <sub>9</sub>
$\overline{AS}$	6		59	D <sub>10</sub>
$\overline{UDS}$	7		58	D <sub>11</sub>
$\overline{LDS}$	8		57	D <sub>12</sub>
$\overline{R/\overline{W}}$	9		56	D <sub>13</sub>
$\overline{DTACK}$	10		55	D <sub>14</sub>
$\overline{BG}$	11		54	D <sub>15</sub>
$\overline{BGACK}$	12		53	GNI
$\overline{BR}$	13		52	A <sub>23</sub>
V <sub>CC</sub>	14	68000	51	A <sub>22</sub>
CLK	15	CPU	50	A <sub>21</sub>
GND	16		49	V <sub>CC</sub>
$\overline{HALT}$	17		48	A <sub>20</sub>
$\overline{RESET}$	18		47	A <sub>19</sub>
$\overline{VMA}$	19		46	A <sub>18</sub>
E	20		45	A <sub>17</sub>
$\overline{VPA}$	21		44	A <sub>16</sub>
$\overline{BERR}$	22		43	A <sub>15</sub>
$\overline{IPL}_2$	23		42	A <sub>14</sub>
$\overline{IPL}_1$	24		41	A <sub>13</sub>
$\overline{IPL}_0$	25		40	A <sub>12</sub>
FC <sub>2</sub>	26		39	A <sub>11</sub>
FC <sub>1</sub>	27		38	A <sub>10</sub>
FC <sub>0</sub>	28		37	A <sub>9</sub>
A <sub>1</sub>	29		36	A <sub>8</sub>
A <sub>2</sub>	30		35	A <sub>7</sub>
A <sub>3</sub>	31		34	A <sub>6</sub>
A <sub>4</sub>	32		33	A <sub>5</sub>

# What is “Microprocessor Architecture”

- For our purposes the architecture is the software or programmer’s model of the microprocessor
  - The CPU registers available to the programmer
  - The basic instructions the CPU can perform
  - The ways these instructions can specify a memory location
  - The way data is organized in memory
  - How the CPU accesses & controls peripheral devices

# 68000 Programming Model

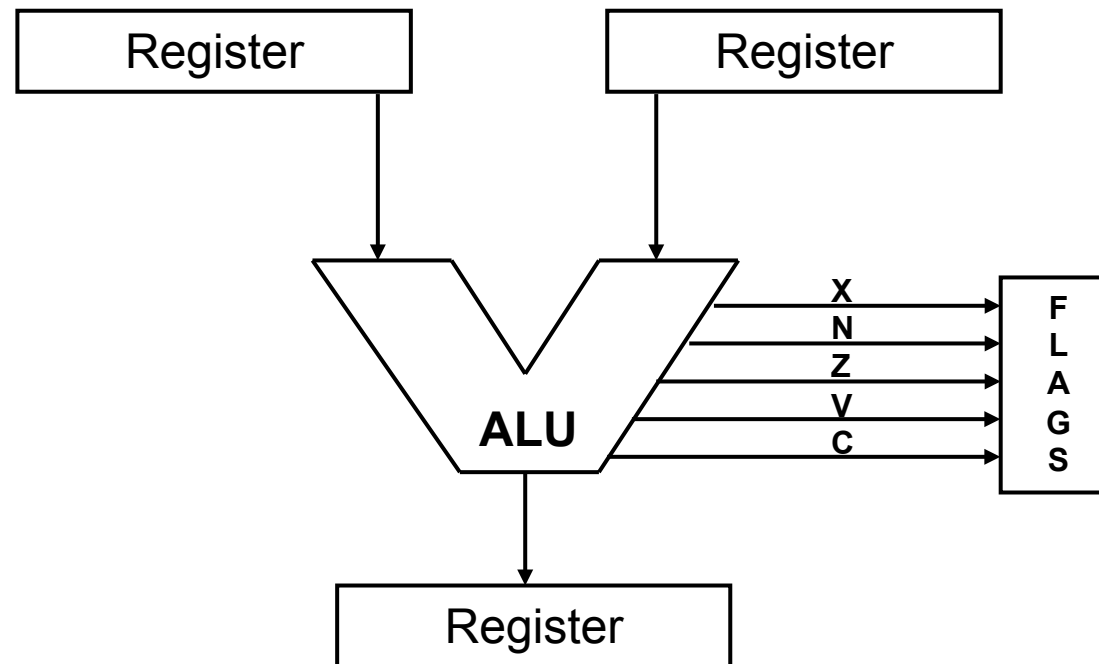




# 68000 Registers

Name	Label	Number	Size	Function
Data registers	D0-D7	8	32 bit	Stores 8-/16-/32-bit data
Address registers	A0-A6	7	32 bit	Stores 16-/32-bit pointers (addresses of data)
Stack pointer	SP	2	32 bit	Store a pointer to a group of data known as the stack. Also known as A7. There's two stack pointers: USP and SSP.
Program counter	PC	1	32 bit	Contains the address of the NEXT instruction to fetch and execute
Status Register	SR	1	16 bit	Contains information on the results of the last instruction. Consists of the system byte and the condition codes register (CCR)

