CPU08 The CPU08 Central Processing Unit
M68HC08 and HCS08 microcontrollers

Introduction

- Central Processing Unit is one of the components that makes up a computer.
  - CPU
  - Memory Subsystem
  - Bus structure
- CPU is made up of Control and Execution units, and handles:
  - Arithmetic & logical operations
  - Code execution & Branching
  - Data transfer to & from memory
  - Interrupt management

Architecture of a Microcontroller
Topics of Discussion

- Features of CPU08
- Architecture Overview
- Addressing Modes
- A note on assemblers
- CPU08 Instruction Summary

Features of CPU08

- CPU08 is an 8-bit architecture
- 8-MHz CPU standard bus frequency
- 64-Kbyte memory space
- Fully object-code compatible with the M68HC05
- Instructions designed around stack manipulation
- 16 addressing modes
- Instructions capable of moving data from memory-to-memory without using the accumulator
- Extensible addressing range beyond the 64K boundary

Architecture Overview

- The CPU08 is an 8-bit architecture utilizing the following registers:
  - A - Accumulator
  - CCR - Condition Code Register
  - H - High byte of Index Register
  - X - Low byte of Index Register
  - PC - Program Counter
  - PCH - High byte of PC
  - PCL - Low byte of PC
  - SP - Stack Pointer
CPU08 Registers

- **Accumulator (8 bit)**
  - A general purpose 8-bit register used by the CPU to hold operands, or results of arithmetic or logical operations.

- **Index Register (16 bit)**
  - Can be accessed either as a 16 bit (H:X) register, or independently as two 8 bit H, or X registers.
  - Used by the CPU to index into, or address a 64KB memory range.

CPU08 Registers

- **Stack Pointer (16 bit)**
  - The Stack Pointer always points to the next available location of the stack.
  - Address decrements as data is moved (pushed) to the stack, and increments as data is moved out (pulled) from the stack.

- **Program Counter (16 bit)**
  - Holds the address of the next instruction to be fetched.
  - Automatically incremented, unless a jump or a branch takes place.
  - Is loaded with the contents of the Reset Vector at time of reset.

Condition Code Register

- The Condition Code Register is an 8-bit register which is comprised of the Interrupt Mask, and 5 flags which contain the results of the last executed command.

- **V – Overflow Flag**
  - Set when two's complement overflow occurs.
  - Set by ASL, ASR, LSL, LSR, ROL, ROR.
  - Used by BGT, BGE, BLE, BLT.
Condition Code Register

- **H** – Half-Carry Flag
  - Set when a carry occurs between bits 3 and 4 of the accumulator
  - The DAA uses the value of H to make appropriate adjustments (BCD)

- **I** – Interrupt Mask
  - When set, all interrupts are disabled
  - When an interrupt occurs, Interrupt Mask is automatically set
  - Further interrupts are latched until an RTI is issued

- **N** – Negative Flag
  - Set whenever an arithmetic or logical operation produces a negative result

- **Z** – Zero Flag
  - Set whenever an arithmetic or logical operation produces a zero result

- **C** – Carry or Borrow Flag
  - Set when an arithmetic operation produces a carry flag, or a subtraction operation requires a borrow
  - Some other operations such as bit test can also set or clear this flag

Control Unit

- The control unit is comprised of the Sequencer, the Control Store, and control logic
- It is primarily responsible for "preparing" the instruction for execution (i.e., fetching, decoding, address resolution)
- Sequencer provides the next state of the machine to the control store based on the contents of the Instruction Register (IR)
- Control store provides the next state condition to the Execution Unit (EU)
Instruction Execution

Addressing modes
- Inherent
- Immediate
- Direct
- Extended
- Indexed
  - No offset, post increment
  - 8-bit offset, post increment
  - 16-bit offset, post increment
- Stack Pointer
  - 8-bit offset
  - 16-bit offset
- Relative
- Memory to Memory (4 modes)

Inherent Addressing
- Have no operand associated with them (addresses are inherent)
- They require no memory address
  - INCA – Increment Accumulator
  - CLRA – Clear Accumulator
  - PSHH – Push Index High onto Stack
- Example
  - LDA $55 ; A = $55
  - INCA ; A = $56
Immediate Addressing

- Operand immediately follows the opcode
- Operand sizes are 1 or 2 bytes long
  - ADD – Add immediate value to Accumulator
  - AIX – Add immediate to Index register (H:X)
- Example
  - CLRH : H = 0
  - CLRX : X = 0
  - AIX #$1FF : H = $01
    : X = $FF (H:X = $1FF)

Direct Addressing

- Direct Addressing is limited to operands in the $0000 - $00FF area of memory (page 0)
- Operands are the low byte of the memory address, the high byte is assumed to be $00
- Fast execution – one clock cycle
- Important since most RAM is originated at $0000
- Examples are
  - Arithmetic Shifts (ASL, ASR, etc.)
  - Loads and stores (LDA, STA, etc.)

