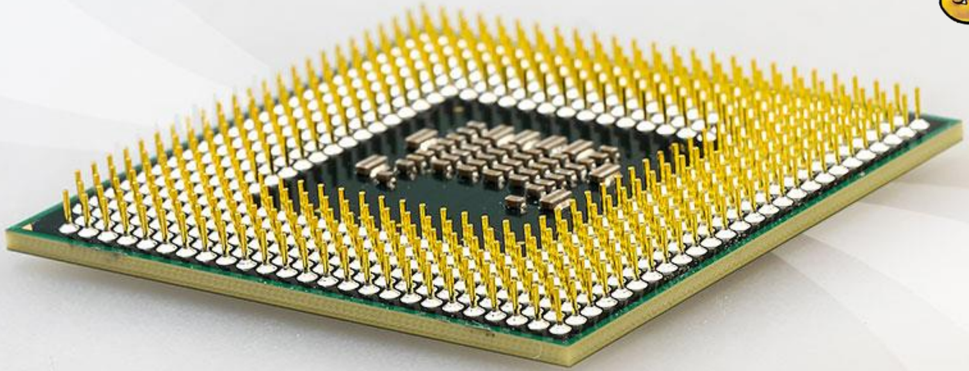
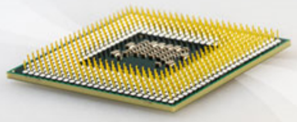


An Overview on Motorola 68000

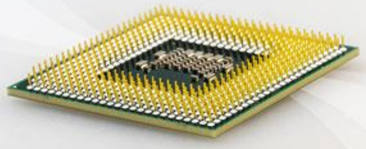
Ali Abbasi
Dr. Keikha

ECE of University of Sistan & Baluchestan

Outline



Genealogy



Previous Generation 6800

- 8-bit microprocessor
- Up to 2 MHz
- 64 KB RAM
- No I/O ports

68000

- 32-bit CPU
- 16-bit data bus
- Up to 20 MHz
- 16 MB RAM
- No I/O ports

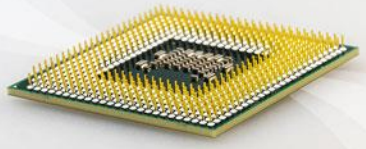
Related Family 68008

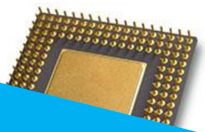
- 8-bit data bus
- Up to 16.67 MHz
- 4 MB RAM

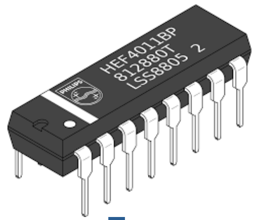
Next Generation 68010

- 32-bit CPU
- 16-bit data bus
- Up to 16 MHz
- 16 MB RAM
- Virtual memory supprt
- No I/O ports

Specifications







68000

- 32-bit CPU
- 16-bit data bus
- Up to 20 MHz
- 16 MB RAM
- No I/O ports

PGA

64-pin DIP

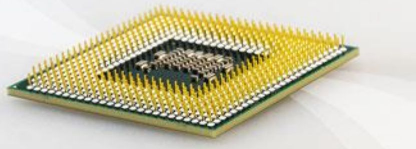
68-pin LCC

Specifications

Motorola 68K (MC68000)

- Released in 1979
- The first member of 680x0 line of microprocessors
- HMOS technology
- 14 addressing mode
- 7 interrupt levels
- Synchronous and asynchronous data transfers
- Competed against the Intel 8086 and Intel 80286A CISC microprocessor
- Much more flexible than other CPU families (z80, 80x86, z80000, etc.)
- An excellent computer for running C code

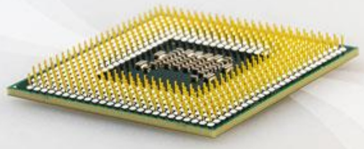
Specifications



Cheaper 68K (68EC000)

- Designed for embedded controller applications
- 8-bit/16-bit Data bus
- 8 MHz/16 MHz Configurations
- FPU was not available (Co-Processor: MC68881/2)

Specifications



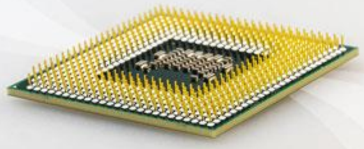
Computers:

- **Apple Lisa 2** (January, 1983)
- **Apple Macintosh 128K** (January, 1983)
- **Atari 520STfm** (January, 1985)
- **Commodore Amiga 500** (July, 1985)
- **Commodore Amiga 1000** (April, 1987)
- **Thousands of printers, automotive engine controllers, and medical manufacturers**

Have you ever heard…

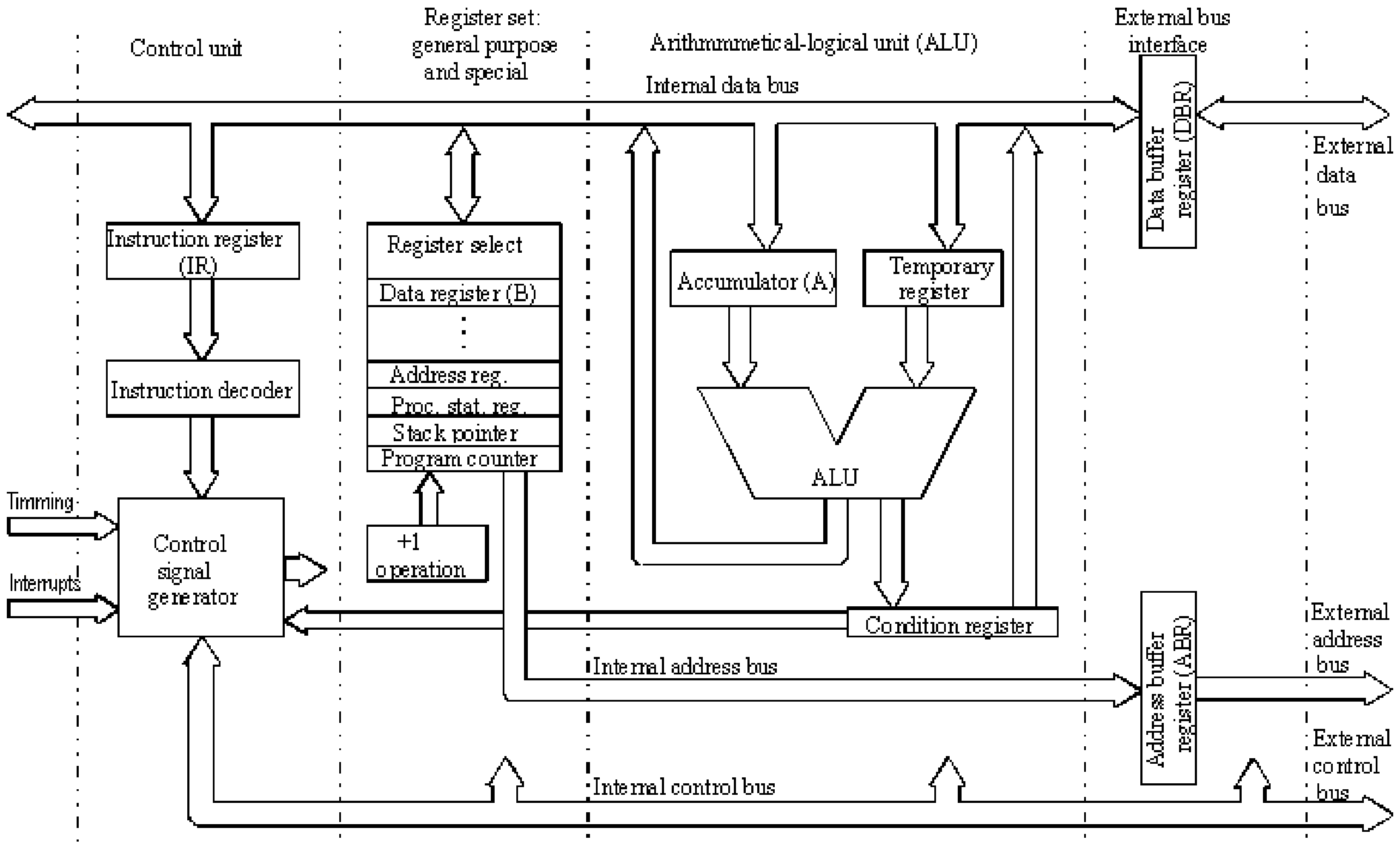
"In the Sega Saturn, the 68EC000 was used as the sound processor. Also a version of Motorola 68000 manufactured by Hitachi, the FD109, was used in various Sega arcade systems."