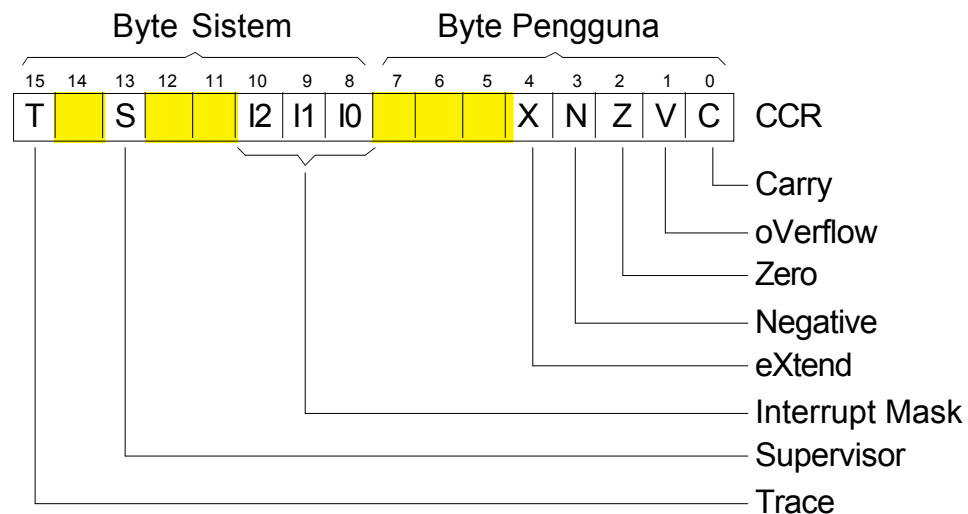
# Status Register General Idea



Status register stores an "analysis" of the last operation involving the ALU

# Control/Status Register

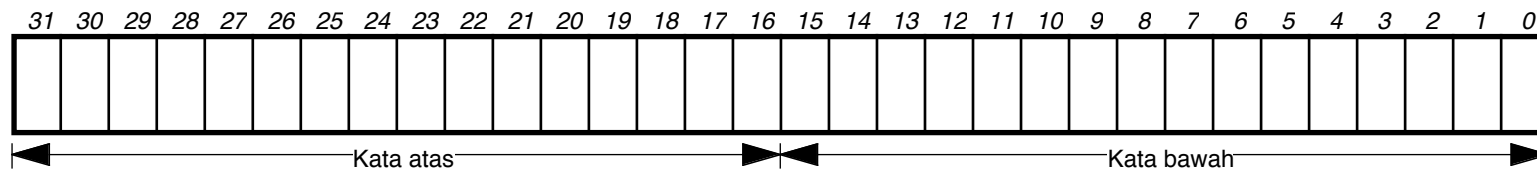
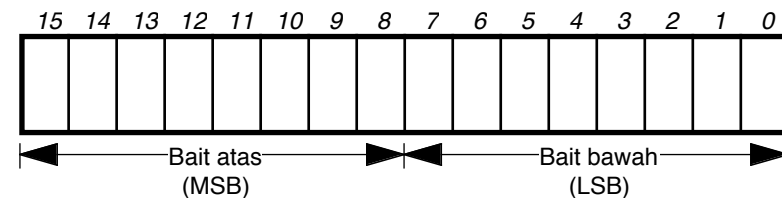
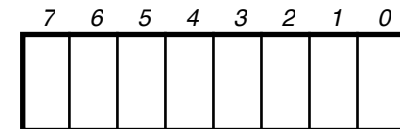
- System Byte
  - Only modifiable is supervisor mode
  - Details in later modules
- User Byte: CCR
  - For user-level programs
  - Behavior depends on instruction



Bit	Meaning
C	Set if a carry or borrow is generated. Cleared otherwise.
V	Set if a signed overflow occurs. Cleared otherwise.
Z	Set if the result is zero. Cleared otherwise.
N	Set if the result is negative. Cleared otherwise.
X	Retains the carry bit for multi-precision arithmetic
I	Systems responds to interrupts with a level higher than I
S	1 means CPU in supervisor mode, 0 means user mode
T	0 for normal operation, 1 to stop the CPU after EVERY instruction for runtime debugging

# 68000 Data Sizes

- Bit (“binary digit”)
  - Smallest amount of data
  - 1 bit stores either binary 0 or binary 1.
- BCD (“binary coded decimal”)
  - 4 bits that represents decimal 0 to 9
  - Used by only 3 instructions
- *Byte*
  - 8 bits that is processed as one unit
- *Word*
  - 16 bits that is processed as one uni
- *Longword*
  - 32 bits that is processed as one unit



# Byte Addressing

- Bytes can be stored in any even or odd location

000000	10101010	11000101	000001
000002	00011001	11110010	000003
000004			000005
FFFFFC	11001000	11111111	FFFFFD
FFFFFE	00011010	01010111	FFFFFF

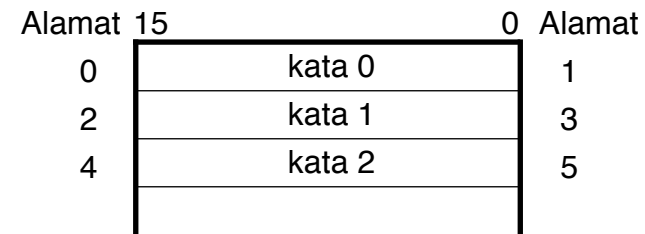
Alamat genap

Alamat ganjil

Alamat			Alamat
0	bait 0	bait 1	1
2	bait 2	bait 3	3
4	bait 4	bait 5	5
6	bait 6	bait 7	7

# Word Addressing

- Word must stored at even addresses
- Attempt to store word at odd address result in a “trap”
- Trap : recoverable crash



Contoh:  $ABCD_{16}$  disimpan di alamat 0.

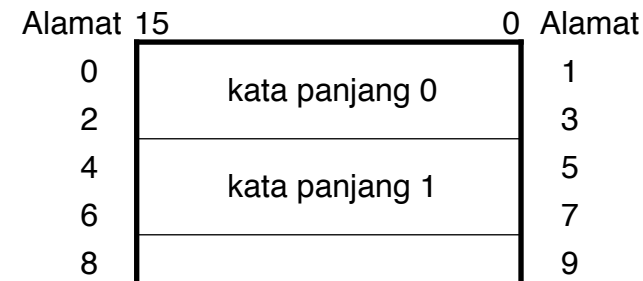
$000000_{16}$ 

1010 1011	1100 1101
-----------	-----------

 $0000001_{16}$

# Longword Addressing

- Longword can be stored at even address
- Longword requires two memory accesses (two rows in memory map)



Contoh:  $ABCD1234_{16}$  disimpan di alamat 0.