Extended Addressing

- Used to access any address in a 64K boundary
- Any address above direct or zero page requires this instruction
- Example
  - org $E000
    db $55
    ...
    Start:  LDA $E000 ; A = $55
Indexed Addressing

- Have 1, 2, or 3 operands for no offset, 8-bit, or 16-bit offset
- Indexed addressing is of great value as instructions can seamlessly move a pointer through a table
- Incrementing the pointer is done automatically through post-increment
- Using 8 or 16-bit offset, we can easily access any element in an array

Indexed Addressing Example

- The following example shows the three different modes of index addressing
  - LDA,x : no offset
  - LDA $FF,x : 8-bit offset
  - LDA $1000,x : 16-bit offset

Stack Pointer Addressing

- Indexing off of the Stack Pointer is done much the same way as Indexed Addressing
- If interrupts are off, this mode allows the Stack Pointer to be used as a second index register
Relative Addressing
- Relative addressing is done relative to the current value of the Program Counter (PC)
- The CPU automatically adds a signed offset (-128 to 127) to the PC
- The offset gives the relative address starting from the address location following the current branch instruction
- Only conditional branch instructions use this addressing mode

Memory to Memory Addressing
- Memory to Memory is expensive
  - It is a three-byte, four cycle operation
- But is less expensive than having to save the contents of the accumulator
- Moving Memory with Accumulator takes 9 cycles:
  - PSHA ; (2 cy) save contents of A
  - LDA #$55 ; (2 cy) load A with data
  - STA $10 ; (3 cy) save A into memory
  - PULA ; (2 cy) restore A
- Moving Memory with MOV takes 4 cycles:
  - MOV #$55, $10 ; (4 cy)

Memory to Memory cont.
- Moving memory using Direct to Direct mode also saves time
  - 10 cycles for doing so with accumulator
  - 5 cycles for doing so using MOV
- This is a valuable addressing mode because the contents of the accumulator are not changed
- Savings can be substantial with a lot of data movement
A Note on Assemblers

- The programmer generally does not need to keep in mind which mode of Index, Relative, or SP addressing mode to use.
- For instance, the assembler can automatically select no offset, 8-bit, or 16-bit Index addressing mode.
- The assembler can also calculate offsets for Relative Addressing mode, and verify that they are within range.

Resets and Interrupts

- CPU is designed in a way as to execute instructions sequentially.
- However, external events such as interrupts and resets are asynchronous to program execution and require special handling by the CPU.
- Reset is the mechanism by which we force (initialize) the CPU into a known state. This includes loading the Program Counter (PC) from a pre-defined non-volatile memory location to start execution from a known state.
- Interrupts temporarily suspend normal program execution and cause the CPU to switch context to service the interrupt.
- The CPU08 has up to 128 different interrupt sources.

Interrupts come in two flavors, maskable and non-maskable. The only non-maskable "interrupt" on the CPU08 is reset.

Interrupts on the CPU08 are prioritized according to a hardware defined priority scheme.

Once reset occurs, the 16-bit reset vector is fetched from $FFFE, and the stack pointer is set to $00FF.

Some reset sources are the RESET# pin, COP watchdog timer, Illegal opcode, and Low Voltage Inhibit (LVI).
Interrupt Processing is comprised of 5 steps:
- Recognition – Interrupt source must be unmasked
- Arbitration – Interrupt priority is evaluated, arbitrate between pending INTs, etc.
- Stacking – Saving the current state of the processor onto the stack
- Execution – Fetching the interrupt vector & handling it
- Returning – Restoring the state of the processor
### Selecting a Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Description</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT89C52</td>
<td>8-bit microcontroller</td>
<td>Low power, high speed</td>
</tr>
<tr>
<td>PIC16F84</td>
<td>8-bit microcontroller</td>
<td>Low cost, high performance</td>
</tr>
<tr>
<td>STM32F103</td>
<td>32-bit microcontroller</td>
<td>High performance, low power</td>
</tr>
<tr>
<td>LPC1768</td>
<td>32-bit microcontroller</td>
<td>High performance, high integration</td>
</tr>
</tbody>
</table>

![Table Image](image-url)
## Package & Product Numbering

### Table 1: Package & Product Numbering

<table>
<thead>
<tr>
<th>Package</th>
<th>Product</th>
<th>Function</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU08</td>
<td>1</td>
<td>Processor</td>
<td>1.0</td>
<td>Main CPU</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Memory</td>
<td>2.0</td>
<td>DDR3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Graphics</td>
<td>3.0</td>
<td>GPU</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Storage</td>
<td>4.0</td>
<td>SSD</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Network</td>
<td>5.0</td>
<td>Ethernet</td>
</tr>
</tbody>
</table>