Manufacturers



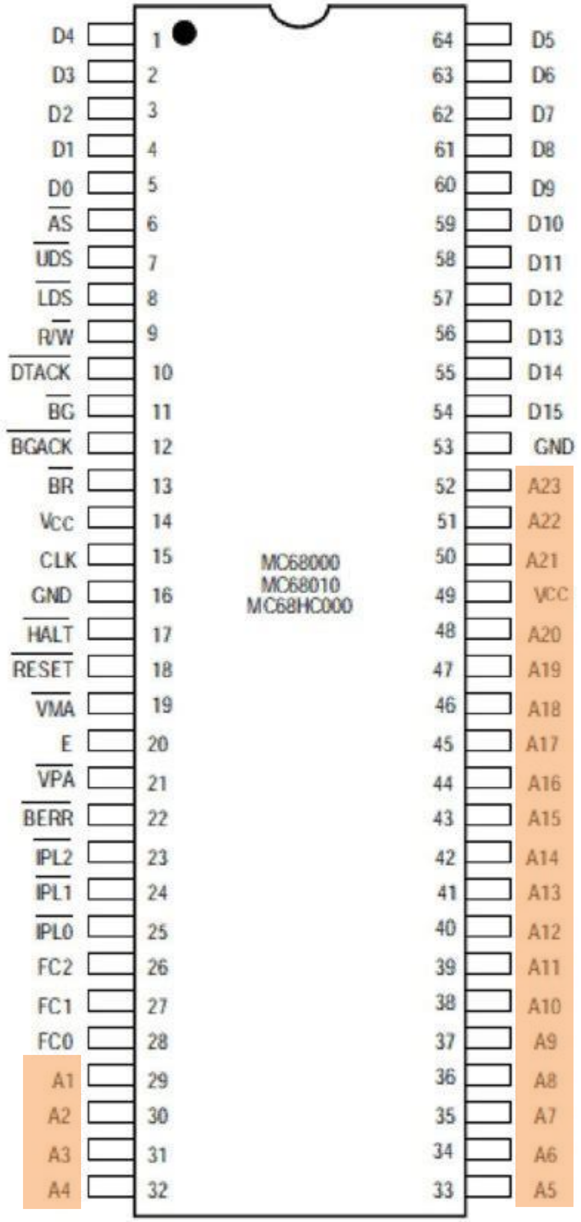
- **Apple** (8 MHz 64-pin side-brazed ceramic DIP)
- **Hitachi** (6 MHz 64-pin side-brazed ceramic DIP)
- **Signetics** (4 MHz 64-pin side-brazed ceramic DIP)
- **Rockwell** (8 MHz 64-pin side-brazed ceramic DIP)
- **Mostek** (8 MHz 64-pin plastic DIP)
- **ST** (10 MHz 64-pin plastic DIP)
- **Thomson** (16 MHz 68-pin plastic LCC)
- **Toshiba** (DIP, Shrink DIP, Ceramic PGA, PLCC, Plastic QFP)

Block Diagram



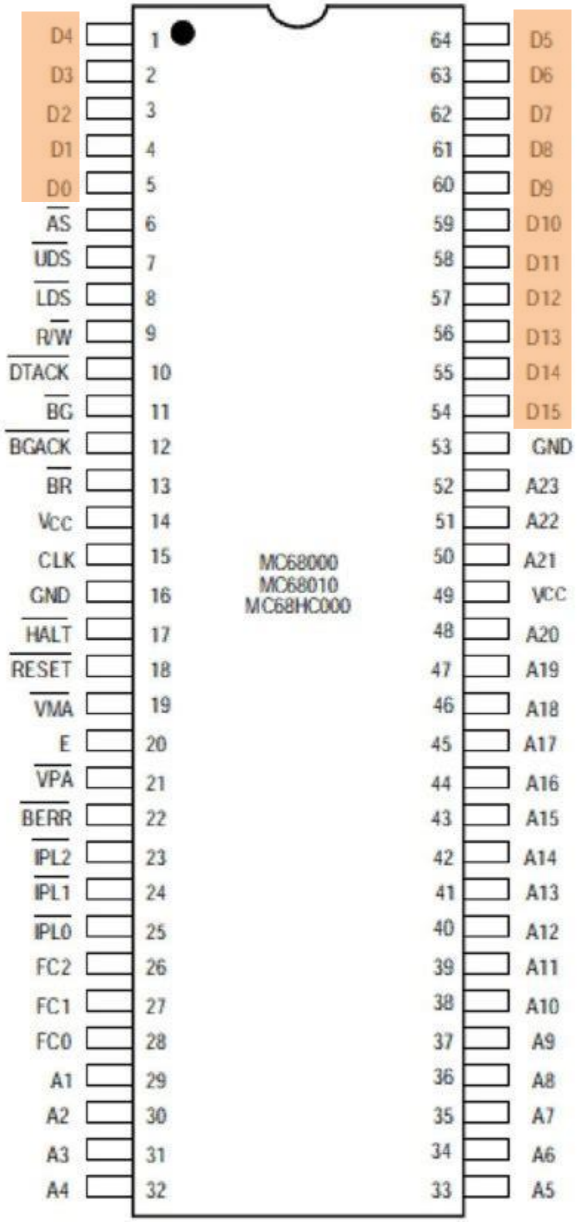
Pins and Signals

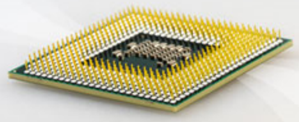
1. **Address Bus (A23-A1)**



Pins and Signals

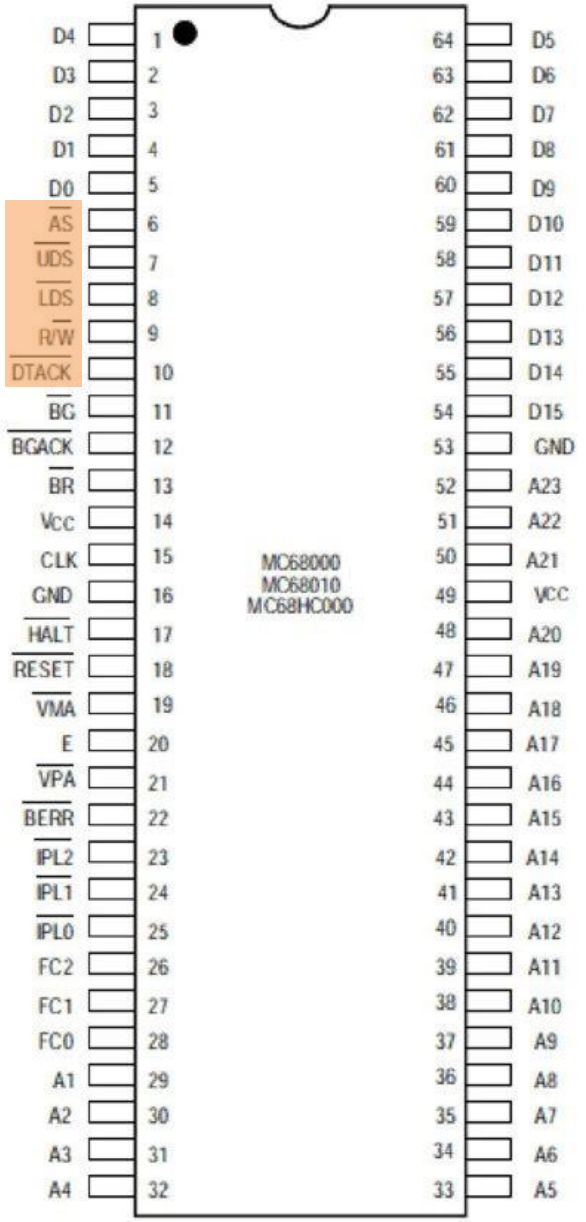
1. **Address Bus (A23-A1)**
2. **Data Bus (D15-D0)**