$000000_{16}$	1010 1011	1100 1101	$0000001_{16}$
$000002_{16}$	0001 0010	0011 0100	$0000003_{16}$

# Register Transfer Language (RTL)

- A simple notation to describe the operations carried out by CPU clearly and unambiguously
  - We will use it to describe the function of instruction

100	means “#100” or “the number 100”
[M(4)]	means “contents stored in memory location 4”
[M(4)] = 100	means “memory location 4 contains #100”
[M(4)] ← 25	means “load number 25 into memory location 4”
[PC] ← 4	means “load number 4 into PC”
[M(4) ← 100+[M(4)]	means “add #100 to contents of location 4 and save”

(This slide falls in the “good to know” category.)



# Instruction Set

- The complete list of instructions is known as the instruction set
- Instructions are categorized according to basic operation performed:
  - Data transfer
  - Arithmetic
  - Logic
  - Shifts & rotates
  - Bit manipulation
  - BCD
  - Program Control
  - System Control

# Basic Instruction Set

Mnemonic	Meaning	Mnemonic	Meaning
ABCD	Add decimal with extend	MOVE	Move source to destination
ADD	Add binary	MULS	Sign multiply
AND	Logical AND	MULU	Unsigned multiply
ASL	Arithmetic shift left	NBCD	Negate decimal with extend
ASR	Arithmetic shift right	NEG	Negate
Bcc	Branch conditionally	NOP	No operation
BCHG	Bit test and change	NOT	One's complement
BCLR	Bit test and clear	OR	Logical OR
BRA	Branch always	PEA	Push effective address
BSET	Bit test and set	RESET	Reset external devices
BSR	Branch to subroutine	ROL	Rotate left
BTST	Bit test	ROR	Rotate right
CHK	Check register with bounds	ROXL	Rotate left through extend
CLR	Clear operand	ROXR	Rotate right through extend
CMP	Compare	RTE	Return from exception
DBcc	Decrement and branch conditionally	RTR	Return and restore
DIVS	Exclusive OR	RTS	Return from subroutine
DIVU	Unsigned divide	SBCD	Subtract decimal with extend
EOR	Jump to subroutine	Scc	Set conditionally
EXG	Exchange registers	STOP	Stop processor
EXT	Sign extend	SUB	Subtract binary
JMP	Jump to effective address	SWAP	Swap data register halves
JSR	Logical shift left	TAS	Test and set operand
LEA	Load effective address	TRAP	Trap
LINK	Link stack	TRAPV	Trap on overflow
LSL	Signed divide	TST	Test
LSR	Logical shift right	UNLK	Unlink stack

# Instruction Format

- Generic instruction format

```
<label> opcode<.size> <operands> <;comments>
```

- **<label>** pointer to the instruction's memory location
- **opcode** operation code (MOVE, ADD, etc)
- **<.size>** size/width of operand (B,W,L)
- **<operands>** data used in the operation
- **<;comments>** for program documentation

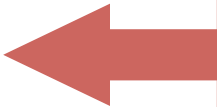
Instruction			RTL
	MOVE.W	#100,D0	[D0] ← 100
	MOVE.W	\$100,D0	[D0] ← [M(\$100)]
	ADD.W	D1,D0	[D0] ← [D0] + [D1]
	MOVE.W	D1,100	[M(100)] ← D0
DATA	DC.B	20	[DATA] ← 20
	BRA	LABEL	[PC] ← label

# Operands

- Operands can be
  - Registers
  - Constants
  - Memory addresses
- Operands specify addressing modes such as
  - Dn: data register direct `MOVE.W D0, D1`
  - An: address register indirect `MOVE.W (A0), D1`
  - #n: immediate `MOVE.W #10, D1`
  - N: absolute `MOVE.W $1000, D1`
- Operands can be specified in several formats
  - Decimal: default
  - Hexadecimal: prefixed by \$
  - Octal: prefixed by @
  - Binary: prefixed by %
  - ASCII: within single quotes 'ABC'

# Addressing Modes

- **Addressing mode** : the mechanism used to compute the operand address
- 68000 has sophisticated addressing modes
  - Simplifies assembler programming because it reduces the number steps required to specify an address
- 68000 has 14 addressing modes but really falls into 6 major categories.
  - Register direct
  - Immediate
  - Absolute
  - Program counter relative
  - Register indirect
  - Inherent
- **Effective address** : the actual address used by the instruction
  - Examples:
    - Data register D1 in the processor
    - Address \$10000 in memory



We'll cover the first three in this module. The rest will be covered later.

# A simple instruction

Format:

**CLR.s**      <ea>

Example:

**CLR.W**      **D1**      ;Clears lower word of D1

Effect:



# Another simple instruction

Format:

**MOVE.s <ea>, <ea>**

Example :

**MOVE.W D0, D1 ; Copy lower word of D0 to D1**

Effect:

D0 

12	34	56	78
----	----	----	----

D1 

78	56	34	12
----	----	----	----

D0 

12	34	56	78
----	----	----	----

D1 

78	56	56	78
----	----	----	----

berubah

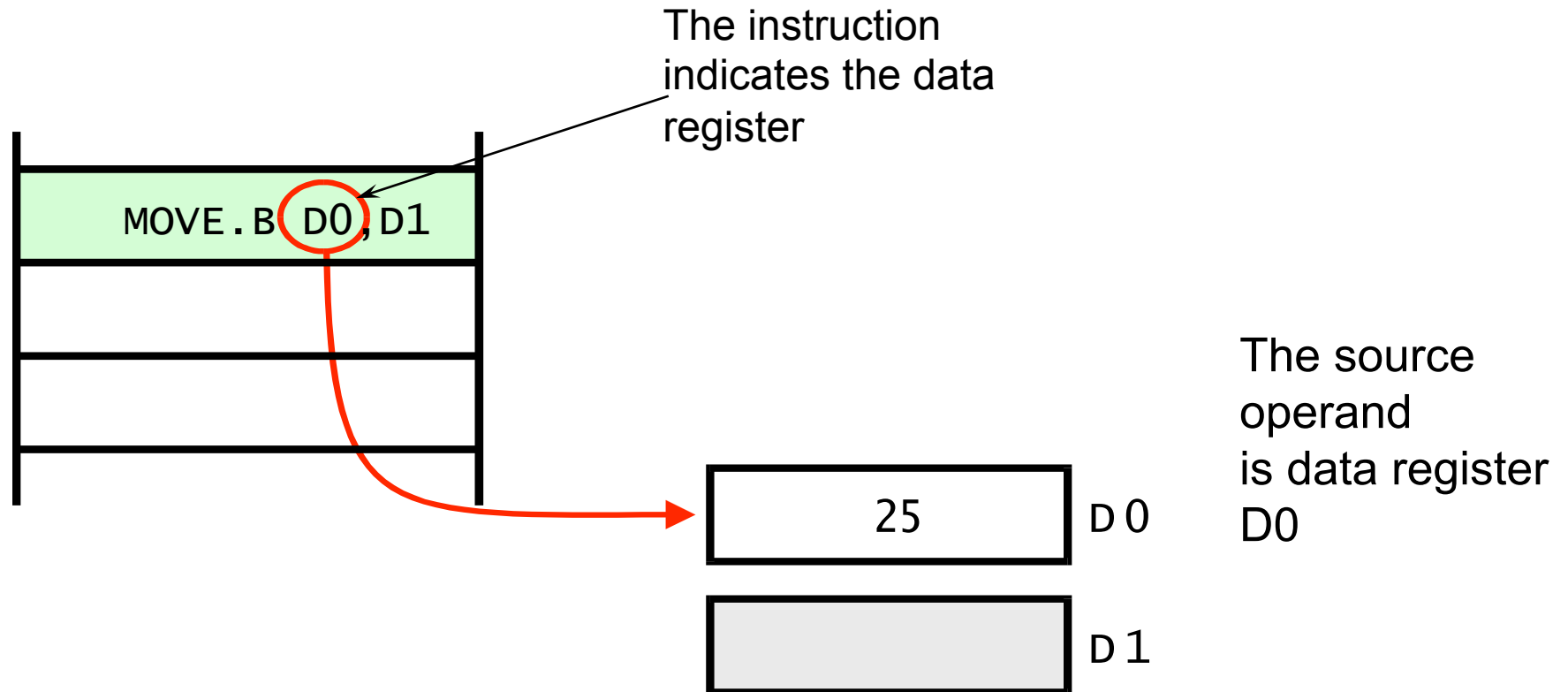
# Register Direct Addressing

- Effective address : one of the 8 data register (D0-D7)
- The simplest addressing mode
- Source or destination of an operand is a data register or an address register.
  - The contents of the specified source register provide the source operand.
  - Similarly, if a register is a destination operand, it is loaded with the value specified by the instruction.
- Examples:

```
MOVE.B D0,D3  Copy the source operand in register D0 to register D3
SUB.L  A0,D3  Subtract the source operand in register A0 from register D3
CMP.W  D2,D0  Compare the source operand in register D2 with register D0
ADD    D3,D4  Add the source operand in register D3 to register D4
```

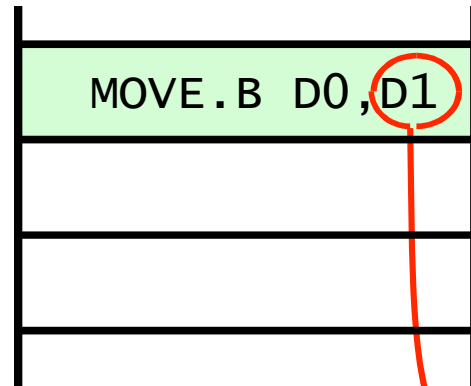


# Register Direct Addressing

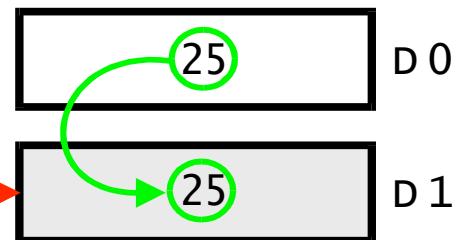


The MOVE.B D0,D1 instruction uses data registers for both source and destination operands

# Register Direct Addressing



The effect of this instruction is to copy the contents of data register D0 in to data register D1



- Short instructions (need only 3 bits to specify one of 8 data registers)
- Fast because the external memory does not have to be accessed.