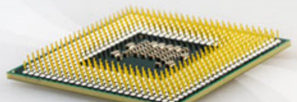




Pins and Signals

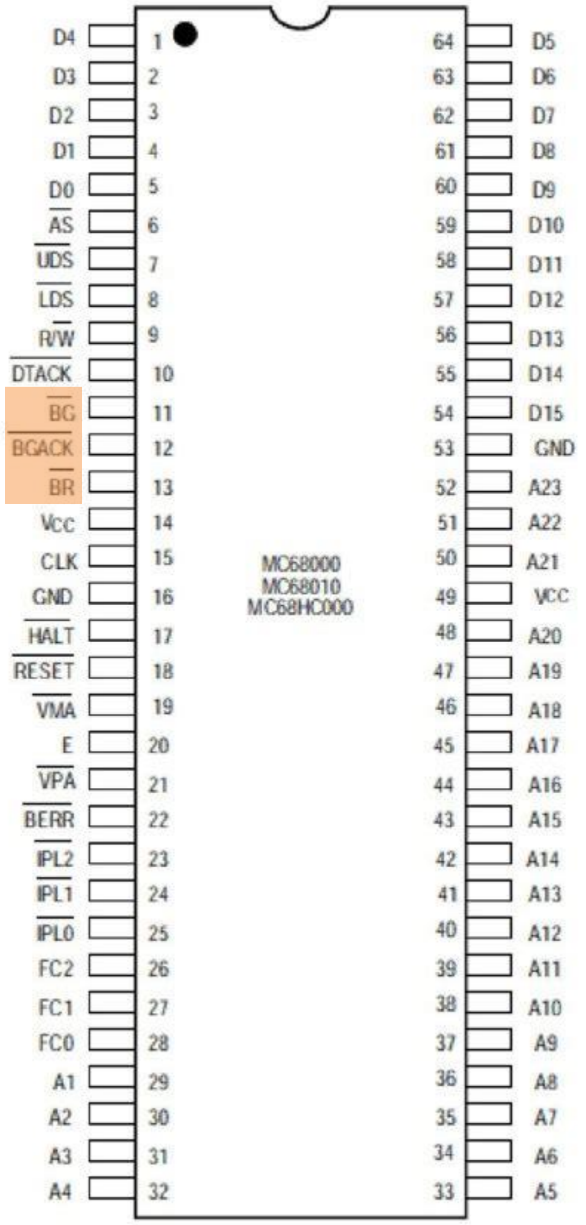
1. **Address Bus (A23-A1)**
2. **Data Bus (D15-D0)**
3. **Asynchronous Bus Control**
   - Address Strobe (~AS)
   - Read/Write (R/~W)
   - Upper and Lower Data Strobes (~UDS, ~LDS)
   - Data Transfer Acknowledge (~DTACK)





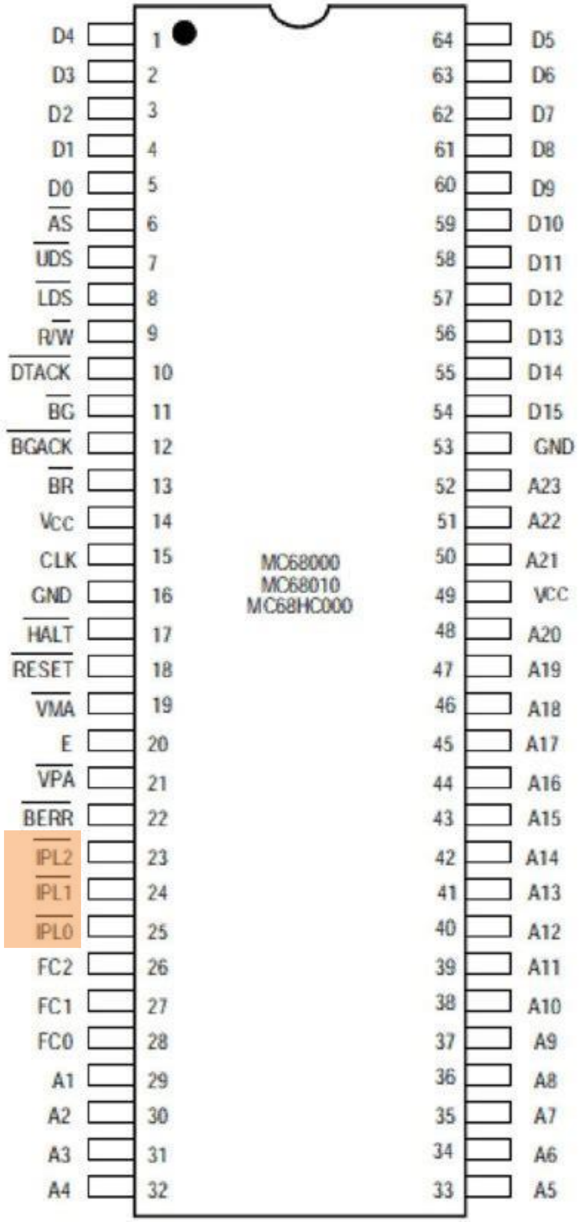
Pins and Signals

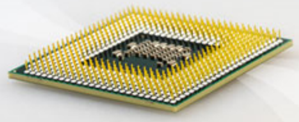
1. **Address Bus (A23-A1)**
2. **Data Bus (D15-D0)**
3. **Asynchronous Bus Control**
   - Address Strobe (~AS)
   - Read/Write (R/~W)
   - Upper and Lower Data Strobes (~UDS, ~LDS)
   - Data Transfer Acknowledge (~DTACK)
4. **Bus Arbitration Control**
   - Bus Request (~BR)
   - Bus Grant (~BG)
   - Bus Grant Acknowledge (~BGACK)



Pins and Signals

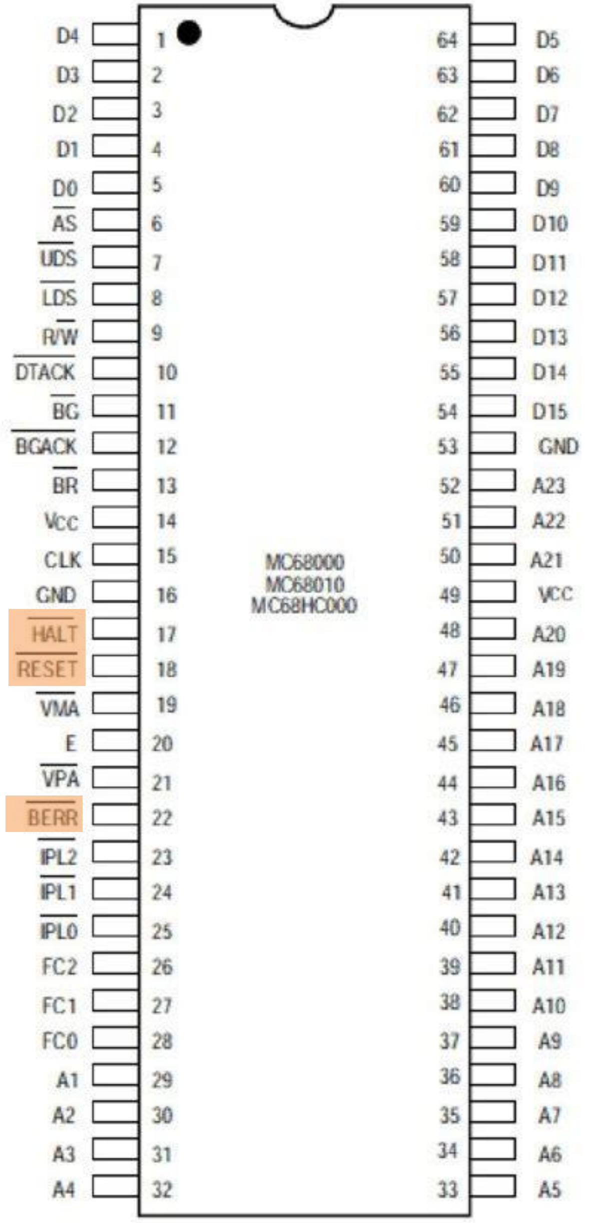
5. **Interrupt Control (IPL0-IPL2)**
    - indicate the encoded priority level of the device requesting an interrupt
    - Level seven has the highest priority

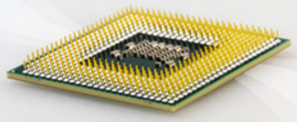




Pins and Signals

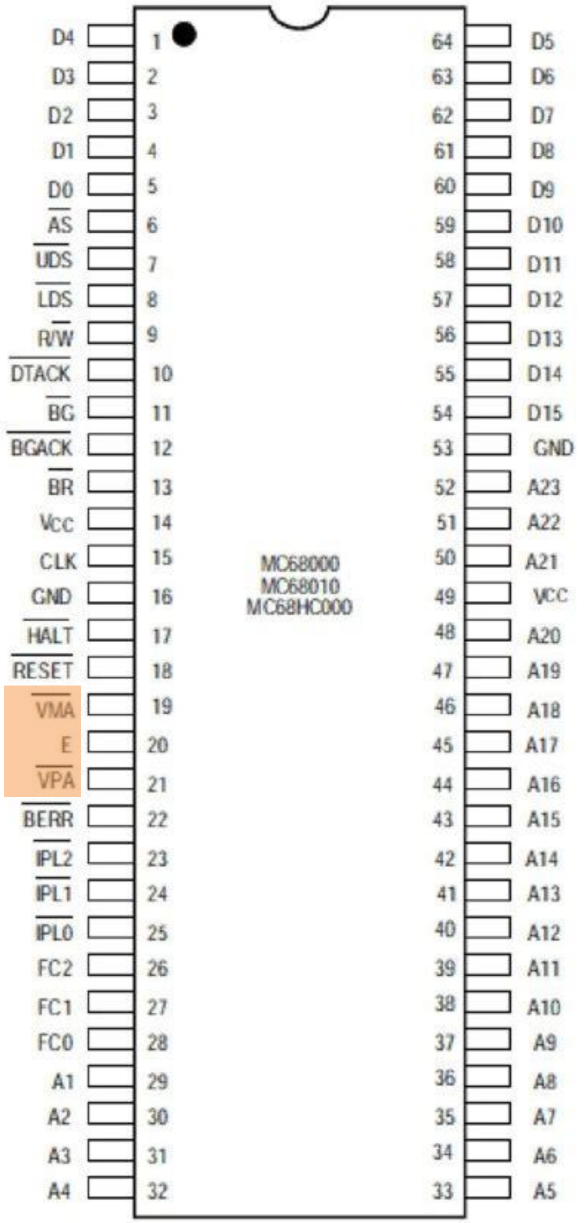
5. **Interrupt Control (IPL0-IPL2)**
   - indicate the encoded priority level of the device requesting an interrupt
   - Level seven has the highest priority
6. **System Control**
   - Bus Error (~BERR)
   - Reset (~RESET)
   - Halt (~HALT)

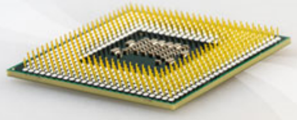




Pins and Signals

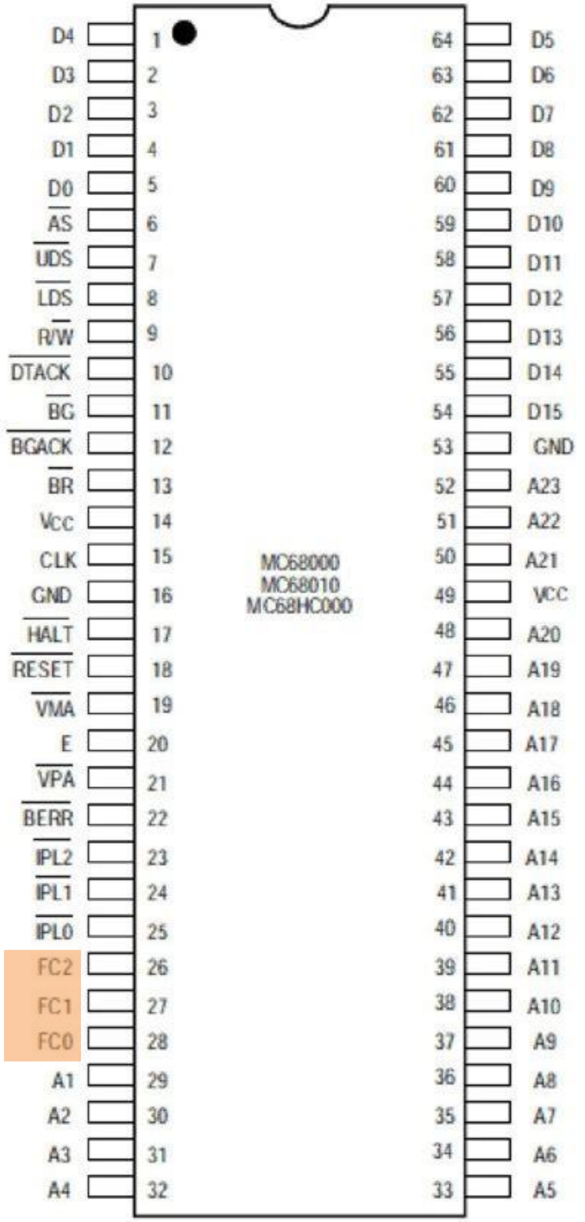
5. **Interrupt Control (IPL0-IPL2)**
   - indicate the encoded priority level of the device requesting an interrupt
   - Level seven has the highest priority
6. **System Control**
   - Bus Error (~BERR)
   - Reset (~RESET)
   - Halt (~HALT)
7. **M6800 Peripheral Control**
   - Enable (E)
   - Valid Peripheral Address (~VPA)
   - Valid Memory Address (~VMA)





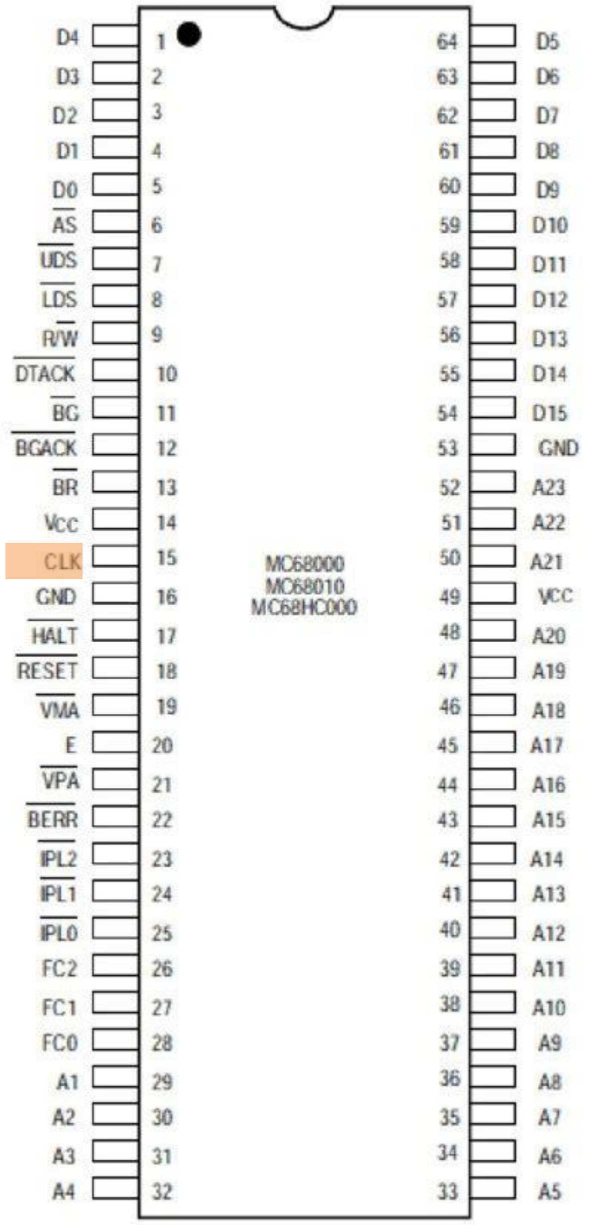
Pins and Signals

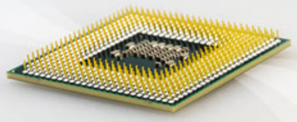
8. **Processor Function Codes (FC0-FC2)**