# Immediate Addressing Mode

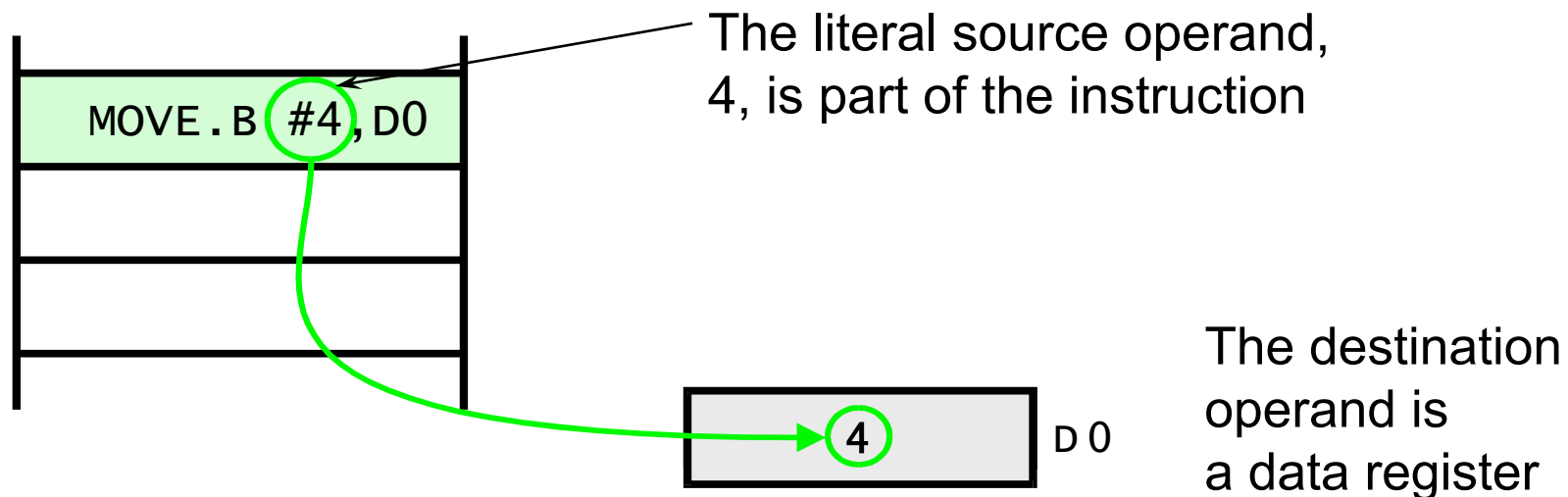
- Effective address : the memory address immediately following the instruction word
- Indicated by a # symbol in front of the source operand.
- Can be used only to specify a source operand.
- The actual operand forms part of the instruction.
- An immediate operand is also called a literal operand.
- Can only be used as a **source** addressing mode. The destination of the data must also be specified by the destination addressing mode.

## TIP:

**Very useful to load constants (values that never change).**

# Immediate Addressing Mode

- The instruction **MOVE.B #4,D0** uses a literal source operand and a register direct destination operand



- The effect of this instruction is to copy the literal value 4 to data register D0

# Absolute Addressing Mode

- Effective address : the memory location specified by the instruction
- In *direct* or *absolute addressing*, the instruction provides the address of the operand in memory.
- Direct addressing requires two memory accesses. The first is to access the instruction and the second is to access the actual operand.

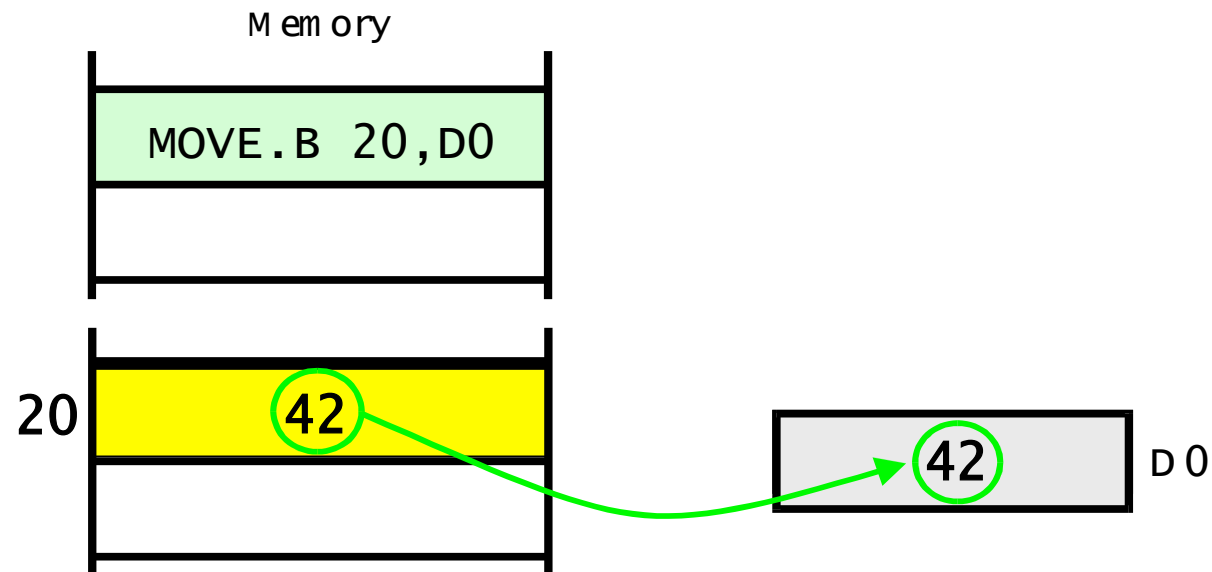
## TIP:

Most addresses are specified in hex so the '\$' hex symbol is commonly found when memory accesses are involved

- Example:

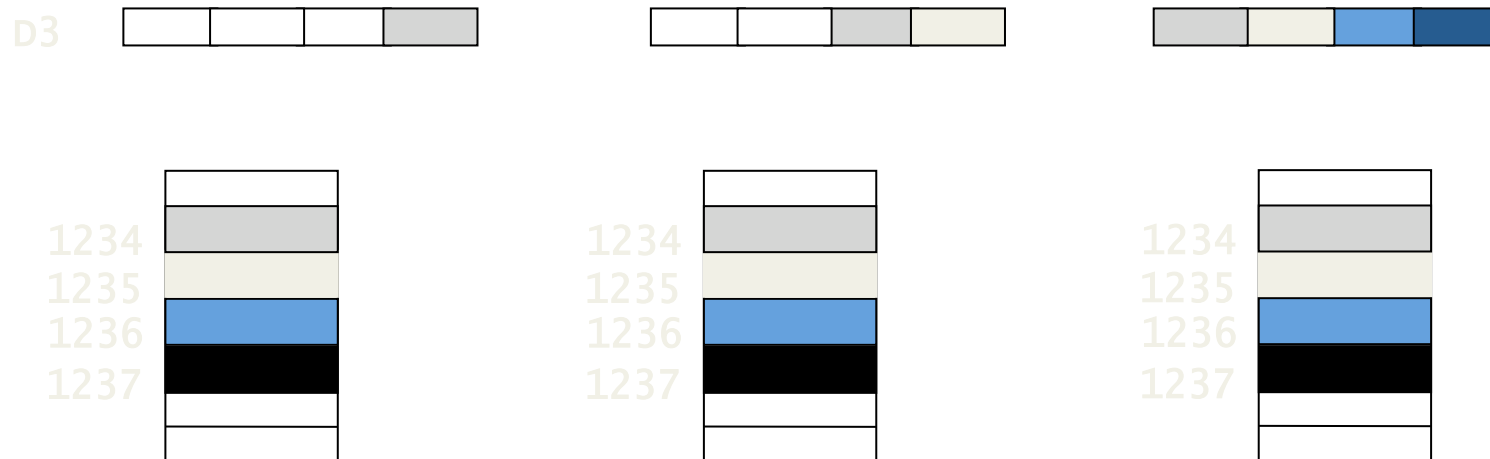
**CLR.W**    **\$2000**    clears word located at address 2000 hex.