Pins and Signals

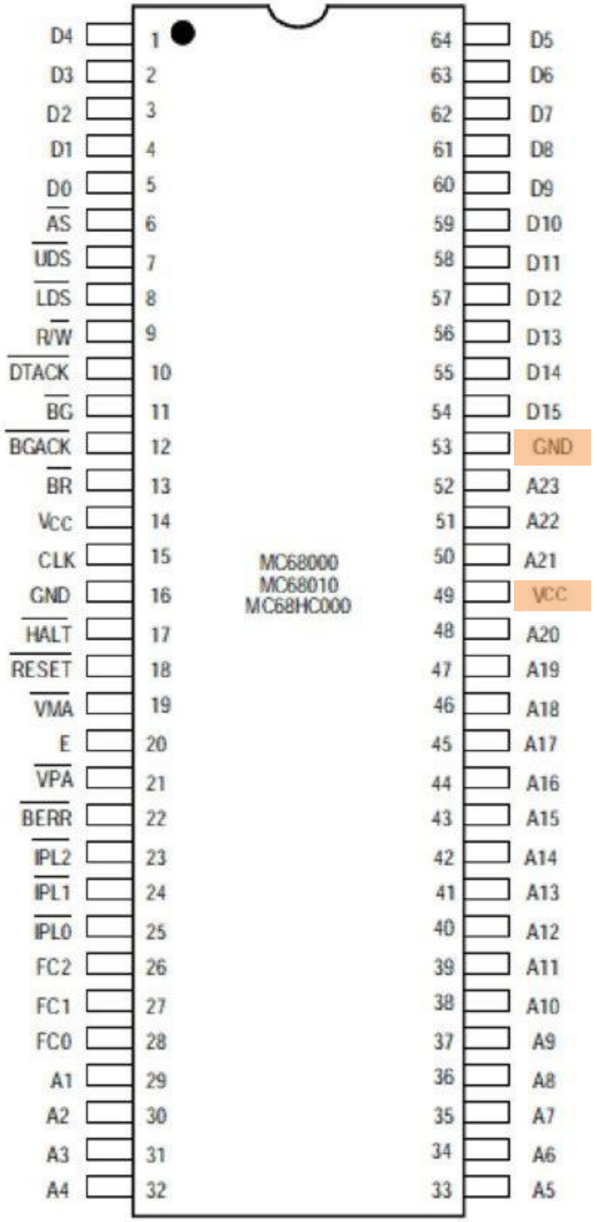
8. **Processor Function Codes (FC0-FC2)**
9. **Clock (CLK)**

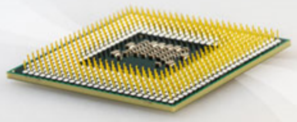




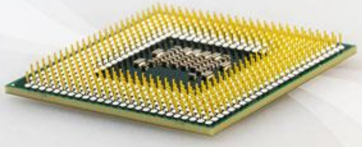
Pins and Signals

8. Processor Function Codes (FC0-FC2)
9. Clock (CLK)
10. Power Supply (V_{CC} and GND)



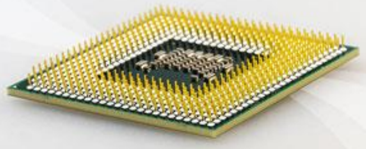


Registers



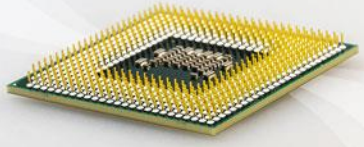
Name	Label	Number	Size	Function

Registers



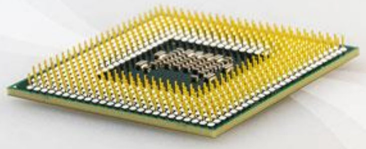
Name	Label	Number	Size	Function
Data registers	D0-D7	x8	32-bit	General purpose registers. Stores 8/16/32 bit data

Registers



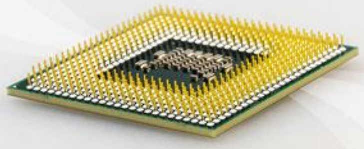
Name	Label	Number	Size	Function
Data registers	D0-D7	x8	32-bit	General purpose registers. Stores 8/16/32 bit data
Address registers	A0-A6	x7	32-bit	Stores 16/32 bit pointers (addresses of data)

Registers



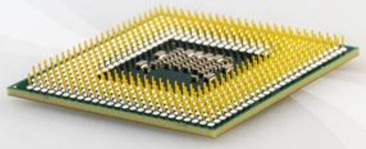
Name	Label	Number	Size	Function
Data registers	D0-D7	x8	32-bit	General purpose registers. Stores 8/16/32 bit data
Address registers	A0-A6	x7	32-bit	Stores 16/32 bit pointers (addresses of data)
Stack pointer	SP	x2	32-bit	Store a pointer to a group of data know as stack. Also known as A7.

Registers



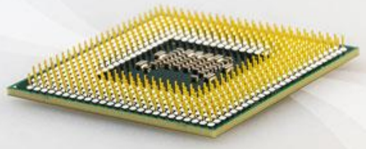
Name	Label	Number	Size	Function
Data registers	D0-D7	x8	32-bit	General purpose registers. Stores 8/16/32 bit data
Address registers	A0-A6	x7	32-bit	Stores 16/32 bit pointers (addresses of data)
Stack pointer	SP	x2	32-bit	Store a pointer to a group of data know as stack. Also known as A7.
Program counter	PC	x1	32-bit	Contains the address of next instruction to fetch and execute

Registers



Name	Label	Number	Size	Function
Data registers	D0-D7	x8	32-bit	General purpose registers. Stores 8/16/32 bit data
Address registers	A0-A6	x7	32-bit	Stores 16/32 bit pointers (addresses of data)
Stack pointer	SP	x2	32-bit	Store a pointer to a group of data know as stack. Also known as A7.
Program counter	PC	x1	32-bit	Contains the address of next instruction to fetch and execute
Status register	SR	x1	16-bit	Contains information on the results of the last instruction

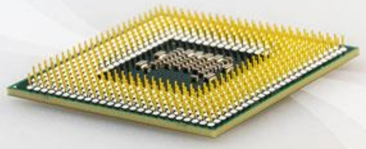
Registers



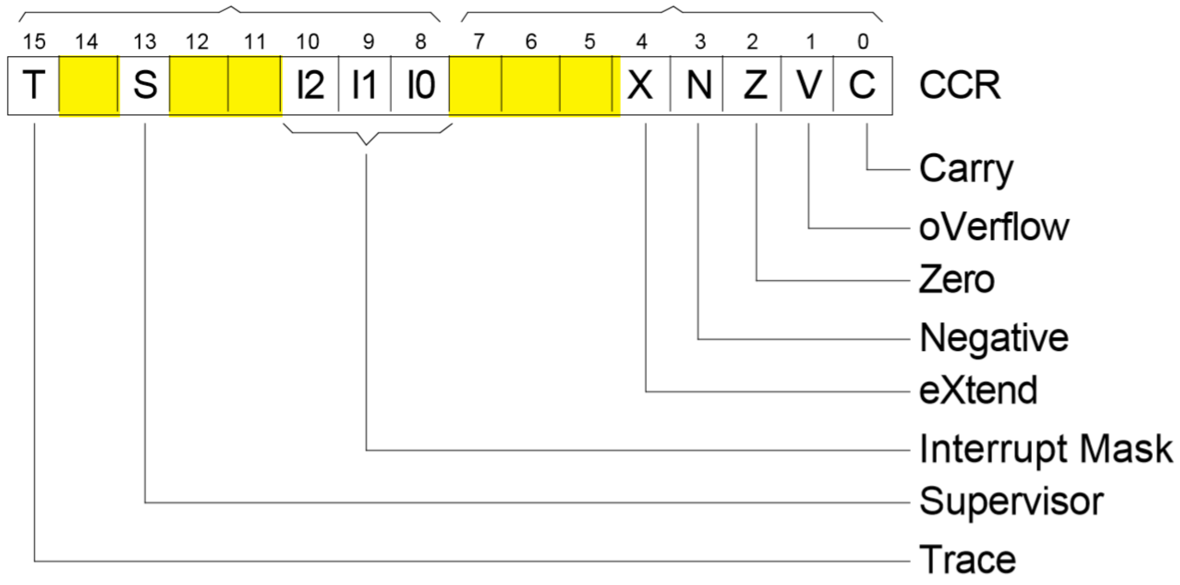
Status register:

- Only the low-order byte of the SR, which is called the CCR (Condition Code Register), can be accessed by the user.
- the so-called System Byte, can be seen and accessed only by the Operating System during special emergency cases
- The CCR allows conditional behavior
- The Control Unit often bases its decisions on the contents of the CCR
- Almost every instruction that is executed by the CPU forces an update on the value of one or more CCR bits

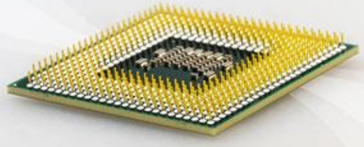
Registers



Setup of the Control/Status Register:

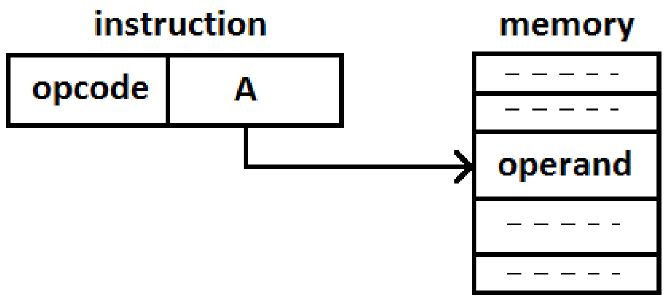


Addressing Modes



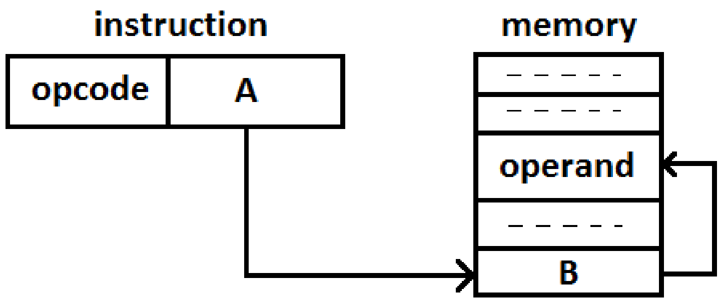
Addressing Modes

1. Direct addressing

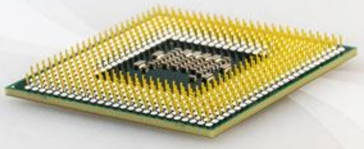


Addressing Modes

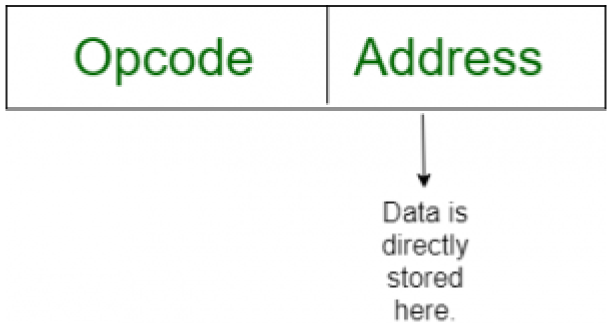
1. Direct addressing
2. Indirect addressing



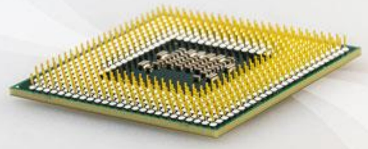
Addressing Modes



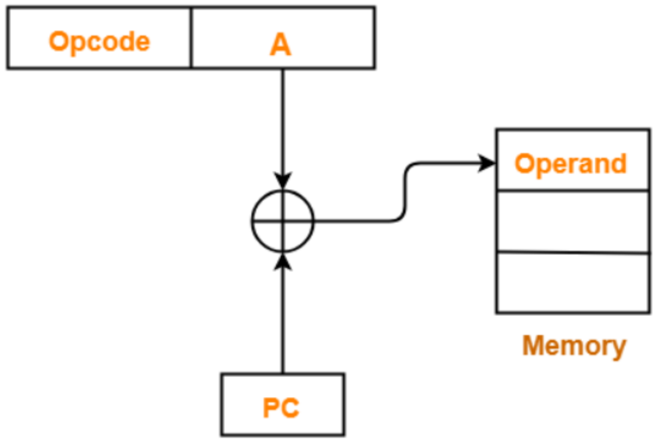
1. Direct addressing
2. Indirect addressing
3. Immediate addressing



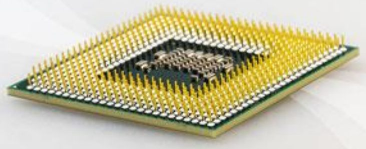
Addressing Modes



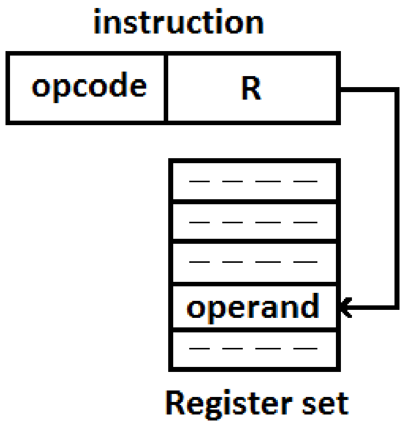
1. Direct addressing
2. Indirect addressing
3. Immediate addressing
4. Relative addressing



Addressing Modes

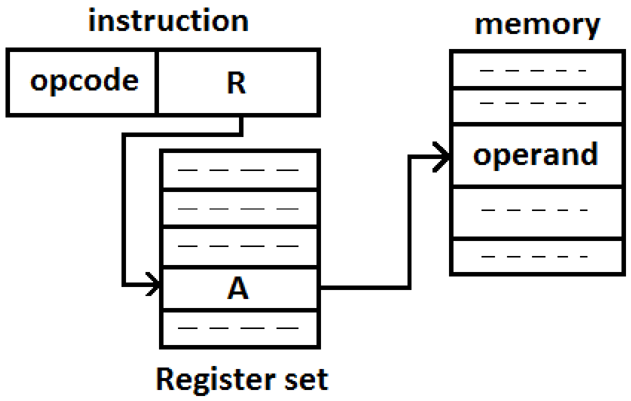


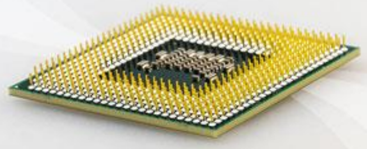
1. Direct addressing
2. Indirect addressing
3. Immediate addressing
4. Relative addressing
5. Register direct addressing



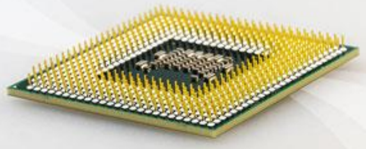
Addressing Modes

1. Direct addressing
2. Indirect addressing
3. Immediate addressing
4. Relative addressing
5. Register direct addressing
6. Register indirect addressing