# Absolute Addressing Mode



The effect of `MOVE.B 20, D0` is to read the contents of memory location 20 and copy them to D0

# Bytes, Words, and Longwords



**MOVE.B \$1234,D3**

$[D3(0:7)] \leftarrow [M(\$1234)]$

**MOVE.W \$1234,D3**

$[D3(0:15)] \leftarrow [M(\$1234)]$

**MOVE.L \$1234,D3**

$[D3] \leftarrow [M(\$1234)]$

# Address Register Indirect Addressing

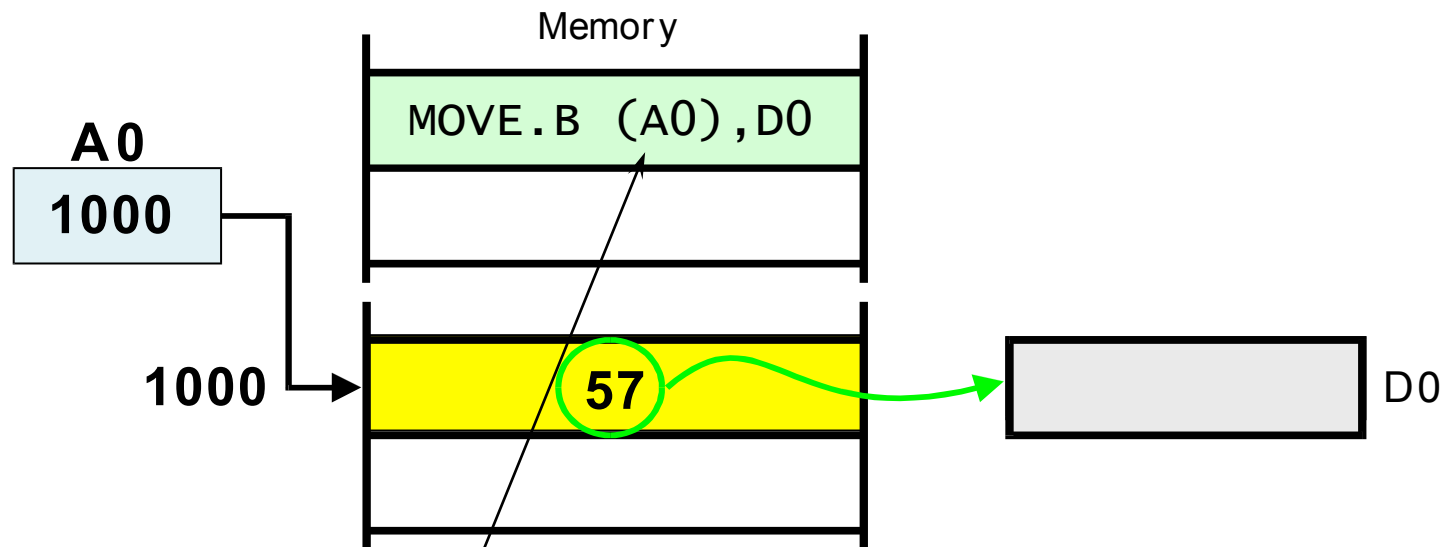
- Effective address : the memory address specified by the address register contained in the instruction
- The instruction specifies one of the 68000's address registers; for example, MOVE.B (A0),D0.
- The specified address register contains the address of the operand.
- The processor then accesses the operand pointed at by the address register.
- Example:

CLR.W (A0) clears located in memory, specified by A0.



# Register Indirect Addressing (ARI) Mode

- This instruction means load D0 with the contents of the location pointed at by address register A0



The instruction specifies the source operand as (A0).

# Relative Addressing Mode

- Effective address : calculated by adding a **displacement** to the PC
- Format : XXXX or XXXX(PC)
- Contoh:

**BRA \*+2**

The “branch always” instruction transfers control to an instruction 2 bytes ahead. The “\*” character means “current position”.

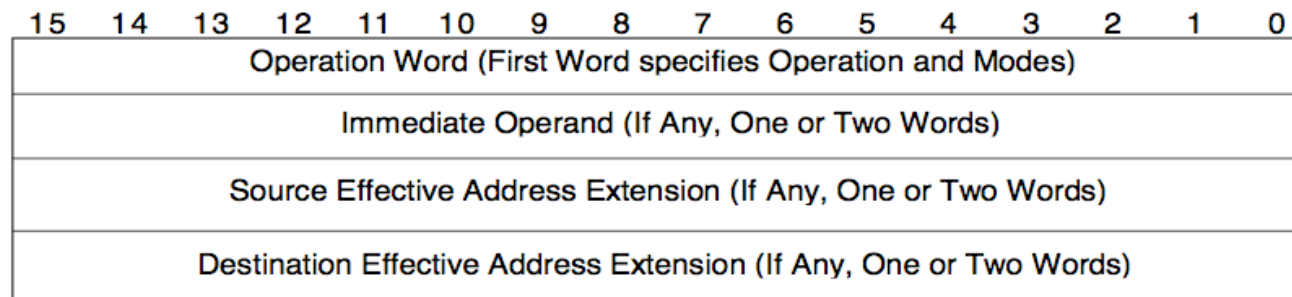
# Inherent Addressing Mode

- Effective address : in processor register (CCR, SR, SP, USP, SSP or PC) or memory but not indicated in the instruction
- Format : usually no operand
- Example:
  - RTS

Takes 4 bytes from the “top of stack” and loads it into the PC. Involves PC, SP and memory.