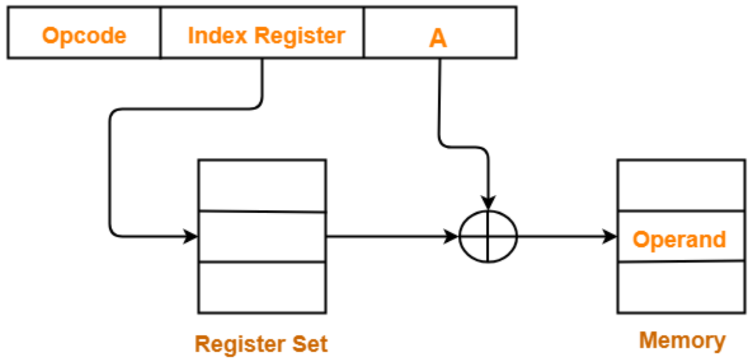




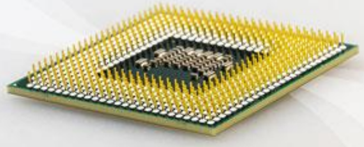
Addressing Modes



1. Direct addressing
2. Indirect addressing
3. Immediate addressing
4. Relative addressing
5. Register direct addressing
6. Register indirect addressing
7. Indexed addressing

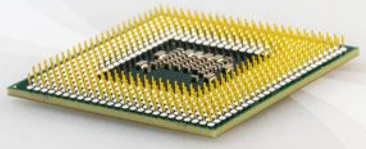


Addressing Modes



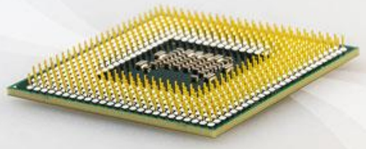
1. Direct addressing
2. Indirect addressing
3. Immediate addressing
4. Relative addressing
5. Register direct addressing
6. Register indirect addressing
7. Indexed addressing
8. Auto increment mode

Addressing Modes



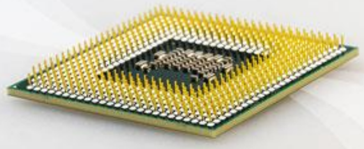
1. Direct addressing
2. Indirect addressing
3. Immediate addressing
4. Relative addressing
5. Register direct addressing
6. Register indirect addressing
7. Indexed addressing
8. Auto increment mode
9. Auto decrement mode

Addressing Modes



1. Direct addressing
2. Indirect addressing
3. Immediate addressing
4. Relative addressing
5. Register direct addressing
6. Register indirect addressing
7. Indexed addressing
8. Auto increment mode
9. Auto decrement mode
10. Inherent addressing

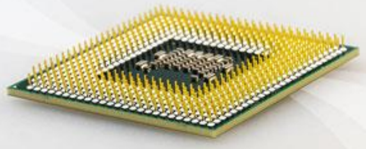
Instruction Set Architecture



Data Types:

- **Binary Digit (b)**
  - 1 bit stores either binary 0 or 1
- **Binary Coded Decimal (BCD)**
  - 4 bits that represents 0 to 9
- **Byte (B)**
  - 8 bits that is processed as one unit
- **Word (W)**
  - 16 bits that is processed as one unit
- **Long word (L)**
  - 32 bits that is processed as one unit

Instruction Set Architecture

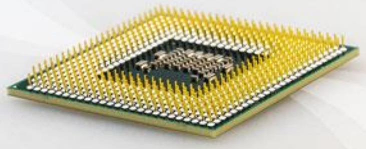


Instructions Categories:

Motorola 68000 instruction set is consist of 56 instructions in 8 different categories:

1. **Data transfer**
2. **Arithmetic**
3. **Logic**
4. **Shifts and rotates**
5. **Bit manipulation**
6. **BCD**
7. **Program control**
8. **System control**

Instruction Set Architecture



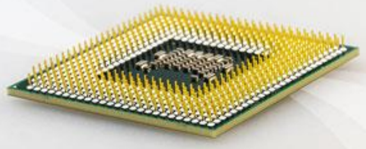
Data transfer instructions examples:

Instruction	Operation
MOVE <ea>,<e>	[destination] ← [source]
MOVEA <ea>,An	[An] ← [source]

Arithmetic instructions examples:

Instruction	Operation
NEG <ea>	[destination] ← 0 - [destination]
ADDI #<data>,<ea>	[destination] ← <literal> + [destination]
MULU <ea>,Dn	[destination] ← [destination] * [source]

Instruction Set Architecture



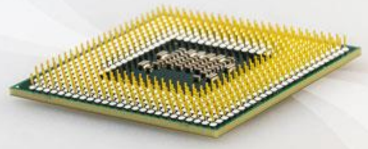
Logic instructions examples:

Instruction	Operation
NOT <ea>	[destination] ← [destination]
ANDI #<data>,CCR	[CCR] ← <data>.[CCR]

Shift & rotate instructions examples:

Instruction	Operation
LSR #<data>,Dy	[destination] ← [destination] shifted by <count>
ROL #<data>,Dy	[destination] ← [destination] rotated by <count>

Instruction Set Architecture



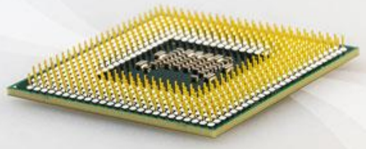
Bit manipulation instructions examples:

Instruction	Operation
BTST #<data>,<ea>	[Z] ← $\overline{\text{<bit number> OF [destination]}}$
CLR <ea>	[destination] ← 0

BCD instructions examples:

Instruction	Operation
ABCD Dy,Dx	[destination]10 ← [source]10 + [destination]10 + [X]
NBCD <ea>	[destination]10 ← 0 – [destination]10 - [X]
SBCD Dy,Dx	[destination]10 ← [destination]10 - [source]10 - [X]

Instruction Set Architecture



Program control instructions examples:

Instruction	Operation
Bcc <label>	If cc = 1 THEN [PC] ← [PC] + d
JMP <ea>	[PC] ← destination

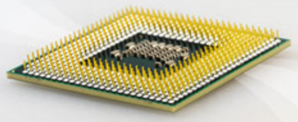
System control instructions examples:

Instruction	Operation
EORI #<data>,CCR	[CCR] ← <literal> ⊕ [CCR]

MOVE <ea>,SR	IF [S] = 1
	THEN [SR] ← [source]
	ELSE TRAP

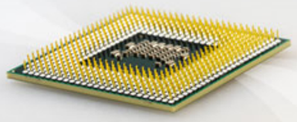
A complete list of instructions

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*



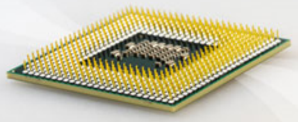
An example of Motorola 68K program

```
main
        move.l  #str,a0                          ;load A0 register with address of string
        movem.l a0,-(sp)                         ;push address of string on stack
        bsr     _puts                            ;branch to subroutine "_puts"
        bra     main                             ;keep looping!

        org     $2000
str     dc.b    'Hello, World!',10,0

        org     $3000
******
_puts                                            ;Like C/C++ puts function (LF added)
******                                           ;returns nothing
                                                 ;save regs
    movem.l d0-d1/d7/a0/a5/a6,-(sp)
    move.l  28(sp),a5                            ;get address of string from stack
                                                 ;find end of string
    move.l  a5,a6
1$ move.b  (a6)+,d0                              ;get next char of string
    cmp.b   #0,d0                                ;is it a null?
    beq 2$                                       ;yes, found end of string
    jmp 1$                                       ;no, so keep looping

2$ subq    #1,a6                                 ;don't print the null
    move.w  #227,d7                              ;call out1cr trap
    trap    #14

    ;retore regs & return
    movem.l (sp)+,d0-d1/d7/a0/a5/a6
    rts
END                                              ;end _puts
```

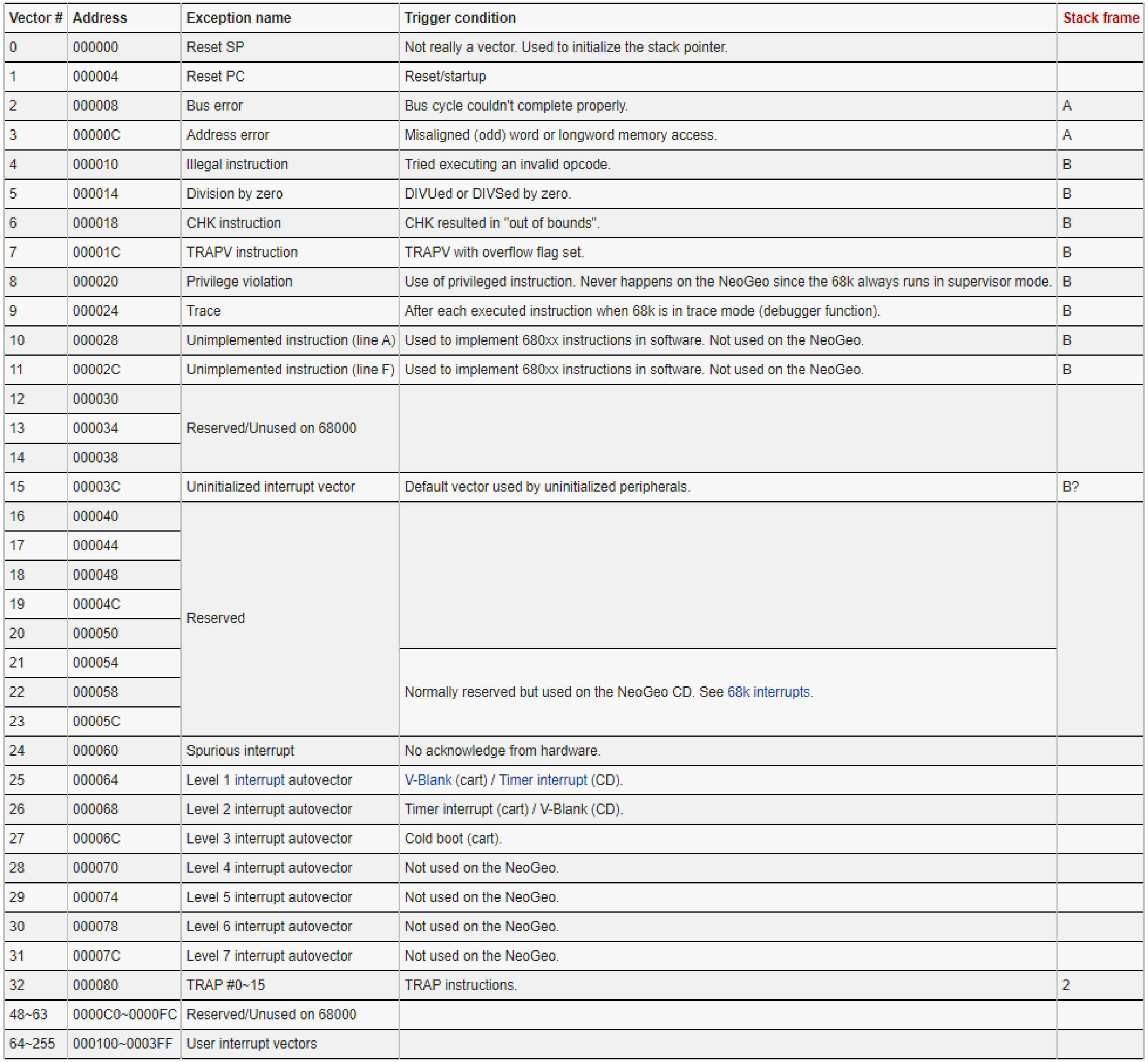


Another example of Motorola 68K program

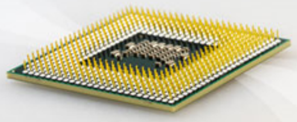
```
; strtolower:
; Copy a null-terminated ASCII string, converting
; all alphabetic characters to lower case.
;
; Entry parameters:
;    (SP+0): Source string address
;    (SP+4): Target string address

                org       $00100000       ;Start at 00100000
Strtolower      public
                link      a6,#0           ;Set up stack frame
                movea     8(a6),a0        ;A0 = src, from stack
                movea     12(a6),a1       ;A1 = dst, from stack
Loop            move.b    (a0)+,d0        ;Load D0 from (src), incr src
                cmpi      #'A',d0         ;If D0 < 'A',
                blo       copy            ;skip
                cmpi      #'Z',d0         ;If D0 > 'Z',
                bhi       copy            ;skip
                addi      #'a'-'A',d0     ;D0 = lowercase(D0)
copy            move.b    d0,(a1)+        ;Store D0 to (dst), incr dst
                bne       loop            ;Repeat while D0 <> NUL
                unlk      a6              ;Restore stack frame
                rts                       ;Return
                end
```



Motorola 68000
Exceptions Vector Table



Vector #	Address	Exception name	Trigger condition	Stack frame
0	000000	Reset SP	Not really a vector. Used to initialize the stack pointer.	
1	000004	Reset PC	Reset/startup	
2	000008	Bus error	Bus cycle couldn't complete properly.	A
3	00000C	Address error	Misaligned (odd) word or longword memory access.	A
4	000010	Illegal instruction	Tried executing an invalid opcode.	B
5	000014	Division by zero	DIVUed or DIVSed by zero.	B
6	000018	CHK instruction	CHK resulted in "out of bounds".	B
7	00001C	TRAPV instruction	TRAPV with overflow flag set.	B
8	000020	Privilege violation	Use of privileged instruction. Never happens on the NeoGeo since the 68k always runs in supervisor mode.	B
9	000024	Trace	After each executed instruction when 68k is in trace mode (debugger function).	B
10	000028	Unimplemented instruction (line A)	Used to implement 680xx instructions in software. Not used on the NeoGeo.	B
11	00002C	Unimplemented instruction (line F)	Used to implement 680xx instructions in software. Not used on the NeoGeo.	B
12	000030	Reserved/Unused on 68000		
13	000034			
14	000038			
15	00003C	Uninitialized interrupt vector	Default vector used by uninitialized peripherals.	B?
16	000040	Reserved		
17	000044			
18	000048			
19	00004C			
20	000050			
21	000054		Normally reserved but used on the NeoGeo CD. See 68k interrupts.	
22	000058			
23	00005C			
24	000060	Spurious interrupt	No acknowledge from hardware.	
25	000064	Level 1 interrupt autovector	V-Blank (cart) / Timer interrupt (CD).	
26	000068	Level 2 interrupt autovector	Timer interrupt (cart) / V-Blank (CD).	
27	00006C	Level 3 interrupt autovector	Cold boot (cart).	
28	000070	Level 4 interrupt autovector	Not used on the NeoGeo.	
29	000074	Level 5 interrupt autovector	Not used on the NeoGeo.	
30	000078	Level 6 interrupt autovector	Not used on the NeoGeo.	
31	00007C	Level 7 interrupt autovector	Not used on the NeoGeo.	
32	000080	TRAP #0~15	TRAP instructions.	2
48~63	0000C0~0000FC	Reserved/Unused on 68000		
64~255	000100~0003FF	User interrupt vectors		



Guys!

Do you really love 68000?

I have to tell you something...

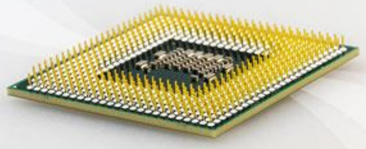
Further information

Watch on YouTube:

Motorola 68000 Oral History Panel

Scan me

References



(Kim , Muhammad Mun'im Ahmad Zabidi , Stritter and Gunter 1979, Starnes 1983, Clements 1992, Dávila 2000, Polsson 2006, Shvets 2018)

Clements, A. (1992). Microprocessor systems design: 68000 hardware, software, and interfacing, PWS Publishing Co.

Dávila, O. G. (2000). Assembly Language Macros

Kim, C. EECE416 Microcomputer Fundamental 68000 Processor, Howard University.

Muhammad Mun'im Ahmad Zabidi SEE 3223 Microprocessor Systems - 68000 Architecture. Kuala Lumpur, Malaysia, University Technology Malaysia: 6, 9, 11, 12, 17, 18, 21.

Polsson, K. J. O. D. (2006). "Chronology of Microprocessors."

Shvets, G. (2018). "Motorola 68000 microprocessor family." from http://www.cpu-world.com/CPUs/68000/.

Starnes, T. W. J. B. (1983). "Design philosophy behind Motorola's MC 68000. Part I: A 16-bit processor with multiple 32-bit registers." 8(4): 70-92.

Stritter, E. and T. J. C. Gunter (1979). "Microsystems a Microprocessor Architecture for a Changing World: The Motorola 68000." (2): 43-52.

Any Question?

Thanks for your kind attention!