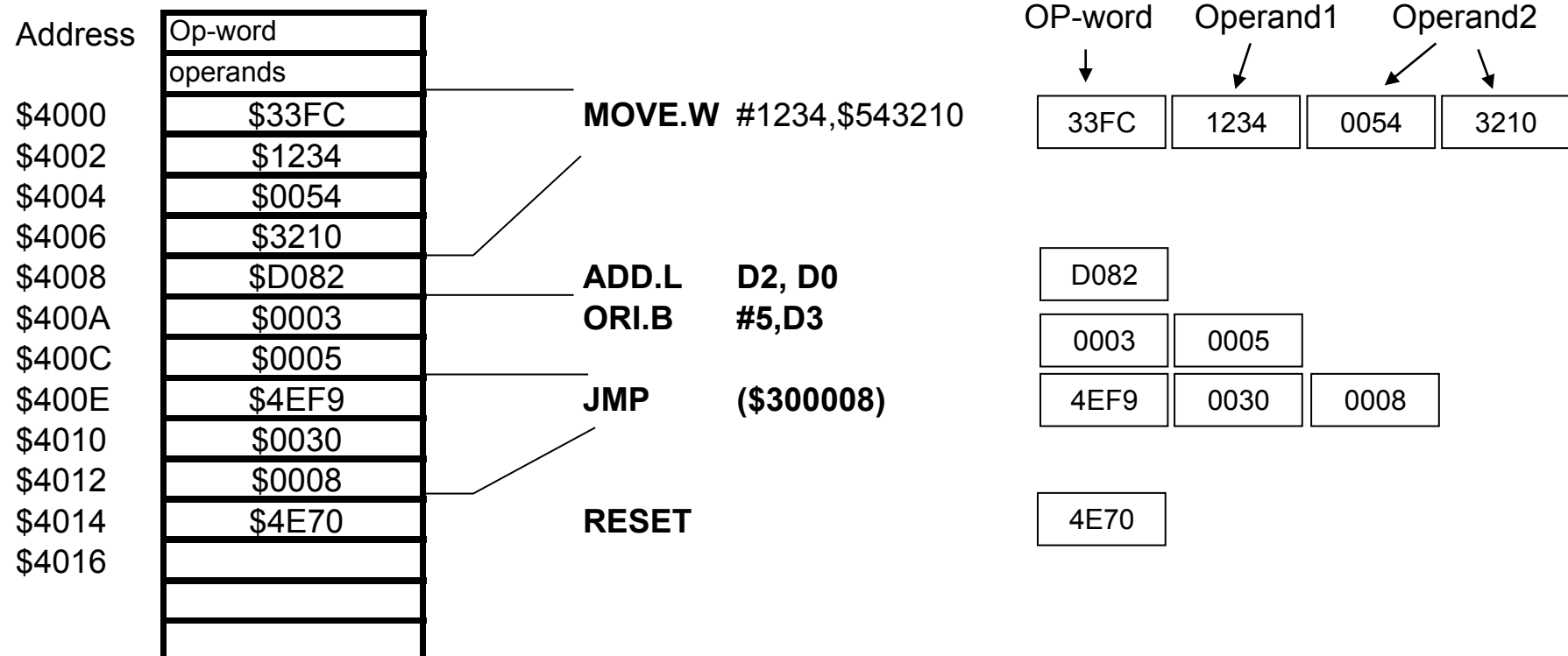
# Machine instruction

- Each instruction is at least 1 word, at most 5 words.
- The first word is known as the *operation* word, which determines:
  - Operation required
  - Data size: byte, word or longword
  - Length of the complete instruction
  - Where to find data (effective address)
- The method of instruction encoding (how a instruction is written in binary) is complex!



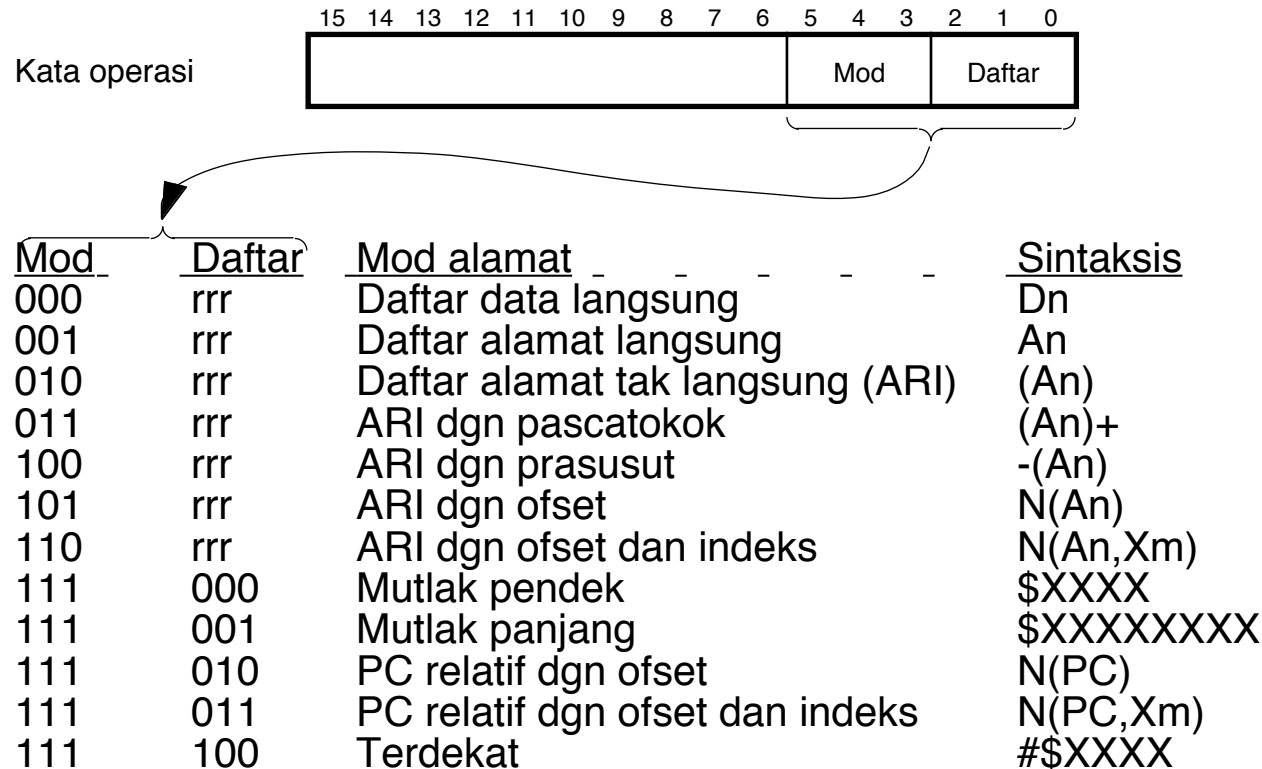
# Variable Length Instructions

- Not only is the 68000 instruction format complex, the number of bytes in an instruction also varies.



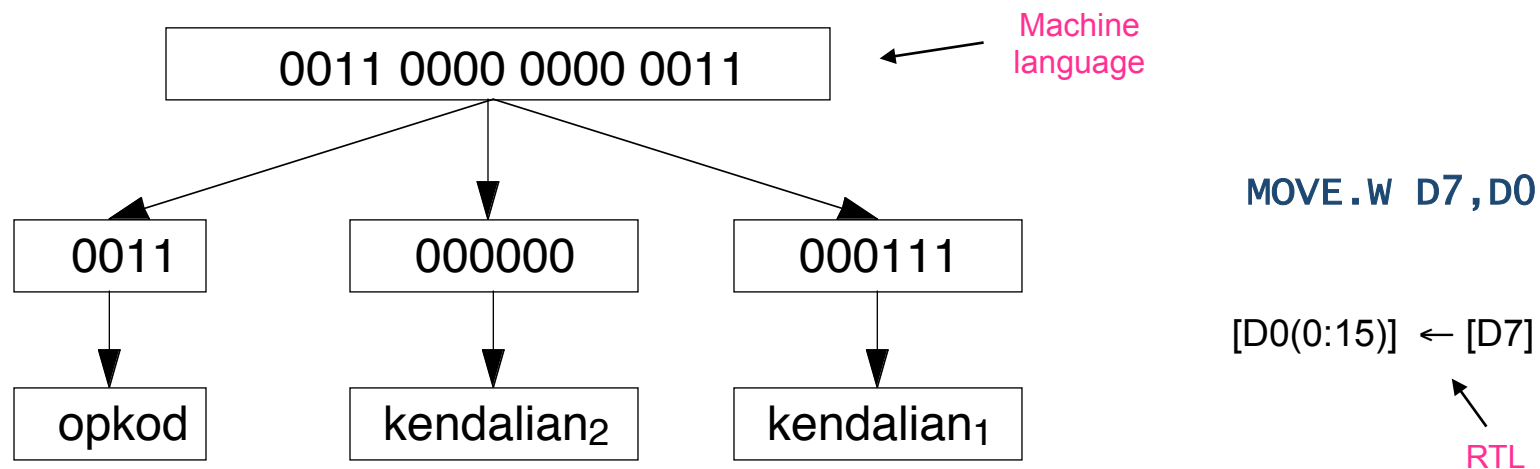
# Addressing Mode Encoding

- Addressing mode is encoded using 6 bits within an instruction.
- For a single-effective address instruction, the addressing mode is located in bits 0-5.



# Machine Format for MOVE Instruction

- The MOVE instruction is most heavily used.
  - Generally, the most commonly used 68000 instructions are encoded in fewer bits.
  - MOVE is encoded by only two bits (bit 15:14 = 00)
- The effective address encoding is slightly different for the destination operand
- The 68000 is a prime example of the CISC (Complex Instruction Set Computer)
- Example: Machine code for MOVE.W D7,D0



# Opcodes

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

0	0	0	1	1	0	1	0	0	0	1	1	1	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Bit 15, 14, 13, 12	Operation
0 0 0 0	
0 0 0 1	MOVE Byte
0 0 1 0	MOVE Long
0 0 1 1	MOVE Word
0 1 0 0	Miscellaneous
0 1 0 1	ADDQ / SUBQ / ScC / DBcc
0 1 1 0	Bcc
0 1 1 1	MOVEQ
1 0 0 0	OR / DIV / SBCD
1 0 0 1	SUB / SUBX
1 0 1 0	(Unassigned)
1 0 1 1	CMP / EOR
1 1 0 0	AND / MUL / ABCD / EXG
1 1 0 1	ADD / ADDX
1 1 1 0	Shift / Rotate
1 1 1 1	(Unassigned)



# Summary

- **Register direct addressing** is used for variables that can be held in registers
- **Literal (immediate) addressing** is used for constants that do not change
- **Direct (absolute) addressing** is used for variables that reside in memory
- The only difference between register direct addressing and direct addressing is that the former uses registers to store operands and the latter uses memory
- For Further Info:
  - Motorola 68000 From Wikipedia, the free encyclopedia:
    - <http://en.wikipedia.org/wiki/68000>
  - CPU World - Motorola 68000 microprocessor family:
    - <http://www.cpu-world.com/CPUs/68000>
  - 68000 Programmer's Reference Manual
    - [http://www.freescale.com/files/archives/doc/ref\\_manual/M68000PRM.pdf](http://www.freescale.com/files/archives/doc/ref_manual/M68000PRM.pdf)